A. Updated Concept Overview

SpartanSync is an LMS made for instructors, students, and teaching assistants to set course materials, assignments, announcements, and grading. The project has changed significantly since Milestone 1: from a basic prototype with a few stubs into a functional platform with different roles to choose from, database integrations, dashboards specific to each user type, and course/assignment management features.

The authentication and role-based access system is now complete, allowing users can register themselves as Students, TAs, or Instructors and have different UI components displayed according to their set role. Course creation, class selection, announcement management, and assignment creation are completed, thus instructors can manage the course content and work of students through a clean and simple dashboard.

Workflows of submission and grading have been partially implemented, with the ability for students to view assignments and submit work, and instructors and TAs to view submissions and access rubrics for the assignment. Rubric creation and grading are currently in development and expected to be complete in the next milestone.

B. Detailed Use Cases (6 Total)

Use Case 1: Instructor Creates an Assignment

Primary Actor: Instructor
Preconditions:

- Instructor is logged in.
- At least one course exists in the system.

Primary Sequence:

1. Instructor clicks New Assignment.
2. The system displays a form requesting assignment title, description, course, due date, and points.
3. Instructor clicks “Save Assignment”.
4. System validates the input with flask-wtf and creates the assignment in the database.
5. The system displays the assignment detail page.

Alternate Sequences:

- A1: Missing required field → system shows validation errors and redisplays the form.

Postconditions:

- A new assignment is stored in the database and appears in the course’s assignment list.

Use Case 2: Student Submits an Assignment

Primary Actor: Student
Preconditions:

- Student is logged in.
- The student is enrolled in the course for the assignment.
- Assignment exists and is published.

Primary Sequence:

1. The student opens the assignment detail page.
2. Student clicks Submit Work.
3. The system shows a submission form (text entry).
4. Student enters their work in the form.
5. System validates the form and stores the submission.
6. System displays a success message and then marks the assignment as submitted on the students' dashboard.

Alternate Sequences:

- A2: Student attempts a late submission → system displays a warning or prevents submission depending on instructor settings.

Postconditions:

- Submission is stored and associated with the student and the assignment.

Use Case 3: Instructor/TA Grades a Submission Using the Rubric - IN PROGRESS

Primary Actor: Instructor or TA
Preconditions:

- Instructor/TA is logged in.
- At least one student submission exists for the assignment.
- Rubric items exist.

Primary Sequence:

1. Instructor/TA navigates to the assignment detail page.
2. They open the list of student submissions.
3. They select one submission to grade.
4. System displays rubric rows with score inputs.
5. Instructor/TA enters scores based on the rubric and optional feedback.
6. The system calculates the total score and saves the grade for the student.
7. System updates the submission status to “graded.”

Alternate Sequences:

- A2: Instructor attempts to publish without full rubric completion → system warns the user

Postconditions:

- Graded submission with rubric scores is stored and visible on the students' dashboard.

Use Case 4: Instructor Posts a Course Announcement

Primary Actor: Instructor
Preconditions:

- Instructor is logged in.
- The course exists.

Primary Sequence:

1. Instructor clicks “New Announcement” from the nav bar.
2. The system displays a form requiring a title, body text, and course selection.
3. Instructor submits the form.
4. System validates the data and creates the announcement.
5. Announcement appears in the announcements list and on the course overview.

Alternate Sequences:

- A1: Missing entries in the dorm → system displays validation errors.
- A2: Instructor selects “General” instead of a specific course → announcement appears site-wide.

Postconditions:

- The new announcement is stored and displayed to all students enrolled in the selected course.

Use Case 5: Student Selects Courses for Their Dashboard

Primary Actor: Student
Preconditions:

- Student is logged in.
- The course catalog contains available courses.

Primary Sequence:

1. Student navigates to Manage My Classes.
2. System displays all available courses with multi-select options.
3. The student selects or deselects courses.
4. Student saves their changes.
5. System updates the student’s chosen course list.
6. Dashboard updates to show only selected courses.

Alternate Sequences:

- A1: Student attempts to save without selecting any course → system warns them but allows saving.

Postconditions:

- Dashboard course list reflects the student's selections.

Use Case 6: TA Edits Assignment Rubric but Cannot Publish/Delete Assignments - IN PROGRESS

Primary Actor: Teaching Assistant (TA)
Preconditions:

- TA is logged in.
- Assignment exists.
- Rubric items exist or rubric editor is enabled.

Primary Sequence:

1. TA opens the assignment detail page.
2. TA clicks Edit Rubric.
3. System displays existing rubric rows with fields for criteria, descriptions, and point values.
4. TA modifies rubric items and saves changes.
5. System validates entries and updates the rubric.

Alternate Sequences:

- A1: Missing criteria fields → system prompts the TA to correct errors.
- A2: TA attempts to access "Delete Assignment" → system denies access.
- A3: TA attempts to publish an assignment → system displays an unauthorised access error.

Postconditions:

- Rubric is updated, but assignment publish/delete privileges remain restricted to instructors.

C. Implementation Summary

MVP Features Completed

- User authentication (login, signup, logout)
- Role-based access system for Student, TA, and Instructor
- Course creation and course catalog browsing
- Student class selection/dashboard customisation
- Assignment creation
- Announcement creation and listing

- Dashboards for Students, TAs, and Instructors
- Automatic SQLite database creation

Features In Progress

- Rubric creation/editing system
- Submission + grading (partially implemented)

Architecture and Design Changes Since M1

- Modified the database models to support assignments, submissions, announcements, roles, and rubrics.
- Introduced session-based role access checks across routes to enforce Student/TA/Instructor permissions.
- Adjusted rubric and submission models to support more flexible grading logic.
- Improved template structure to include role-sensitive navigation and dashboards.
- Restructured pages to connect courses → assignments → submissions more cleanly.

D. Testing Plan

We created and implemented a testing plan to verify the functionality of the LMS. All tests were written using test and are inside a tests/ directory with separate files for models, forms, and routes.
Structure:

```
tests/
├── conftest.py        # Fixtures and shared configuration
├── config.py          # Test database and environment settings
├── test_models.py     # Model tests
├── test_forms.py      # Form validation tests
├── test_routes.py     # Route and permission tests
└── __init__.py
```

Coverage:

- Models
    - Validated db model creation, constraints, foreign keys, default values, password hashing, timestamps, role handling, and assignment, submission, rubric associations
- Forms
    - Checked required fields, validation logic, input formatting rules, correct error handling for login, account creation, course and assignment creation, class selection, submissions, announcements and rubric criteria.
- Routes
    - Verified the http endpoints, page access, redirects, login/logout, course and assignment operations, dashboard views, permissions, and proper handling of authenticated and not authenticated requests

Test environment:

- We used reusable fixtures in conftest that provide a test setup, like an in memory sqlite db, test users, sample courses, assignments, etc, and a test flask client to interact with routes and sessions.

We made 69 total tests, and 69 tests passed.
The LMS behaves as we expected and this testing reduces our risk of regressions throughout our next additions.