VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Software Engineering

Assignment 1:

Tutor Support System at HCMUT - Phase 1

Advisor(s):	Trần Trương Tuấn Phát	
Group:	7	
Student(s):	Võ Hoàng Sơn	2353057
	Trần Hải Đăng	2352254
	Lâm Minh Tùng Quân	2352991
	Phan Quốc Đại Sơn	2353053
	Trương Gia Kỳ Nam	2352787
	Nguyễn Hữu Phúc	2352938
	Châu Anh Nhật	2352856

HO CHI MINH CITY, NOVEMBER 2025

Contents

Name	Task	% Complete
Võ Hoàng Sơn	Design activity diagram for Tutor, Department Chair, Program Admin. Design sequence diagram for Tutor Course Feedback. Design state chart of Assignment Lifecycle	100%
Trần Hải Đăng	Design activity diagram for Student, Coordinator. Design sequence diagram for Handling feedback and Reports, Searching and Importing Libary References. Design state-chart for Session Lifecycle	100%
Lâm Minh Tùng Quân	Design sequence diagram for Deadline and Assignemenet management, Session Management, Session request handling, Chat and Notification System, Tutor Evaluation and Feedback.	100%
Phan Quốc Đại Sơn	Design sequence diagram for Sign In process, Course access and Resource Retrieval, Course Matching and Assignment Process, Learning Material Management.	100%
Trương Gia Kỳ Nam	Design user interface - Login, Calendar, Course Dashboard and Integrated chat	100%
Nguyễn Hữu Phúc	Design user interface - Course detail and submission management, My Course overview and Report Module, Tutor Registration and Matching Module	100%
Châu Anh Nhật	Design user interface - System Dashboard, HCMUT library, Eavaluation and report module	100%

1 User Interface (UI) Design: Mockup

1.1 Overview

The User Interface (UI) mockup of the **Tutor Support System (TSS)** is designed to provide an intuitive, accessible, and visually consistent experience for all user roles, including Students, Tutors, Coordinators, Department Chairs, and Program Administrators. Following the system objectives defined in Phase 1, the mockup emphasizes functionality, clarity, and integration with HCMUT's digital infrastructure such as *HCMUT_SSO*, *HCMUT_DATACORE*, and *HCMUT_LIBRARY*.

Developed with a human-centered design approach, the mockup aims to support users in accomplishing academic and administrative tasks efficiently while maintaining a familiar and cohesive visual language aligned with HCMUT's identity. Each UI element is crafted to ensure smooth task flows, reduce cognitive load, and enhance usability across both desktop and mobile devices.

1.2 Key UI screens

1.2.1 Login Interface – HCMUT SSO Authentication

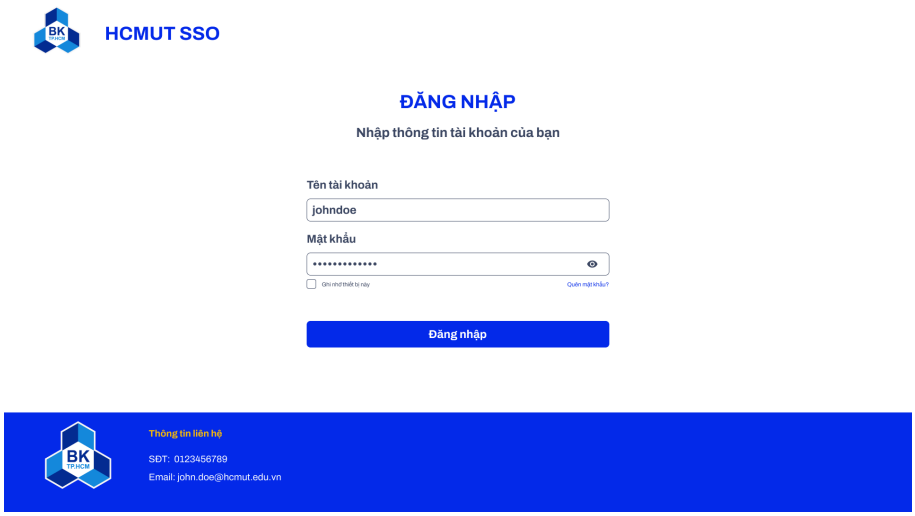


Figure 1.1: Login Interface – Standard State

Purpose: The Login Interface serves as the secure entry point for all user roles within the Tutor Support System (TSS). It integrates with the *HCMUT_SSO* service to ensure unified authentication and centralized credential management across the university's digital ecosystem.

Features:

- **Visual Design:** A minimal, professional interface that uses the university's signature blue palette.
- **Branding Consistency:** The BK TPHCM hexagonal logo is incorporated to reinforce institutional identity.
- **Focused Layout:** The login form is centered on a clean white background to maximize clarity and reduce distractions.
- **Localized Fields:** *Username* and *Password* inputs are clearly labeled in Vietnamese, aligning with the university's bilingual guidelines.

User Flow: When a user enters credentials, the system forwards them to *HCMUT_SSO* for verification. If valid, a session token is issued and the user is redirected to their corresponding dashboard (Student, Tutor, Coordinator, etc.). If authentication fails, inline red validation messages are displayed without page reload, preserving usability and minimizing user frustration.

1.2.2 Calendar Interface

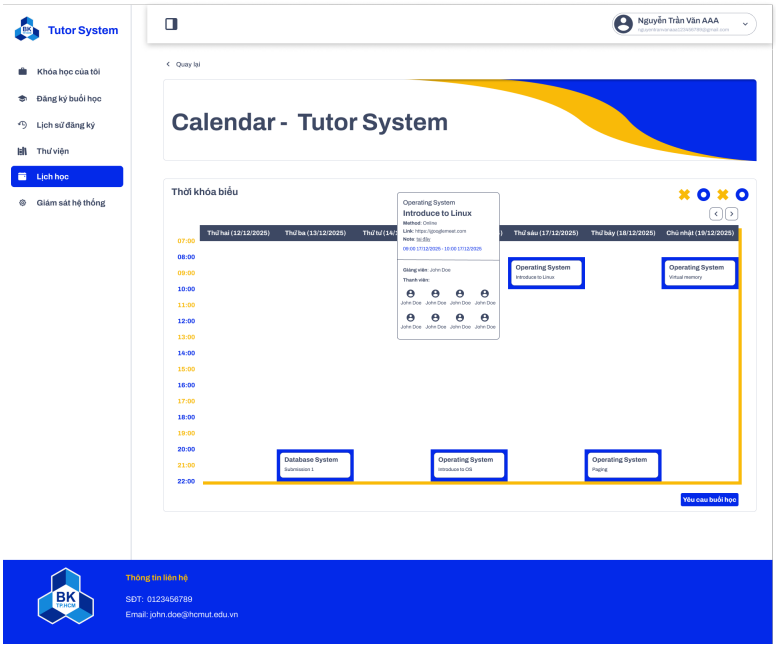


Figure 1.2: Students Calendar Interface

Purpose: The Calendar Interface is the central workspace where tutors and students can view, manage, and interact with their deadlines and tutoring sessions.

Features:

- **Weekly Overview:** Displays all deadlines and tutoring sessions for the selected week, with quick navigation to previous and next weeks.
- **Session Requests:** Allows students to request new tutoring sessions directly from the calendar when additional support is needed.

User Flow:

- Students view their personal calendar. By default, it shows the current week; navigation arrows move between weeks.
- Students click a session block to join the class or access its materials.
- Tutors create sessions by completing a form. After saving, the session appears on the student's calendar.

1.2.3 Course Dashboard and Integrated Chat Interface

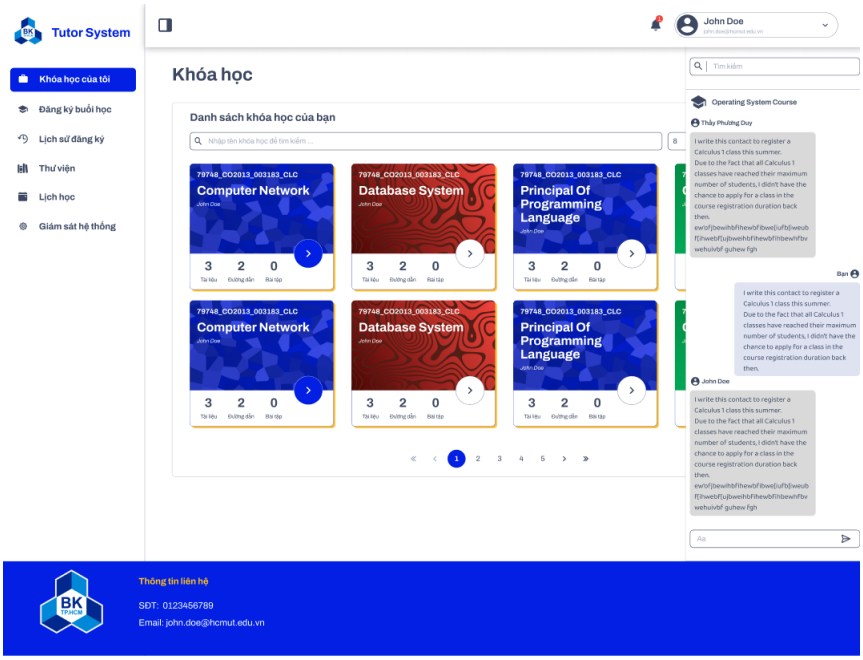


Figure 1.3: Chat Interface – Tutor–Student Messaging Detail

Purpose: The **Course Dashboard** serves as the central hub for students and tutors to manage their enrolled or assigned courses within the Tutor Support System (TSS). It consolidates course access, learning materials, notifications, and real-time communication, ensuring a cohesive environment for teaching and learning interactions.

Features:

- Role-Based Display: Automatically loads course information according to the user's role (Student or Tutor).
- Dynamic Course Grid: Presents all enrolled or assigned courses as interactive cards with course codes, titles, and tutor names.
- Search and Filtering: Supports keyword search, pagination, and sorting to navigate extensive course lists efficiently.
- Notification Drawer: Centralizes course-related updates and announcements for easy access.
- Integrated Chat Panel: Enables real-time messaging between tutors and students for feedback, clarification, and academic discussions.

User Flow:

- After login, the user is redirected to the Course Dashboard.
- The system retrieves and displays all enrolled or assigned courses as interactive cards.
- The user can search, sort, or paginate through the list to find a specific course.
- Clicking on a course card opens its detailed view with access to learning materials and the chat interface.
- The Chat Panel allows tutors and students to exchange messages, share feedback, and coordinate sessions in real time.
- Notifications update dynamically to reflect new course activities or messages.

1.2.4 Course Materials Interface

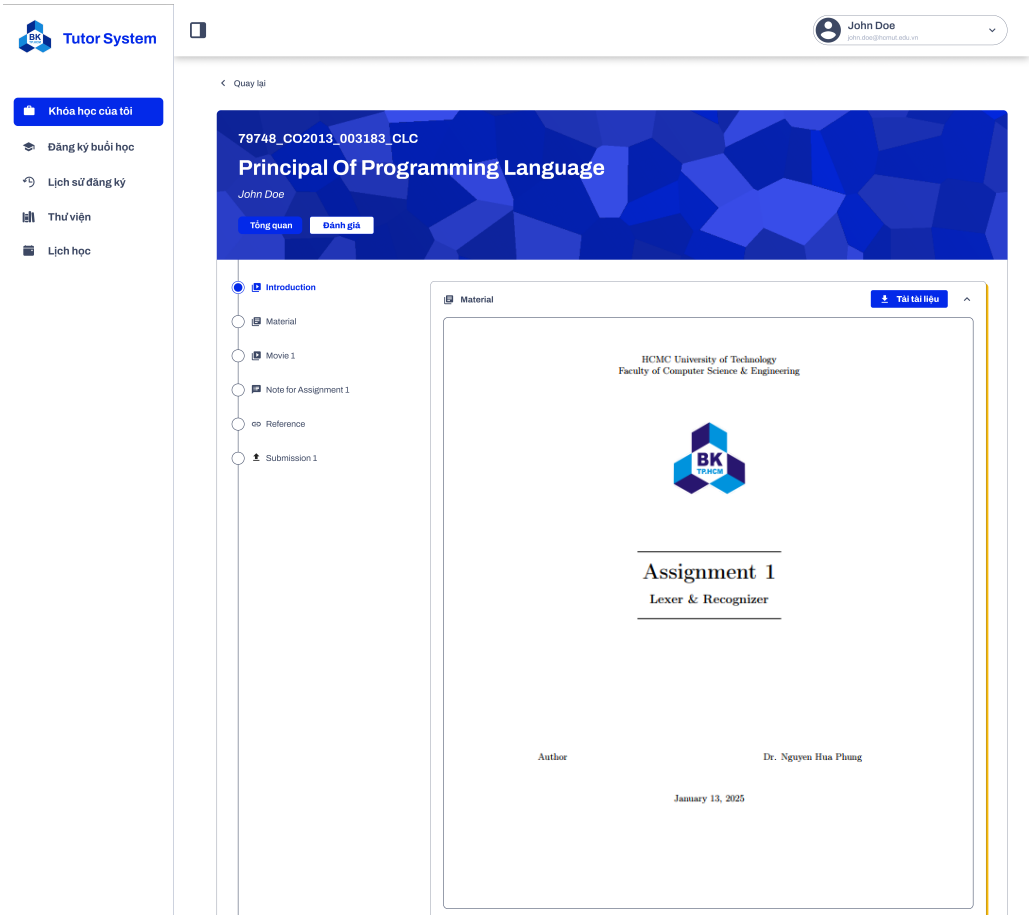


Figure 1.4: Students Course Materials Interface

Purpose: The Course Materials Interface enables tutors and students to interact with course's materials such as file, video, note, hyperlink and library's material.

Features:

- **Material Catalog:** Lists all materials in a course, organized by topic and type for quick scanning.
- **Document Viewer & Downloads:** Previews documents in-browser and allows downloading of the original files.
- **External Links:** Opens hyperlink materials in a new tab, taking users directly to the referenced page.

- **Library Resources:** Deep-links to the library catalog item or its digital copy (when available).
- **Content Management (Tutor):** Supports uploading and editing of course materials with clear, guided forms.

User flow:

- For student:
  1. **Select Course** : The student navigates from "Khóa học của tôi" and clicks on a specific course to view its materials
  2. **Navigate Content** : The student uses the left-hand table of contents to select a topic
  3. **Consume Materials** : The main panel updates, allowing the student to view document, download file, watch embedded videos, access external link and check assignment detail
- For tutor:
  1. **Access course** : The tutor navigates to the course materials page
  2. **Manage Content** : The tutor adds or edits sections in the navigation panel
  3. **Upload Materials** : For each section, the tutor uploads files, embeds videos, adds links, and writes notes for the students.
  4. **Publish** : The tutor saves the changes, making the materials instantly available to students

1.2.5 My Course Overview and Report Module

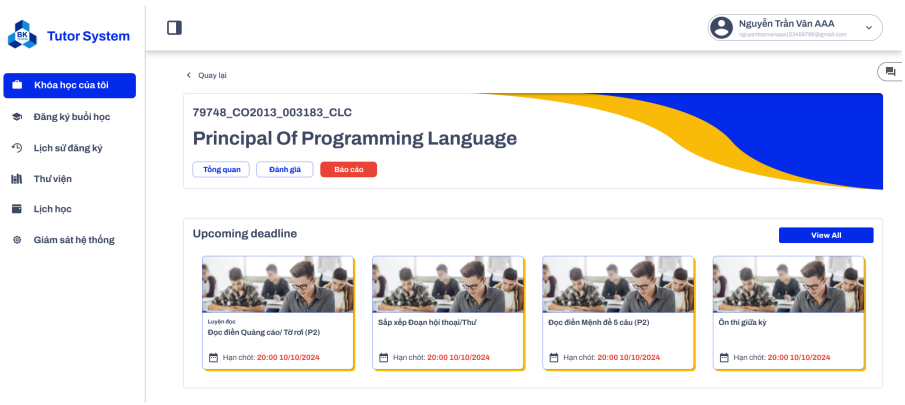


Figure 1.5: My Course Overview – Tutor and Coordinator

Purpose: The My Course Overview and Report Module together form the analytical and managerial layer of the Tutor Support System (TSS). This interface supports both tutors and coordinators in monitoring course participation, upcoming activities, and issue reporting. It combines real-time visualization of deadlines, session scheduling, and student lists with a dedicated channel for submitting feedback or technical reports.

Features:

- **Overview Cards:** Shows *Upcoming Deadlines*, *Upcoming Sessions*, and *Recently Viewed Materials* in a scannable card layout.
- **Student Roster:** A table of enrolled students with email addresses and registration timestamps; supports search, filtering, and pagination.
- **Report Modal:** A one-click *"Báo cáo"* dialog that overlays the page for submitting issues or improvement suggestions inline.
- **Integrated Chat:** An optional side panel to view active chat threads while browsing the page, enabling in-context coordination.
- **Filtering and Pagination:** Users can search by name or email and navigate multiple student pages efficiently.
- **Dynamic Reporting Modal:** Accessible through the *"Báo cáo"* button, it overlays the page and allows inline submission of feedback.

- **Integrated Communication:** Tutors can simultaneously view chat threads while browsing the same page, promoting seamless coordination.

User flow:

1. User accesses the course overview via the navigation sidebar.
2. The dashboard loads all upcoming sessions and materials.
3. Tutors or coordinators can monitor student lists and click "Báo cáo" to submit issues.
4. A modal dialog opens for entering the report text and confirming submission.
5. The system forwards the report to the coordinator's view for review and resolution.

1.2.6 Registration and Matching Module

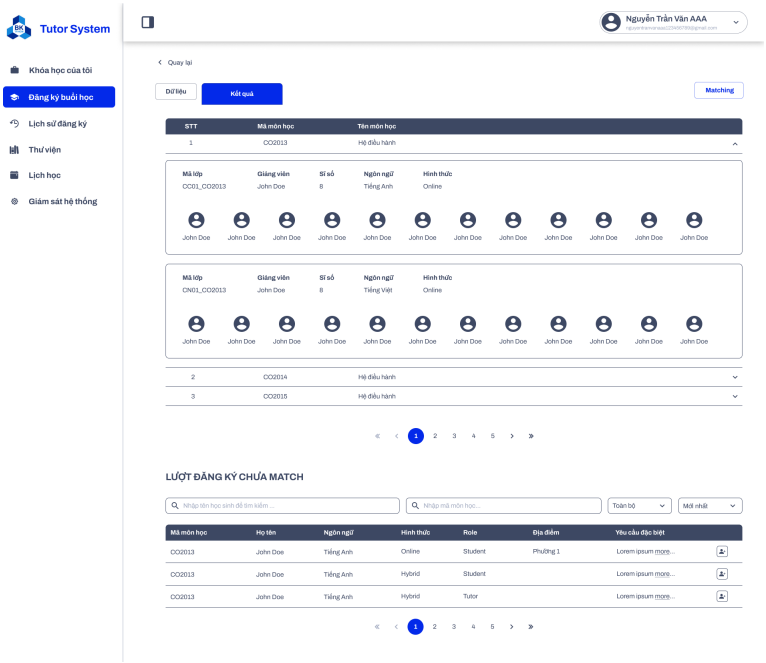


Figure 1.6: Course Matching Result

Purpose: This module manages the registration and pairing process for tutoring programs.

- Students register for available tutor-led courses and specify preferences.
- Tutors submit teaching details such as subjects, languages, and delivery modes.

- Coordinators oversee registration periods and monitor submissions, while the system's AI Matching Service automatically pairs tutors and students based on predefined criteria.

Features:

- Role-Based Forms: Separate registration interfaces for Students, Tutors, and Coordinators.
- Deadline Management: Coordinators define opening and closing dates through a scheduling modal.
- AI Matching: The system filters and pairs participants automatically according to subject, language, and schedule.
- Status Tracking: Registrations display color-coded tags (Pending, In Review, Matched).
- Multilingual Support: Drop-down menus in English or Vietnamese enhance accessibility.

User Flow:

1. Students and tutors complete their respective registration forms.
2. The system validates and saves submissions.
3. Coordinators review and manage registration timelines.
4. The AI Matching Service pairs tutors and students automatically.
5. Matched records are updated in the database, while unmatched ones remain for later review.

1.2.7 HCMUT Library Integration Interface

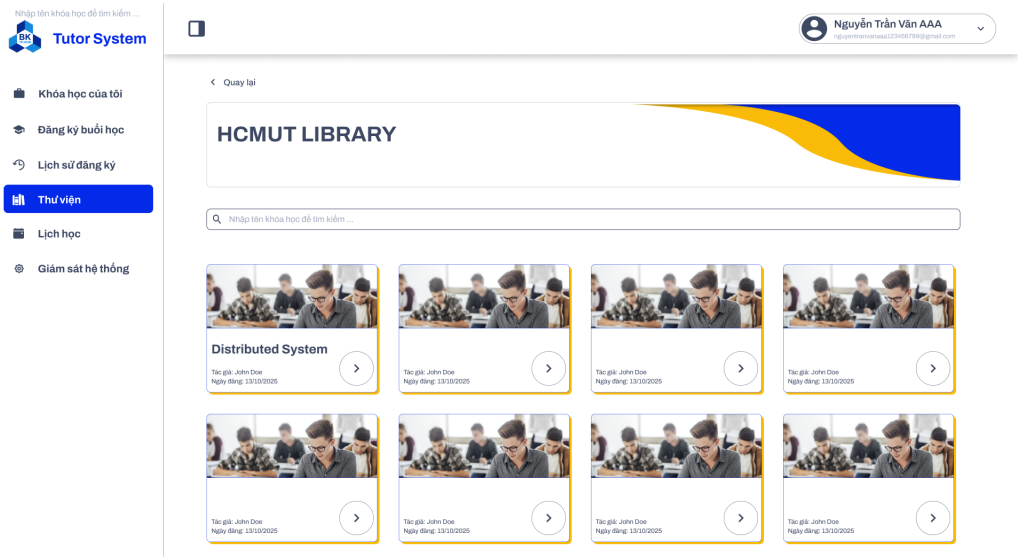


Figure 1.7: Library Interface

Purpose The HCMUT Library Interface links the Tutor Support System (TSS) with the university's centralized digital library, enabling tutors and students to access academic resources directly within the platform. It allows users to search, preview, and download course-related materials such as textbooks, lecture slides, and notes without leaving the system. This integration ensures a seamless transition between tutoring activities and official institutional learning resources hosted on HCMUT_LIBRARY.

Features:

- Search Functionality: Supports keyword-based queries by title or author to filter relevant materials.

User Flow:

1. The user selects Library from the sidebar.
2. The system loads and displays available academic resources.
3. The user enters a keyword in the search bar to find specific materials.
4. The results update dynamically with pagination for navigation.
5. Selecting a resource card opens detailed information or initiates a secure download.

1.2.8 Evaluation and Report Module

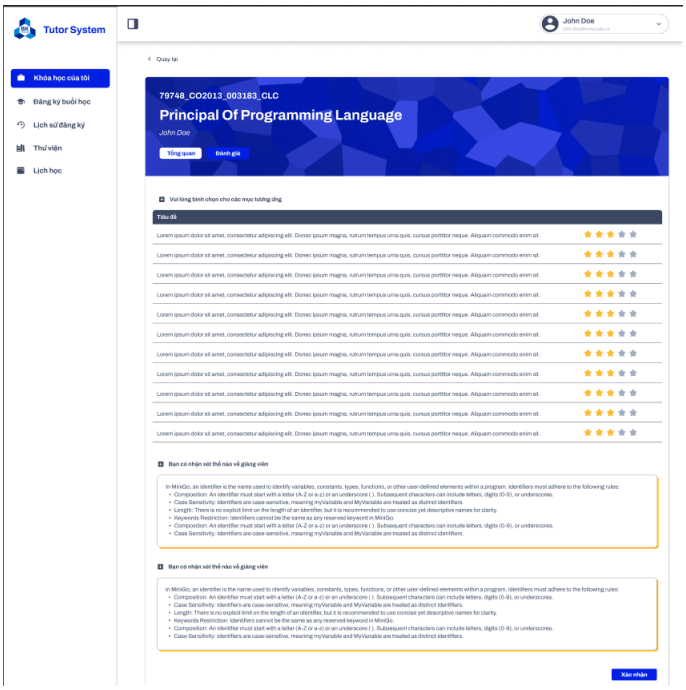


Figure 1.8: Evaluation Interface – Student Perspective with Rating Criteria

Purpose: The **Evaluation and Report Module** provides a structured mechanism for collecting, displaying, and managing feedback among students, tutors, and coordinators. It supports both qualitative comments and quantitative scoring, enabling continuous monitoring of session quality and participant performance across the tutoring program. This module ensures transparency, accountability, and systematic quality improvement in tutoring activities.

Features:

- **Star-Rating Component:** Provides 1–5 scale feedback with visual cues for clarity.
- **Role-Adaptive Interface:** UI content dynamically changes according to the user's account type.
- **Comment Section:** Enables text-based qualitative feedback from both tutors and students.
- **Pagination and Filtering:** Coordinators can search and navigate through multiple evaluations per course.

- **Consolidated Report:** Data collected from multiple participants are displayed in a structured coordinator dashboard for trend analysis.

User Flow:

1. The session concludes, prompting the user to open the evaluation form for that course.
2. Students submit ratings and written feedback about tutors.
3. Tutors reciprocate by evaluating each assigned student.
4. The coordinator accesses all related data through the report dashboard.
5. Consolidated insights are exported for program-level quality tracking and improvement planning.

1.2.9 Course Analytics Module

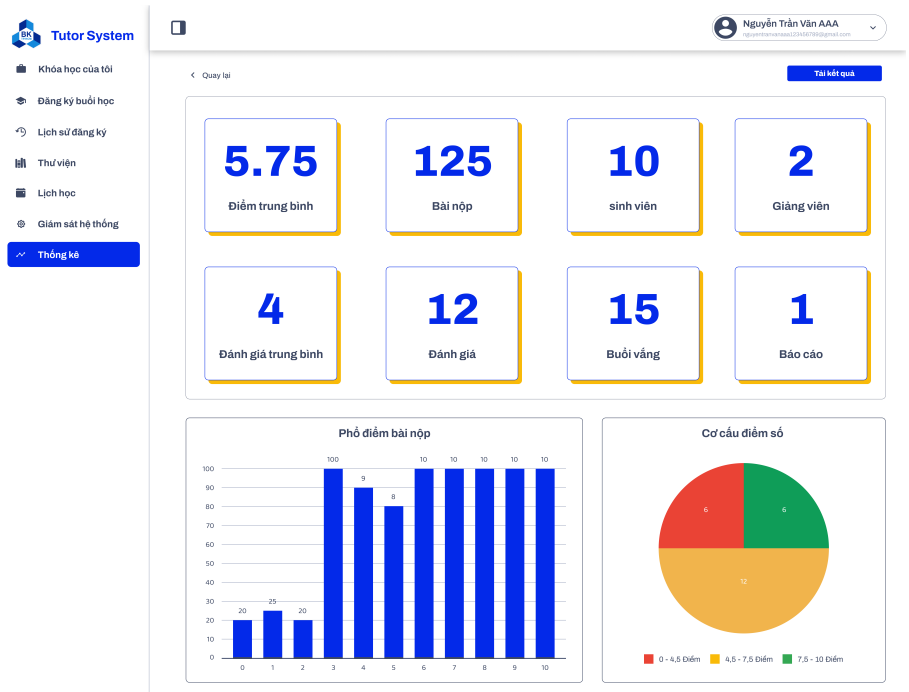


Figure 1.9: Course Analytics

Purpose: The Analytics Module provides coordinators with visual insights into course performance, student participation, and tutor activity. It enables quick evaluation of academic outcomes and helps coordinators identify areas that require support or adjustment.

The system automatically aggregates data from submissions, attendance, and assessments to ensure accuracy and reduce manual work.

Features:

- Course Dashboard Overview: Displays summarized metrics such as average score, total submissions, number of students and tutors, average rating, total evaluations, and attendance records.
- Interactive Charts: Bar and pie charts illustrate grade distribution, rating breakdown, and participation levels for each course.
- Activity Timeline: Shows historical engagement trends over time, allowing coordinators to track progress throughout the semester.
- Filtering and Comparison: Coordinators can select specific courses from their managed list to analyze results and compare performance metrics.
- Data Export: Allows downloading summarized reports in standard formats for departmental review or archival.

User Flow:

1. The coordinator accesses the Statistics section from the sidebar.
2. The system displays a list of managed courses with summary cards (course name, tutor, number of students, and progress).
3. The coordinator selects a course to view detailed analytics.
4. The dashboard loads visual reports including average score, rating distribution, attendance, and submission statistics.
5. The coordinator can export data or return to the course list to view another course.

1.2.10 System Monitoring Module

Figure 1.10: System Monitoring Interface

Purpose: The System Monitoring Module allows coordinators and administrators to track the system's real-time performance, ensuring stability and reliability. It provides live metrics on server health, enabling early detection of performance degradation or resource overload that may affect user experience.

Features:

- Resource Dashboards: Displays real-time gauges for CPU, memory, disk, and swap usage.
- Performance Charts: Line graphs visualize CPU and memory consumption trends over time.
- Dynamic Updates: Data refreshes automatically at short intervals for continuous monitoring.
- Usage Segmentation: Graphs distinguish between user-related and system-level activity (e.g., Busy User vs. Busy I/O).

- Visual Alerts: Color indicators highlight high-usage conditions, helping administrators respond promptly.

User Flow:

1. The coordinator or administrator selects the matching section from the sidebar.
2. The dashboard displays key usage indicators for CPU, memory, disk, and swap.
3. Real-time charts show historical performance trends to assess system load.
4. When abnormal resource usage occurs, visual gauges change color to alert the user.
5. Administrators use the data to optimize resource allocation or perform maintenance.

2 Sequences diagrams

2.1 Sequence Diagram – Sign-In Process

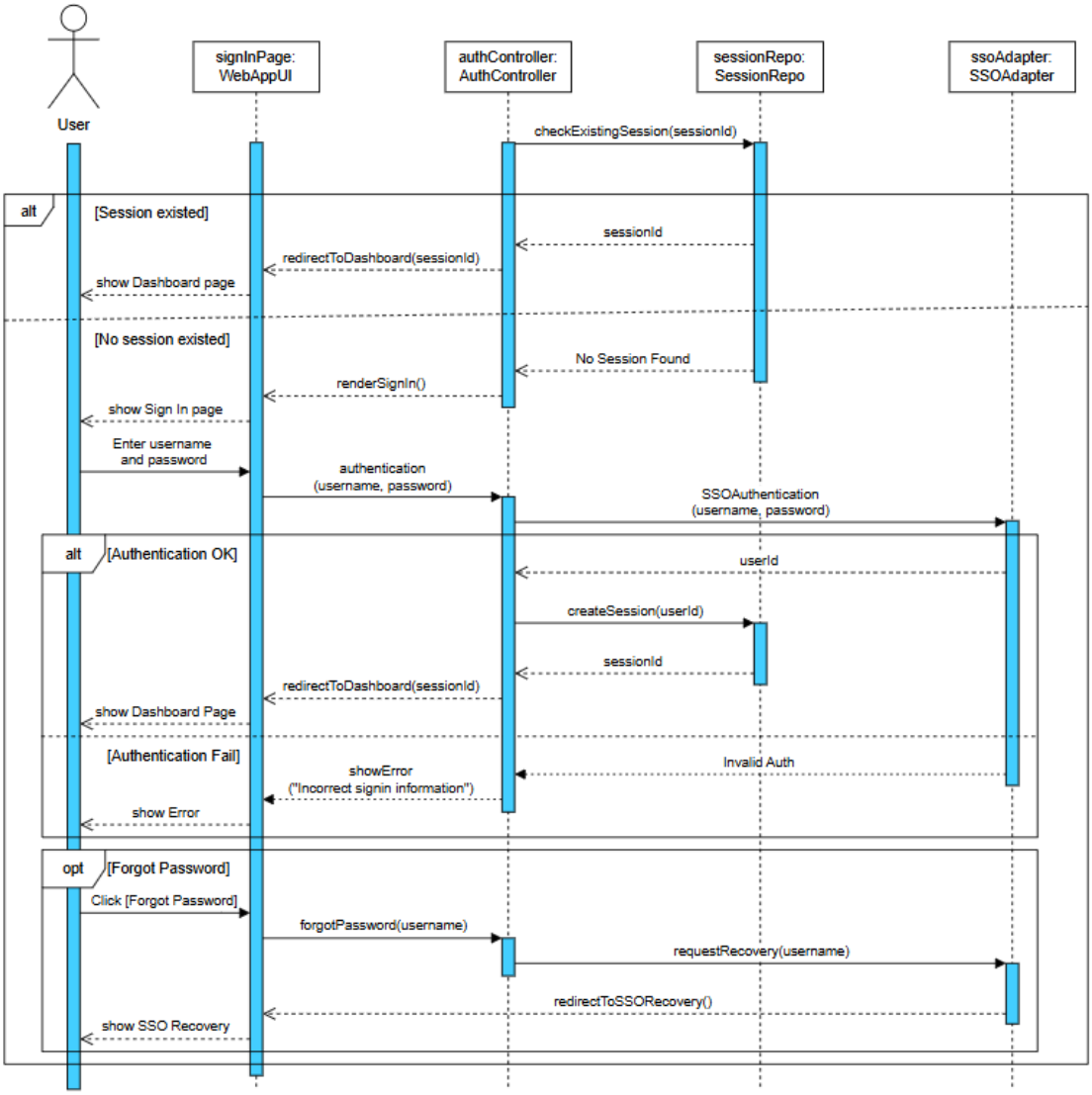


Figure 2.1: Sequence Diagram – Sign-In Process with HCMUT SSO Integration

Actors and Lifelines

- **User:** Initiates the sign-in process through the web interface.
- **signInPage: WebAppUI** – Frontend boundary class responsible for collecting user credentials and displaying responses.

- **authController: AuthController** – Core control component managing the authentication flow and SSO communication.
- **sessionRepo: SessionRepo** – Entity component that stores and validates active user sessions.
- **ssoAdapter: SSOAdapter** – External interface responsible for communicating with HCMUT's centralized SSO service.

Main Flow The diagram consists of three logical segments:

1. **Session Validation:** The controller first checks for an existing session via checkExistingSession(sessionId). If a valid session is found, the user is redirected directly to the dashboard.
2. **Authentication:** When no session exists, the system renders the sign-in page. Upon user input, the controller forwards credentials to `SSOAdapter` for validation through `SSOAuth(username, password)`. Successful authentication triggers `createSession(userId)` and redirects to the dashboard.
3. **Error and Recovery Handling:** The **alt** fragments describe possible exceptions:
   - **Authentication Fail:** Invalid credentials return an "Incorrect sign-in information" error.
   - **Forgot Password (opt):** Users may trigger password recovery, which redirects them to the official HCMUT SSO recovery portal.

2.2 Sequence Diagram – Course Registration Process

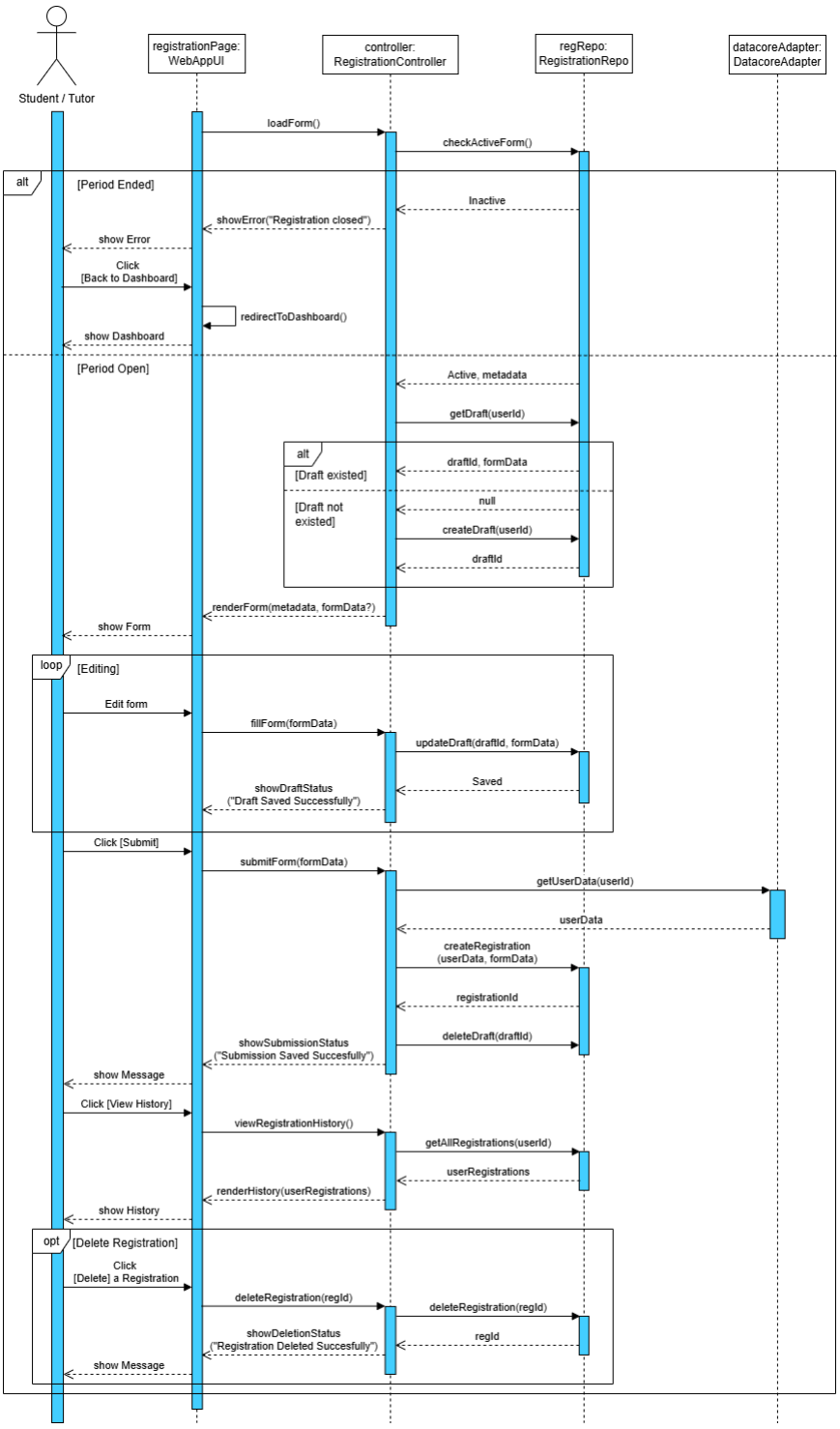


Figure 2.2: Sequence Diagram – Course Registration Workflow for Students and Tutors

Actors and Lifelines

- **Student / Tutor:** Initiates the registration, updates personal information, and submits registration forms.
- **registrationPage: WebAppUI** – Frontend boundary interface responsible for displaying forms, handling input, and showing feedback messages.
- **controller: RegistrationController** – Control layer orchestrating registration logic, including validation, form lifecycle, and communication with repositories.
- **regRepo: RegistrationRepo** – Entity component managing registration data persistence (drafts, submissions, deletions).
- **datacoreAdapter: DatacoreAdapter** – External service connector retrieving verified user and course metadata from the institutional data core system.

Main Flow The process consists of multiple logical stages:

1. **Form Loading:** When the user accesses the registration interface, the system calls `checkActiveForm()`. If no active registration period is detected, an error message "Registration closed" is displayed, and the user is redirected to the dashboard.
2. **Draft Retrieval:** During an active registration phase, the controller retrieves any existing draft with `getDraft(userId)`. If none exists, a new draft is initialized via `createDraft(userId)` before rendering the editable form.
3. **Form Editing (loop):** Users repeatedly edit and save draft data through fillForm(formData). The repository confirms draft persistence with feedback messages like "Draft Saved Successfully".
4. **Form Submission:** Once satisfied, users click `Submit`. The controller fetches verified profile data via `getUserData(userId)` from `DatacoreAdapter` and proceeds to create the final registration entry through `createRegistration(userData, formData)`. The system deletes the corresponding draft and displays "Submission Saved Successfully".
5. **Viewing History:** The user can retrieve all previous registration records via getAllRegistrations(userId). The results are rendered as a chronological list for review.

6. **Optional Deletion:** Through the **opt** fragment, users may delete a previous registration entry. The controller confirms deletion with the message "Registration Deleted Successfully".

2.3 Sequence Diagram – Course Assignment Process

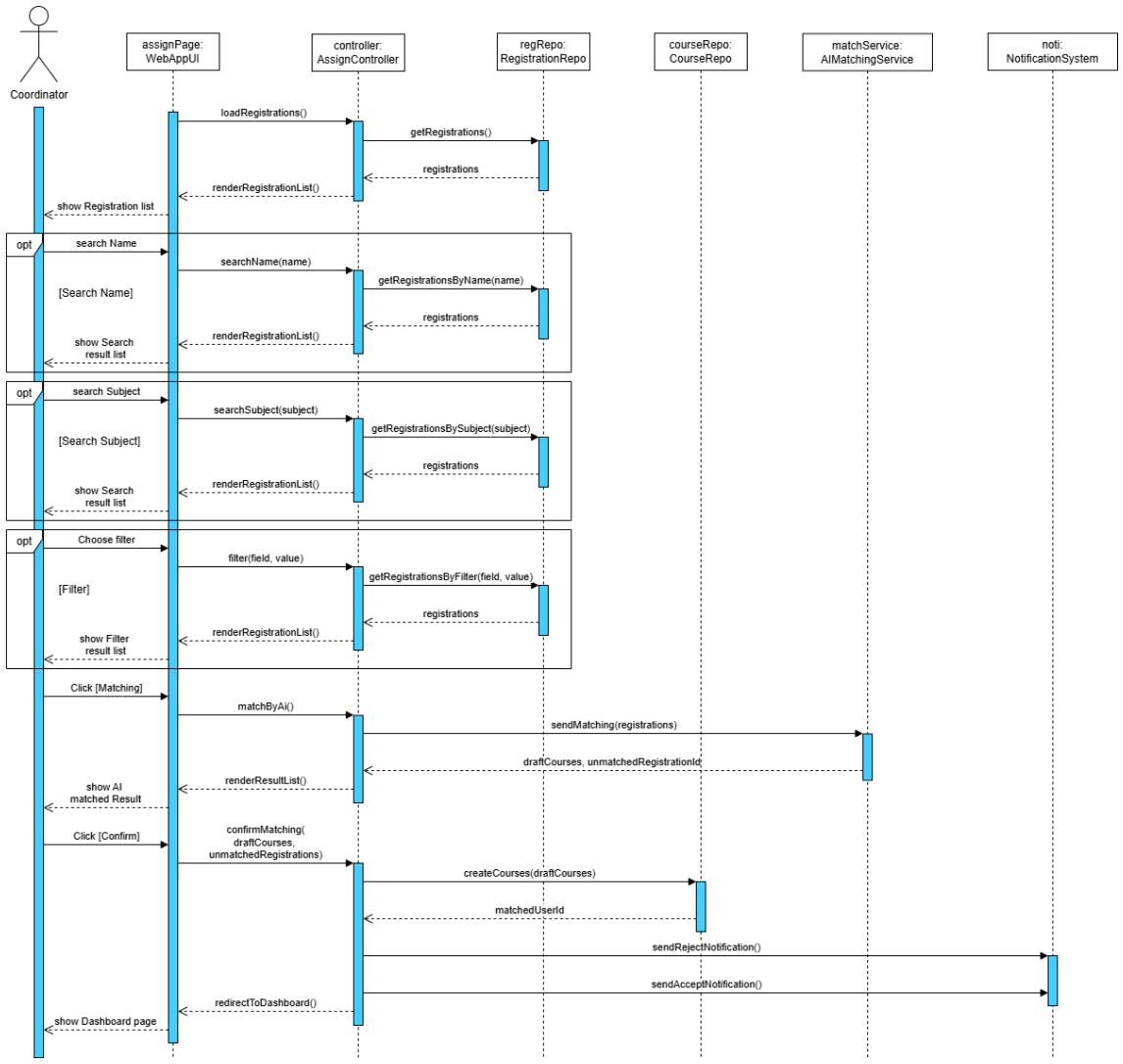


Figure 2.3: Sequence Diagram – Coordinator Matching and Course Assignment Process

Actors and Lifelines

- **Coordinator:** The user responsible for reviewing and confirming tutor–student pairings.

- **assignPage: WebAppUI** – Boundary class that displays the list of registration records, AI recommendations, and confirmation results.
- **controller: AssignController** – Control layer that orchestrates search, filter, match, and confirmation operations.
- **regRepo: RegistrationRepo** – Entity repository storing student and tutor registration records.
- **courseRepo: CourseRepo** – Entity responsible for course creation once a valid tutor–student pair is confirmed.
- **matchService: AIMatchingService** – External service performing automatic pair suggestions using predefined AI matching algorithms.
- **noti: NotificationSystem** – External system handling notification delivery (email and in-app) for assignment results.

Main Flow

1. **Load and Display Registrations:** The coordinator accesses the assignment interface, triggering `loadRegistrations()` which retrieves all pending registrations from `RegistrationRepo`. The results are rendered as a searchable list via `renderRegistrationList(`
2. **Filtering and Searching (opt):** Multiple optional flows allow refinement of the registration list:
   - **Search by Name:** Executes `getRegistrationsByName(name)`.
   - **Search by Subject:** Executes `getRegistrationsBySubject(subject)`.
   - **Filter by Criteria:** Executes `getRegistrationsByFilter(field, value)`.

   Each query updates the displayed registration list.
3. **AI-Assisted Matching:** Upon clicking `Matching`, the controller calls `matchByAI()` to send all selected registrations to `AIMatchingService`. The service returns a set of draft courses and unmatched records, which are displayed for coordinator review.
4. **Confirmation and Course Creation:** The coordinator reviews AI-suggested results and confirms matches through `confirmMatching()`. The controller invokes `createCourses(draftCourses)` in `CourseRepo`, producing finalized course instances and mapping associated tutor and student user IDs.

5. **Notification Dispatch:** After successful creation, `AssignController` triggers the `NotificationSystem` to inform users:
   - Accepted pairings receive `sendAcceptNotification()`.
   - Rejected or unmatched participants receive `sendRejectNotification()`.
6. **Redirection:** Finally, the interface redirects the coordinator to the dashboard page via `redirectToDashboard()`.

2.4 Sequence Diagram – Learning Material Management

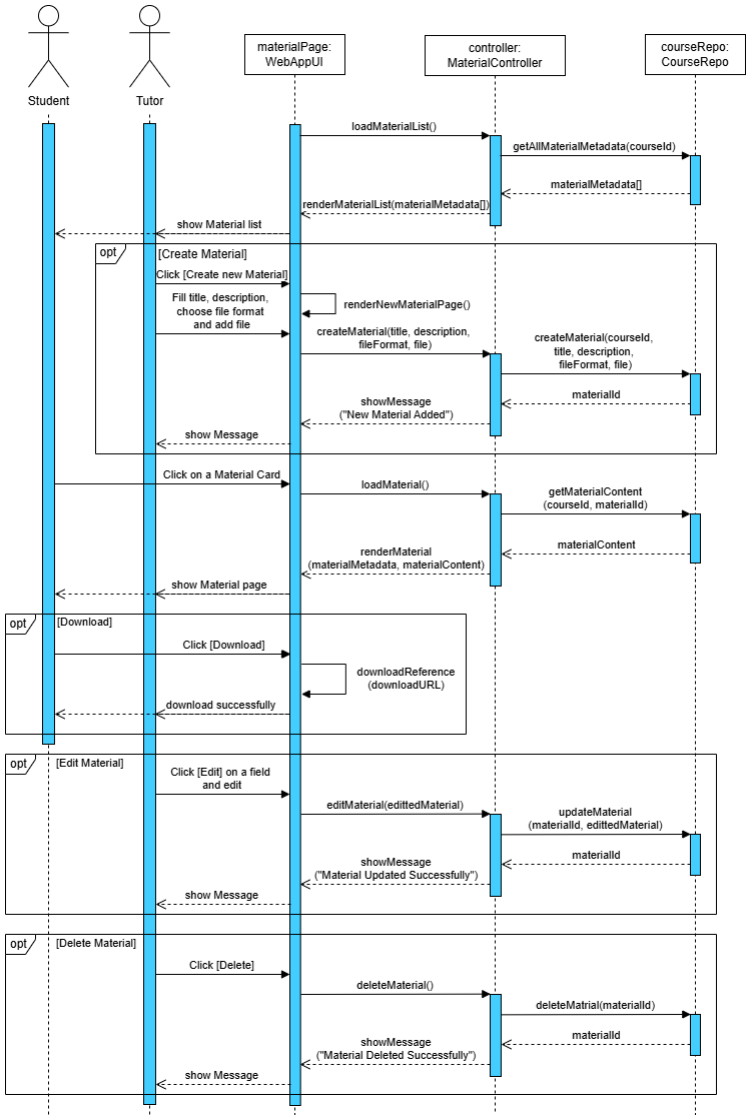


Figure 2.4: Sequence Diagram – Learning Material Management for Tutors and Students

Actors and Lifelines

- **Student:** Views and downloads learning materials shared within the enrolled course.
- **Tutor:** Creates, edits, and deletes course materials relevant to assigned subjects.
- **materialPage: WebAppUI** – Boundary interface displaying available materials, enabling uploads, downloads, and modifications.
- **controller: MaterialController** – Control component coordinating material-related operations, including validation and repository communication.
- **courseRepo: CourseRepo** – Entity repository responsible for persisting material metadata and file references within each course.

Main Flow

1. **View Material List:** When a user accesses the course's material section, the `materialPage` invokes `loadMaterialList()`, triggering getAllMaterialMetadata(courseId) in CourseRepo. The retrieved metadata array is rendered via `renderMaterialList(materialMe`
2. **Create New Material (opt):** Tutors may initiate the creation of new materials by clicking `Create New Material`. After filling in details such as title, description, and file format, the system calls `createMaterial(courseId, title, description, fileFormat, file)`. Upon success, the controller displays a confirmation message *"New Material Added"*.
3. **View Material Details:** When the user selects a specific material card, the UI executes `loadMaterial()`. The controller retrieves detailed content using `getMaterialContent(co materialId)` from the repository and renders it via `renderMaterial(materialMetadata, materialContent)`.
4. **Download Material (opt):** Users can download files by clicking the `Download` button. The UI requests a `downloadReference(downloadURL)` and provides feedback once the download is complete.
5. **Edit Material (opt):** Tutors can modify existing materials by selecting the `Edit` option. The controller calls `updateMaterial(materialId, editedMaterial)` to persist changes and returns a success message *"Material Updated Successfully"*.

6. **Delete Material (opt):** Tutors may remove outdated or incorrect resources via `deleteMaterial(materialId)`. The repository confirms deletion and the controller displays *"Material Deleted Successfully"*.

2.5 Sequence Diagram – Deadline and Assignment Management

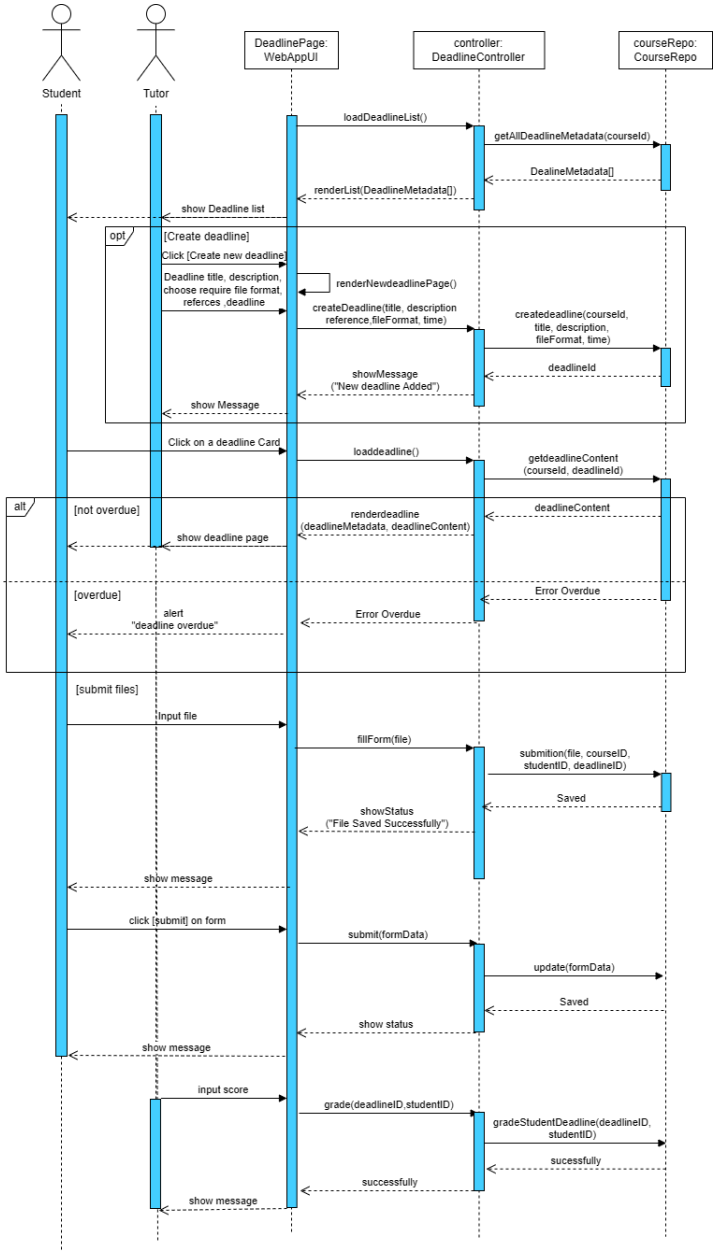


Figure 2.5: Sequence Diagram – Deadline and Assignment Management for Students and Tutors

Actors and Lifelines

- **Student:** Submits assignments, checks deadlines, and downloads reference files.
- **Tutor:** Creates, edits, grades, and deletes deadlines, as well as evaluates student submissions.
- **DeadlinePage: WebAppUI** – Boundary class handling user interactions for creating, editing, and viewing deadlines or submissions.
- **controller: DeadlineController** – Control layer that coordinates operations related to deadlines, drafts, and grading.
- **courseRepo: CourseRepo** – Entity repository responsible for persisting deadline metadata, file references, and grading results.

Main Flow

1. View Deadline List: The user opens the deadline page, and the UI triggers loadDeadlineList(). The controller retrieves data through getAllDeadlineMetadata(courseId) from CourseRepo, rendering the list of upcoming deadlines via renderList(deadlineMetadata[]).
2. **Create Deadline (opt):** Tutors may create a new deadline by entering the title, description, file format, references, and due time. The UI calls createDeadline(courseId, title, description, fileFormat, time), and after successful creation, the controller displays the confirmation message *"New Deadline Added"*.
3. **View Deadline Details:** When a student or tutor clicks a specific deadline card, the system loads its content using getDeadlineContent(courseId, deadlineId). The diagram contains an **alt** fragment distinguishing two scenarios:
   - **[Not Overdue]:** Deadline details are displayed normally.
   - **[Overdue]:** The system raises an alert message *"Deadline Overdue"*.
4. **Download Reference (opt):** Students can retrieve attached files by triggering downloadReference(downloadURL), receiving feedback once the file download succeeds.
5. **Edit or Delete Deadline (opt):** Tutors may update existing deadlines through editDeadline(editedDeadline) or remove them with deleteDeadline(deadlineId). Both operations confirm success through messages: *"Deadline Updated Successfully"* or *"Deadline Deleted Successfully"*.

6. **Submission Workflow (loop):** Students may iteratively fill and update their submissions using fillForm(formData). Saved drafts are confirmed with *"Draft Saved Successfully"*, whereas unsaved attempts prompt *"Draft Not Saved"*. Once ready, the student submits the final work through submit(deadlineId, formData), and the controller validates success or connection errors accordingly.

7. **Grading Workflow (loop):** Tutors assign scores using grade(deadlineId, studentId). If missing or invalid data is detected, the system returns an alert *"Missing Data"*. Otherwise, the controller updates grades through gradeStudentDeadline(deadlineId, studentId) and displays a success notification *"Grade Successfully"*.

2.6 Sequence Diagram – Chat System

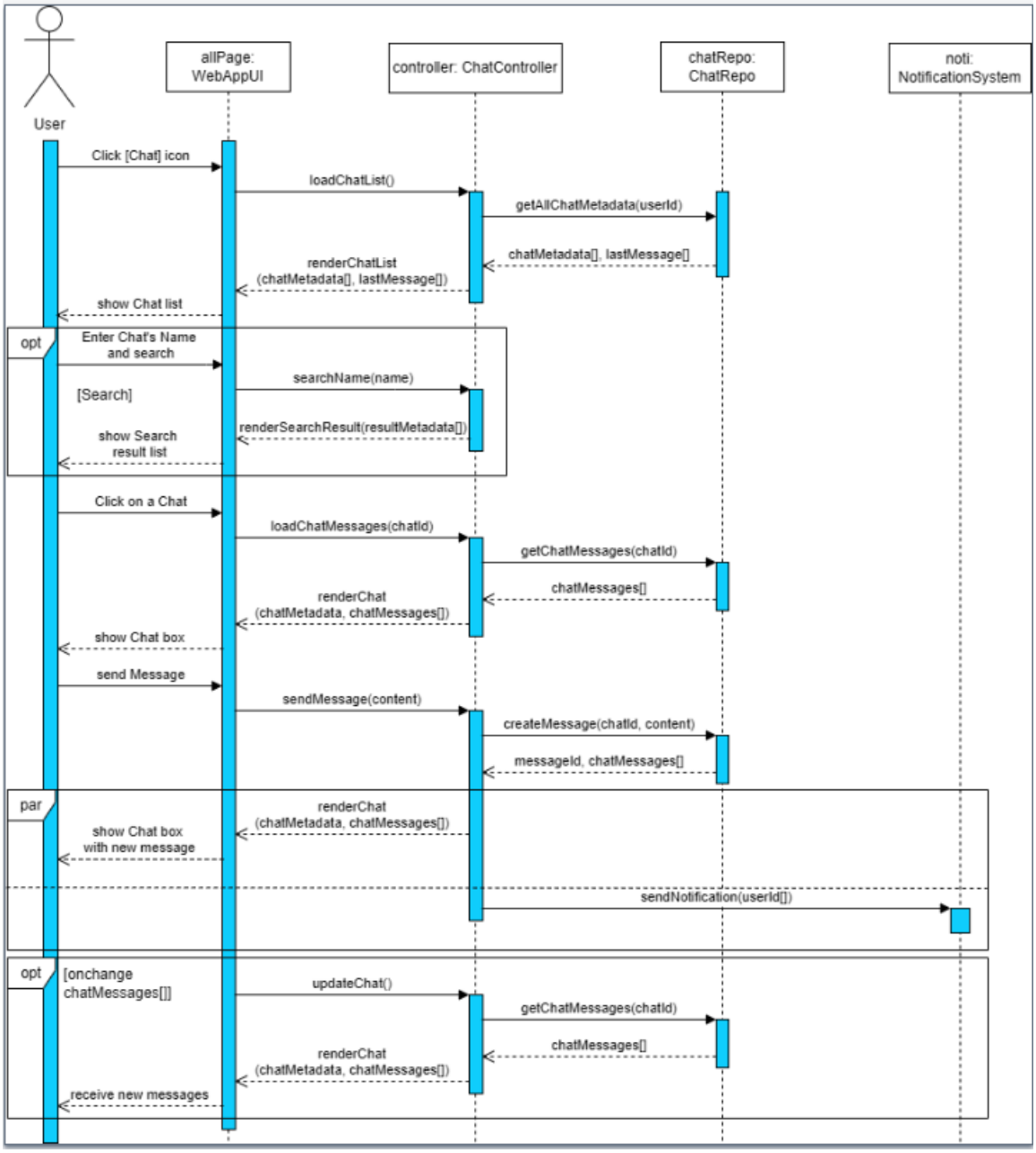


Figure 2.6: Sequence Diagram – Chat and Notification System Interaction

Actors and Lifelines

- **User:** Represents either a student, tutor, or coordinator who participates in one-on-one or group conversations.

- **allPage: WebAppUI** – Boundary component responsible for rendering chat interfaces, search results, and live message updates.
- **controller: ChatController** – Control component handling chat logic, including message retrieval, creation, and synchronization.
- **chatRepo: ChatRepo** – Entity repository that stores chat metadata, message histories, and user references.
- **noti: NotificationSystem** – External subsystem that dispatches message alerts to recipients when new messages arrive.

Main Flow

1. **Open Chat Interface:** When the user clicks the chat icon, the system executes `loadChatList()`, which triggers `getAllChatMetadata(userId)` in `ChatRepo`. The retrieved data (`chatMetadata[]` and `lastMessage[]`) are rendered via `renderChatList()`, displaying recent conversations.
2. **Search for Chat (opt):** The user can search for a specific contact by name through `searchName(name)`. The controller queries `ChatRepo` and displays matching chat metadata using `renderSearchResult(resultMetadata[])`.
3. **View Conversation:** Upon selecting a chat thread, the UI calls `loadChatMessages(chatId)`, retrieving data through `getChatMessages(chatId)`. The controller then renders the conversation via `renderChat(chatMetadata, chatMessages[])`.
4. **Send Message:** When a message is sent, the UI invokes `sendMessage(content)`. The controller processes the request through `createMessage(chatId, content)` and stores it in `ChatRepo`. After saving, the message is displayed locally with `renderChat()` and simultaneously triggers a notification using `sendNotification(userId)` from the `NotificationSystem`.
5. **Parallel Update (par):** While the message is sent, the system continuously updates the chat box in parallel through real-time rendering, ensuring all participants view the latest conversation state.
6. **Receive New Message (opt):** When new messages are received (detected by `onchange chatMessages[]`), the system updates the chat via `updateChat()`. The controller retrieves new data from `getChatMessages(chatId)` and refreshes the interface using `renderChat(chatMetadata, chatMessages[])`.

2.7 Sequence Diagram – Tutor Evaluation and Feedback

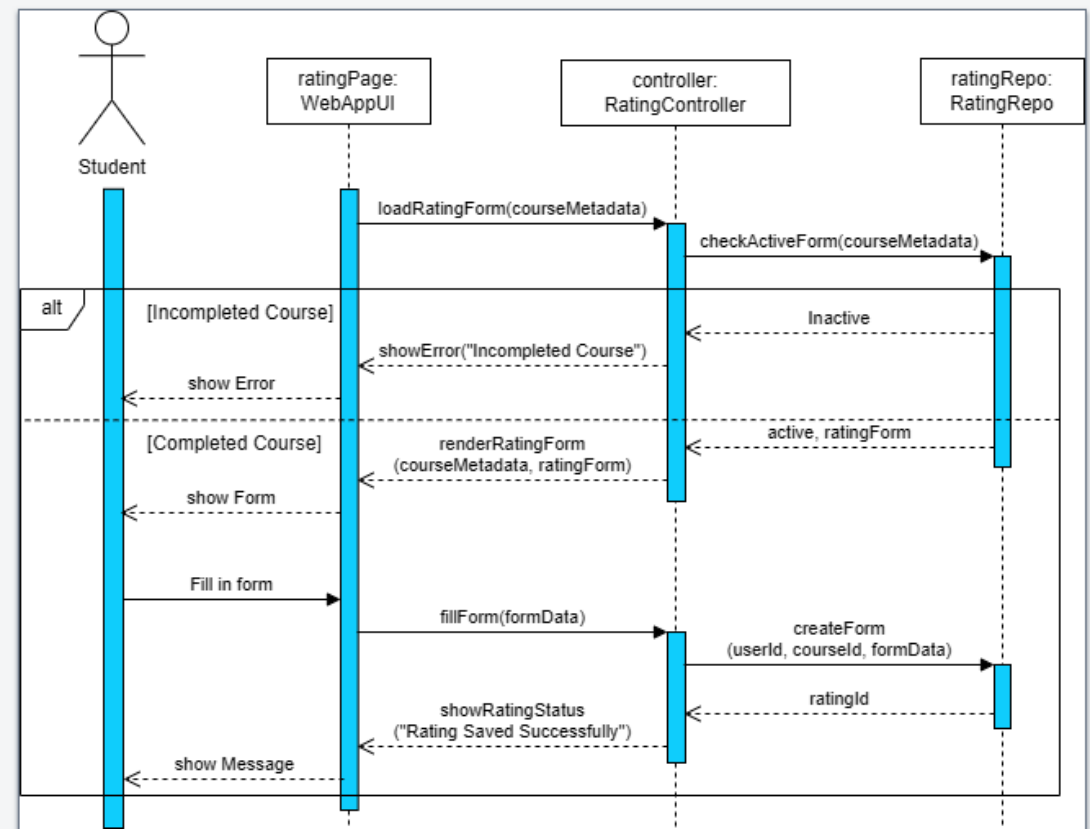


Figure 2.7: Sequence Diagram – Tutor Evaluation and Feedback Process

Actors and Lifelines

- **Student:** The actor responsible for evaluating the tutor and completing the feedback form upon course completion.
- **ratingPage: WebAppUI** – Boundary component that renders the feedback interface and displays system messages.
- **controller: RatingController** – Control component that coordinates validation, form activation, and data submission.
- **ratingRepo: RatingRepo** – Entity repository managing persistent storage of feedback records.

Main Flow

1. **Load Rating Form:** When the student navigates to the evaluation page, the interface triggers `loadRatingForm(courseMetadata)`, prompting the controller to verify the form's eligibility through `checkActiveForm(courseMetadata)` in the `RatingRepo`.

2. **Alternate Path – Incompleted Course (alt):** If the course is not yet completed, the repository returns an `Inactive` state. The controller displays an error message via `showError("Incompleted Course")` and prevents form access.

3. **Main Path – Completed Course (alt):** For completed courses, the repository responds with `active, ratingForm`. The controller instructs the UI to render the form through `renderRatingForm(courseMetadata, ratingForm)`.

4. **Submit Feedback:** The student fills in evaluation criteria (such as teaching quality, punctuality, communication, and overall satisfaction) and submits via fillForm(formData). The controller creates a record by calling `createForm(userId, courseId, formData)` in the `RatingRepo`, which returns a unique `ratingId`.

5. **Show Confirmation:** Once submission is saved, the system displays showRatingStatus("Rating Saved Successfully"), confirming successful completion.

2.8 Sequence Diagram – Searching and Importing Library References

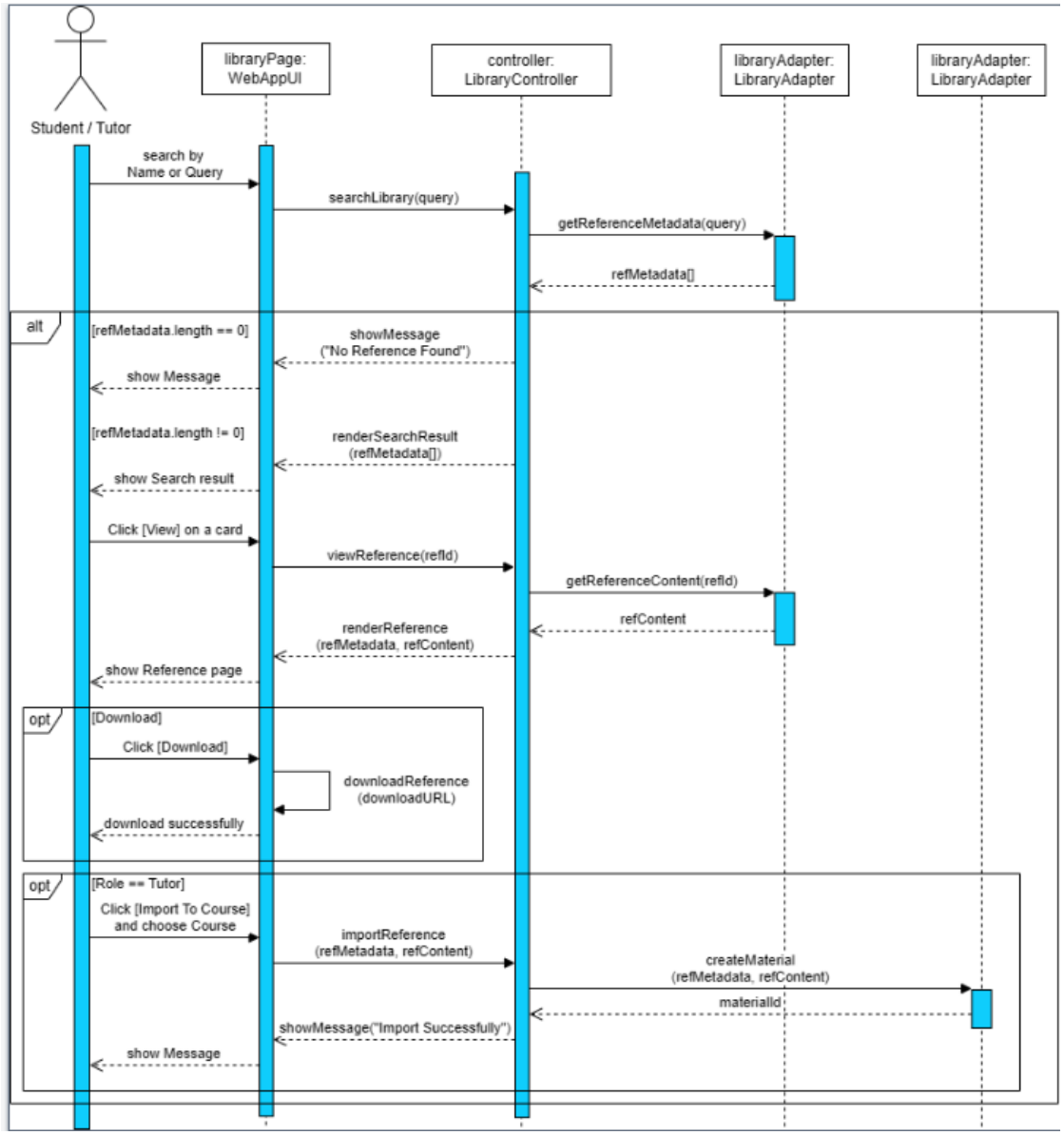


Figure 2.8: Sequence Diagram – Searching and Importing Library References

Actors and Lifelines

- **Student/Tutor:** The actor initiating the search for academic resources or importing materials into courses.
- **libraryPage: WebAppUI** – Boundary component presenting the search interface, query results, and download or import actions.

- **controller: LibraryController** – Control component managing search queries, view rendering, and coordination with the library adapter.
- **libraryAdapter: LibraryAdapter** – External integration layer connecting the TSS to HCMUT's digital library repository for metadata and content retrieval.

Main Flow

1. **Search Reference:** The student or tutor searches the library using a keyword or query. The UI invokes `searchLibrary(query)`, prompting the controller to request metadata from getReferenceMetadata(query) in the `LibraryAdapter`. The adapter returns a list of `refMetadata[]`.
2. **Alternate Path – No Result (alt):** If the returned list is empty, the UI displays `showMessage("No Reference Found")`.
3. **Main Path – References Found (alt):** When results are found, the controller instructs the UI to render them via `renderSearchResult(refMetadata[])`. The user can click `[View]` on a specific item to trigger `viewReference(refId)`.
4. **View Reference Details:** The controller retrieves detailed content through getReferenceContent(refId) and receives `refContent`, which the UI then displays using `renderReference(refMetadata, refContent)`.
5. **Optional – Download Material (opt):** Users may download the reference directly. The system executes `downloadReference(downloadURL)` and confirms with a success message once the file is retrieved.
6. **Optional – Import to Course (opt):** For users with a tutor role, the interface enables importing the reference into a course. The tutor selects a course and triggers `importReference(refMetadata, refContent)`. The controller calls createMaterial(refMetadata, refContent) in the `LibraryAdapter`, generating a new material record and returning `materialId`. A message `showMessage("Import Successfully")` confirms the action.

3 Activity Diagram

3.1 Student - Register Course

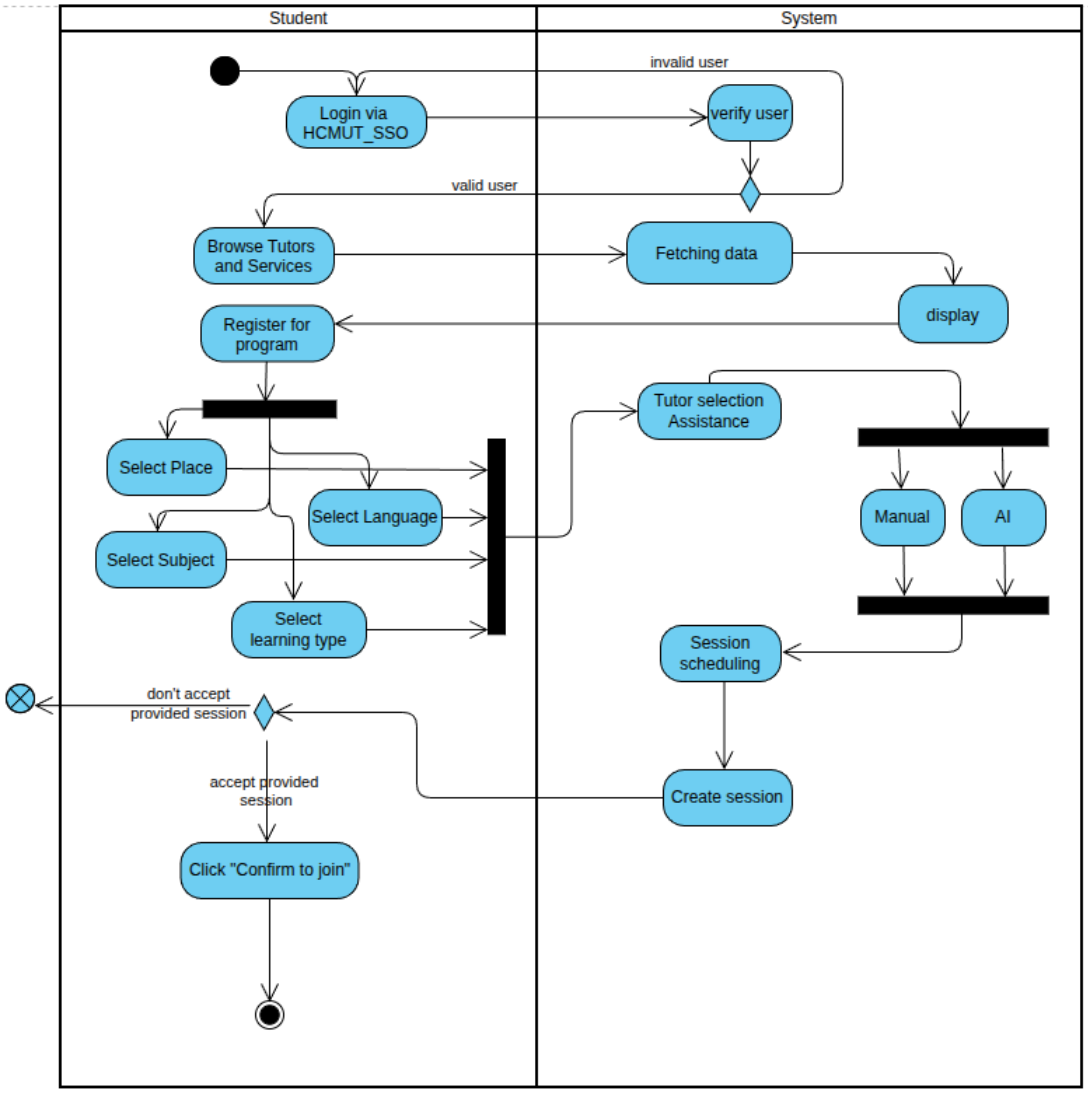


Figure 3.1: Student - Course Registration

- **Login:** Student logs into the system using HCMUT SSO to ensure authentication and access control.
- **Verify user:** The system verifies the user's credentials. If invalid, access is denied; if valid, the student proceeds.
- **Fetch data:** The system retrieves relevant tutoring and service information.

- **Browse tutors and services:** The student searches through available options.
- **Register for program:** The student enrolls in the desired tutoring or service program.
- **Select Place:** The student chooses the location for the session.
- **Select Language:** The student specifies the preferred language for tutoring.
- **Select Subject:** The student picks the desired subject for the tutoring session.
- **Select Learning Type:** The student selects the type of learning (e.g., individual, group).
- **Tutor Selection Assistance:** The system provides assistance to match the student with an available tutor.
  - **Manual:** The student chooses a tutor manually.
  - **AI:** The system recommends a tutor automatically using AI algorithms.
- **Session Scheduling:** The system schedules a session for the student based on availability.
- **Create Session:** The system creates the tutoring session for the student.
- **Provided Session Options:**
  - **Accept:** The student accepts the session and clicks "Confirm to join."
  - **Don't Accept:** The student declines the provided session.
- **Confirm to Join:** The student confirms and commits to joining the scheduled session.

3.2 Coordinator - Session matching

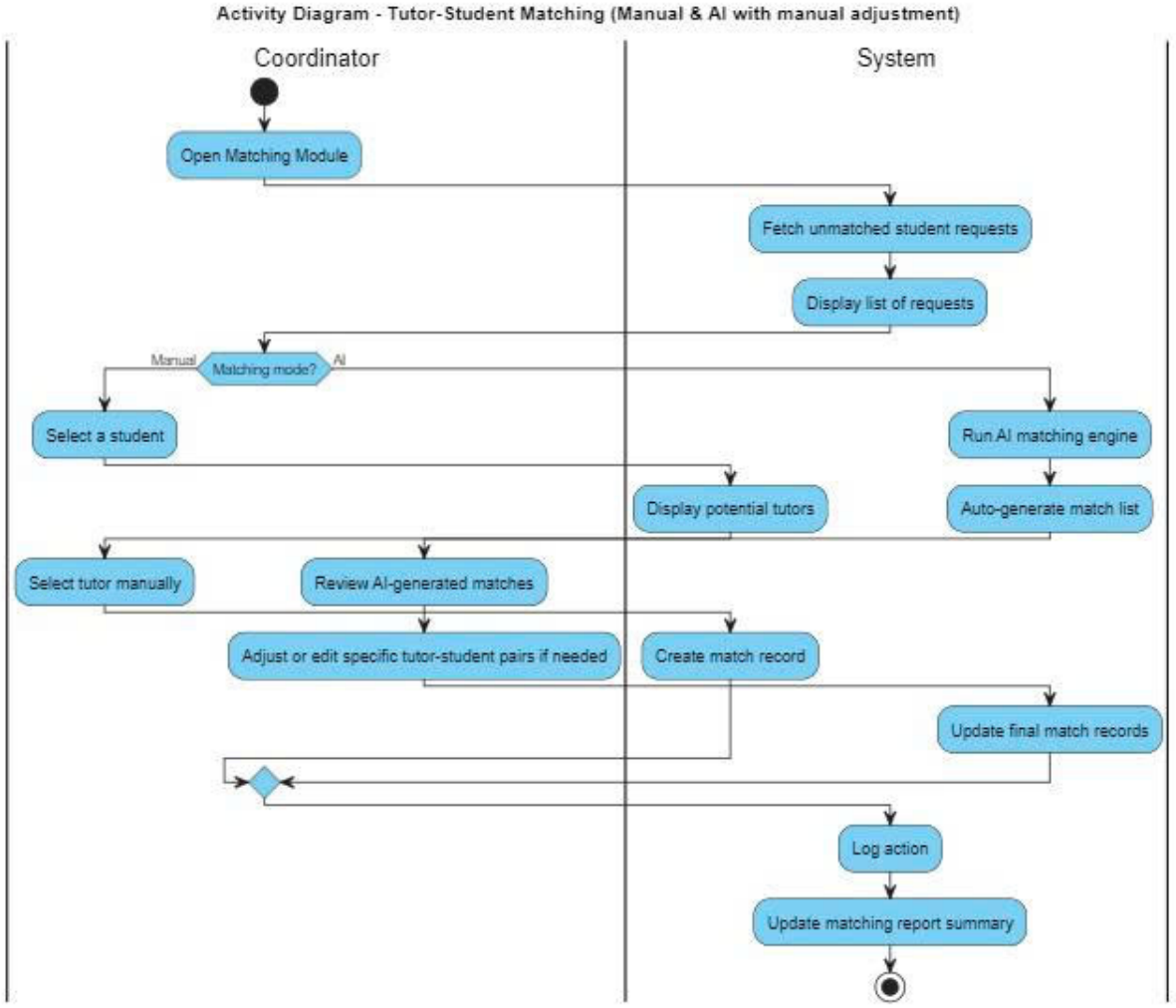


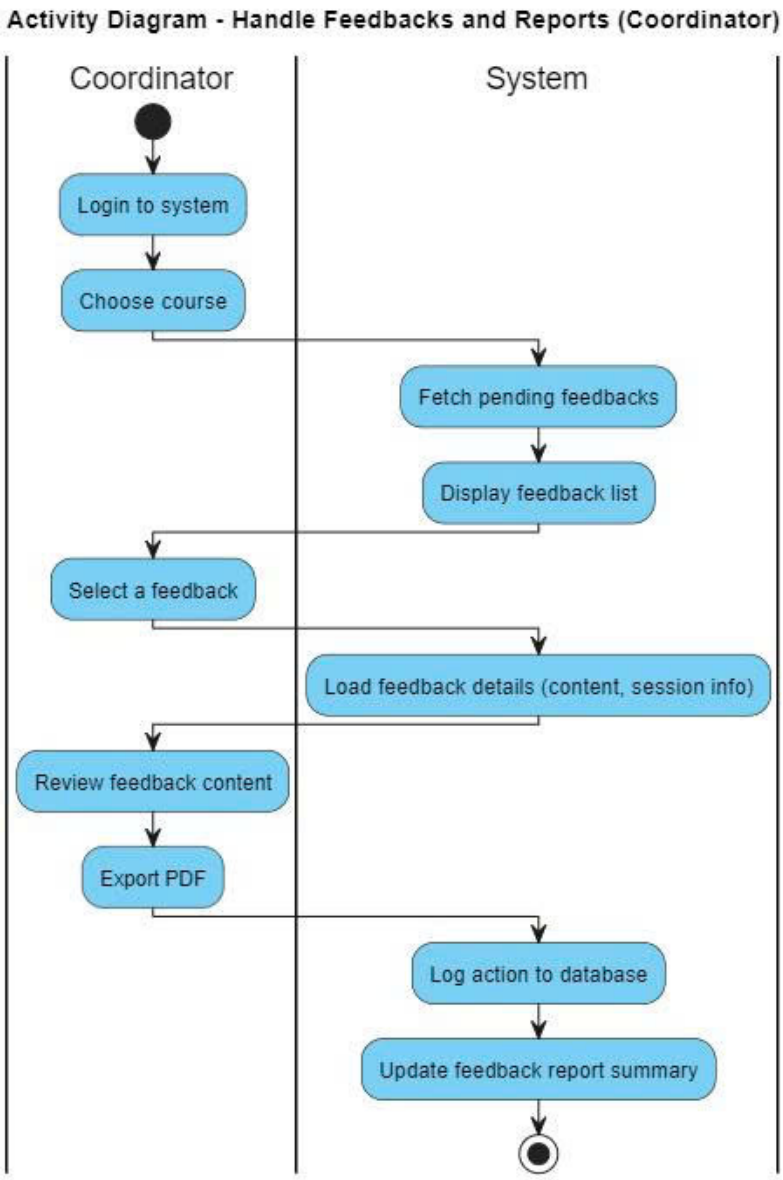
Figure 3.2: Coordinator - Session matching

- **Open Matching Module:** The Coordinator initiates the tutor-student matching by opening the matching module.
- **Fetch unmatched student requests:** The system retrieves all unmatched student requests.
- **Display list of requests:** The system lists the available student requests.
- **Matching mode:** The Coordinator selects whether the matching process will be manual or AI-driven.

  - **Manual** : The Coordinator selects a student and manually chooses a tutor.
    * **Select a student** : The Coordinator picks a student from the list of unmatched requests.
    * **Display potential tutors** : The system displays tutors who may be suitable for the selected student.
    * **Select tutor manually** : The Coordinator manually chooses a tutor for the student.
    * **Adjust or edit specific tutor-student pairs if needed** : The Coordinator can refine the match further, if required.
  - **AI:** The system runs the AI matching engine to generate matches.
    * **Run AI matching engine** : The system processes student requests and tutor availability using AI.
    * **Auto-generate match list**: The system generates a list of AI-recommended tutor-student pairs.
    * **Review AI-generated matches** : The Coordinator reviews the AI-generated match list.
    * **Adjust or edit specific tutor-student pairs if needed** : The Coordinator can refine the AI-generated matches, if necessary.

- **Create match record** : The system creates records for confirmed tutor-student pairs.
- **Update final match records** : The system stores the final updated matching records.
- **Log action** : The system logs the actions taken during the matching process for future reference and auditing.
- **Update matching report summary** : The system creates an updated summary of the matching report.

3.3 Coordinator - Handle feedback and report



- **Login to system:** The Coordinator logs into the system using their credentials.
- **Choose course:** The Coordinator selects the course for which feedback needs to be reviewed.
- **Fetch pending feedback** : The system retrieves all pending feedback related to the chosen course.
- **Display feedback list** : The system displays the list of pending feedback for the Coordinator to review.

- **Select a feedback** : The Coordinator selects a specific feedback from the displayed list.
- **Load feedback details:** The system loads the details of the selected feedback, including the content and session information.
- **Review feedback content** : The Coordinator reviews the content of the feedback to analyze the student's comments.
- **Export PDF** : After reviewing, the Coordinator can export the feedback details and reports in PDF format.
- **Log action to database** : The system saves the Coordinator's action (review, export, etc.) to the database for auditing and records.
- **Update feedback report summary** : The system updates the feedback report summary to reflect the reviewed feedback and associated actions.

3.4 Tutor - Register for program

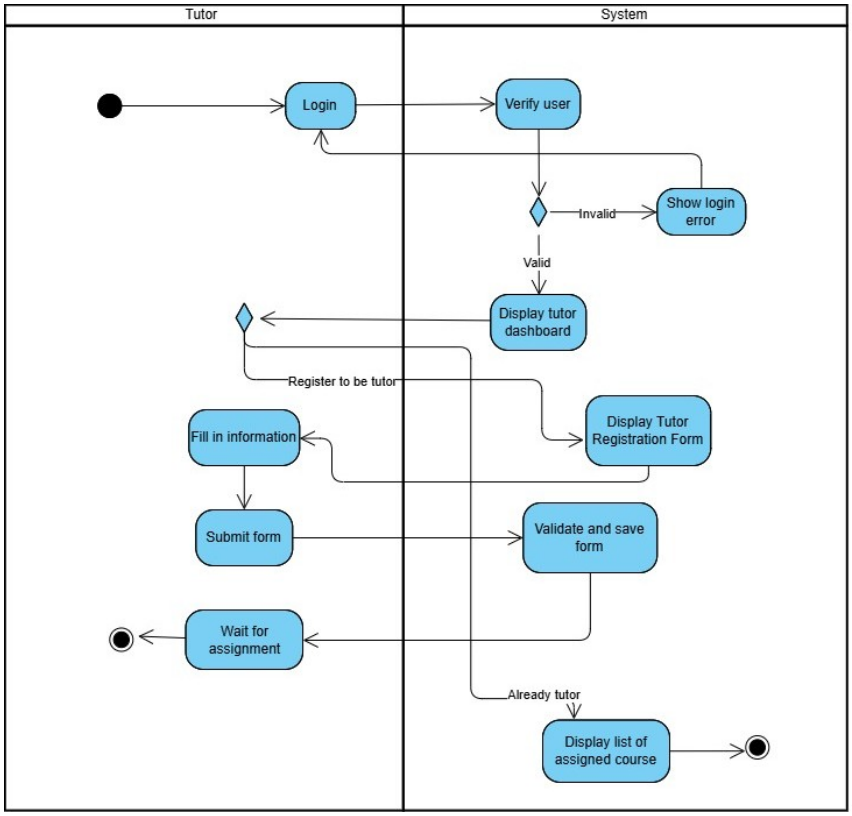


Figure 3.3: Tutor's register activity diagram

- **Login and dashboard access**

  1. **Login**: The Tutor initiates the process by performing the login action.
  2. **Verify User**: The System receives the login credentials and proceeds to verify the user.
  3. **Login Decision**: The System checks if the login is valid. If invalid, the System will show a login error. The flow then loops back to the Tutor's login action, prompting them to try again. If valid, the System proceeds to display the Tutor dashboard.

- **New Tutor Registration**

  1. If the user selects an option to register to be a tutor, the System responds by displaying the tutor registration form

2. **Fill Information**: The Tutor then proceeds to fill in information on the registration form.
3. **Submit Form**: Once the form is complete, the Tutor submits the form.
4. **Validate and Save**: The System receives the submission and validates the information before saving the form.
5. **Wait for Assignment**: After the form is successfully saved, the Tutor enters a state of waiting for assignment. The Tutor needs to wait for a coordinator to approve their registration and assign them courses.

3.5 Tutor - grading and rating

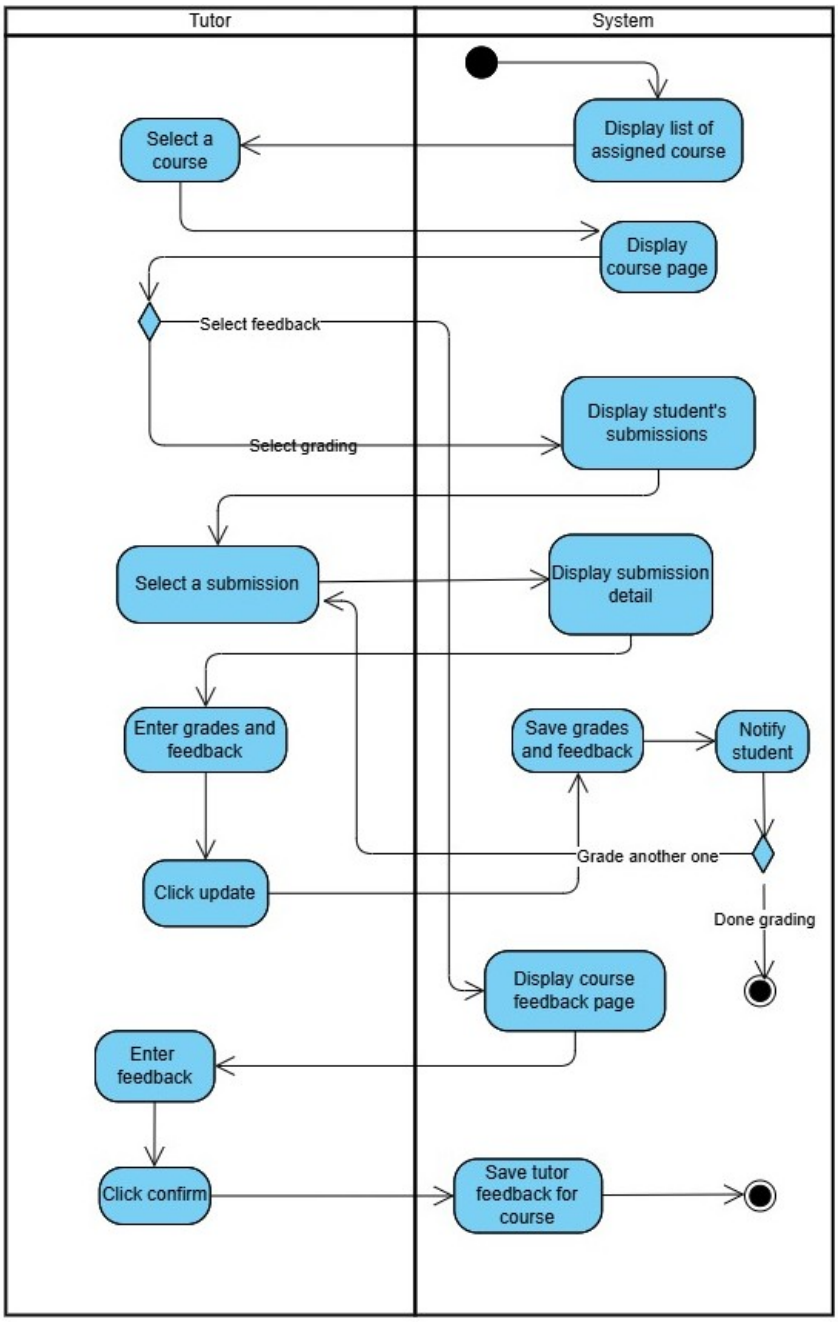


Figure 3.4: Tutor's grading and rating activity diagram

- **Grading**

1. If the user is already a tutor, the system will display list of assigned course

2. **Select Course**: The tutor then chooses a specific course by performing the select course action.
3. **Display Course Page**: The System will display the course page for the selected course.
4. **Select Grading**: From the course page, if the Tutor selects grading, the System proceeds to the grading page.
5. **Display Submissions**: The System displays a list of the student's submissions for that course.
6. **Select a Submission**: The Tutor then selects a submission from the list to grade.
7. **Display Submission Detail**: The System shows the details of the selected submission.
8. **Enter Grades**: The Tutor reviews the work and proceeds to enter grades and feedback.
9. **Update**: The Tutor finalizes the grading by clicking update.
10. **Save and Notify**: The System receives the update command, then saves the grades, feedback, and notifies the student.
11. If the tutor choose to grade another student, the loop comeback to select submission. If the tutor has done grading, the activity path concludes at the final node

- **Feedback**

1. If the tutor choose feedback, the System presents a course feedback page
2. **Enter Rating**: The Tutor interacts with this page to enter a rating and provide comments about the course.
3. **Confirm**: The Tutor finalizes their feedback by clicking confirm.
4. **Save Feedback**: The System then takes this input and saves the Tutor's feedback for the course.

3.6 Tutor - Create session and summary

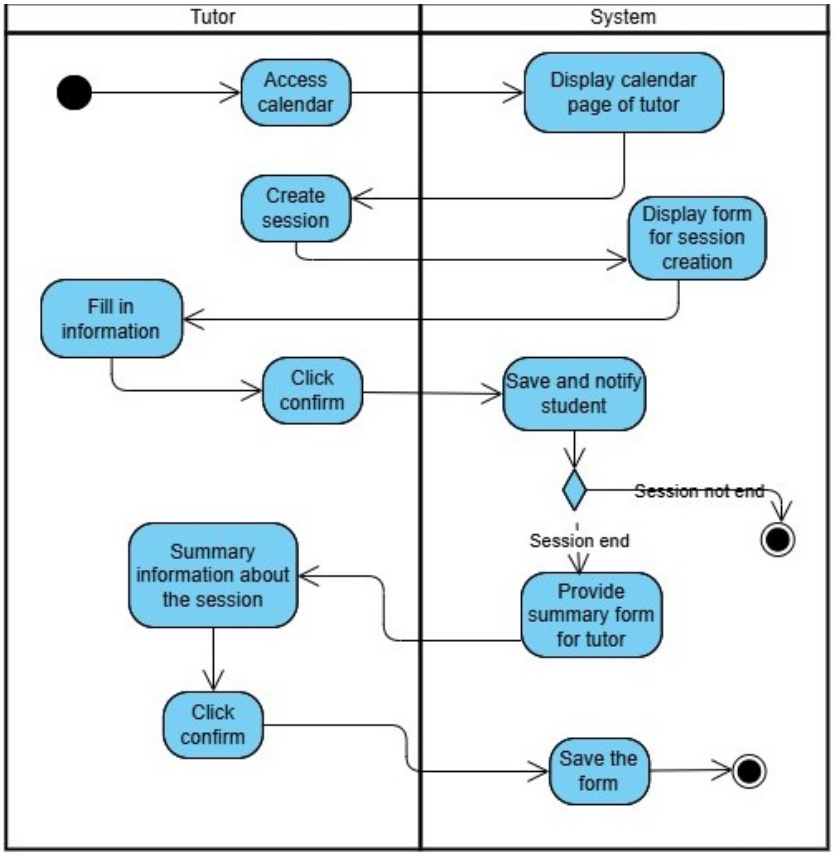


Figure 3.5: Tutor's create session and summary activity diagram

- **Access calendar** : The tutor initiates the process by choosing to access their calendar
- **Display Calendar** : The Tutor then proceeds to take the attendance of students
- **Create Session** : From the calendar view, the tutor decides to create a new session
- **Display form session creation** : The system display the form for the tutor to fill in information for the session
- **Fill in Information** : The tutor is prompted to fill in the necessary details for the new session (e.g., date, time, topic)
- **Click Confirm** : After entering the information, the tutor confirms the details
- **Record Session Note** : The Tutor uses this form to record note for the session

- **Save and Notify Student** : The system receives the confirmed information, saves the new session to the database, and sends a notification to the student
- **Provide summary form** : If the session has ended, the system provides a summary form for the tutor to fill out
- **Fill in Summary Information** : The tutor fills in the summary details about the completed session (e.g Attendance, student's progress, next steps)
- **Click Confirm** : The tutor confirms the summary information they have entered
- **Save the Form** : The system saves the completed summary form

3.7 Department Chair - Create report

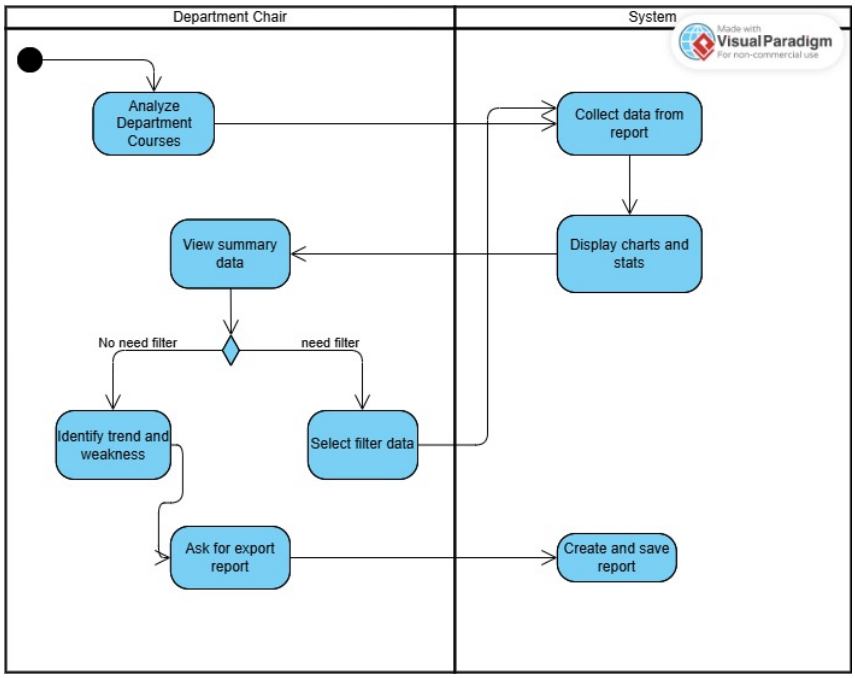


Figure 3.6: Department chair's create report activity diagram

- The Department Chair initiates the process by selecting an option to Analyze Department Courses.
- **Data Collection**: The request is sent to the System, which then performs the action to collect data from reports.

- **Data Visualization**: After collecting the data, the System processes it and then proceeds to display charts and stats.
- **View and Decide**: The flow returns to the Department Chair, who can now view summary data. If the Chair decides they need a filter to narrow down the data, they select filter data. After applying the filter, the flow loops back to the System, which updates the data and displays the filtered charts and stats again. If the Chair determines there is no need for a filter, they proceed down this path.
- **Identify Trends and Weakness**: The Department Chair performs the task to identify trends and weaknesses based on the data they are viewing.
- **Export Report**: After completing the analysis, the Department Chair asks the System to export a report.
- **Report Generation**: The System receives this request, then creates and saves the report.

3.8 Program admin

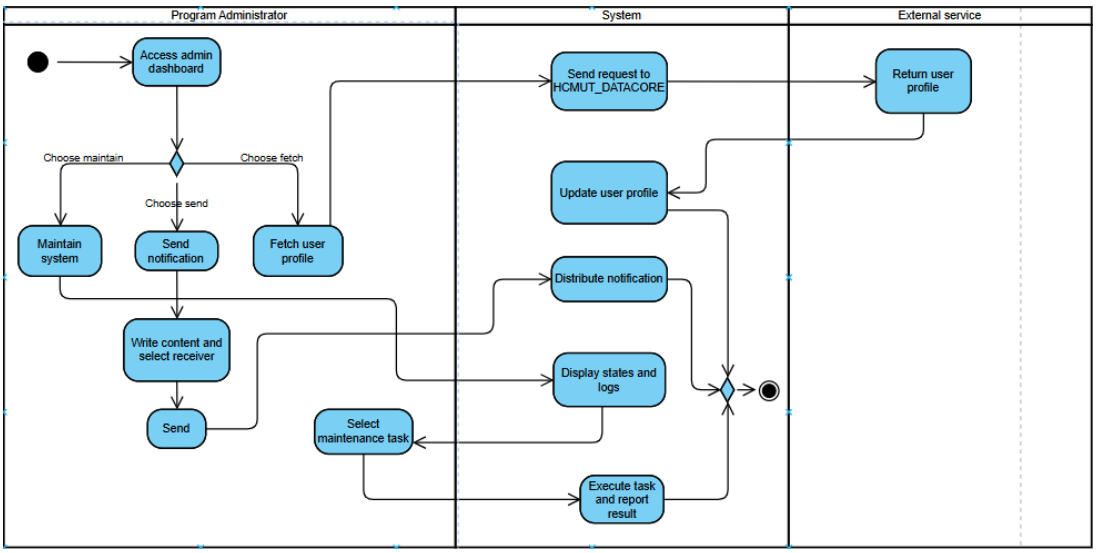


Figure 3.7: Program admin's activity diagram

- The administrator's first action is to access the admin dashboard. This is the central hub from which all other activities are initiated.

- After accessing the dashboard, the admin can choose between "maintain", "send notification" or "fetch user profile"
- If the admin choose to fetch user profile
    1. **Request Data**: This triggers the System to send a request to external services, which is the external service holding the user data.
    2. **Return Data**: The External service processes the request and performs the action to return the user profile.
    3. **Update Profile**: The System receives the profile data and uses it to update the user profile.
- If the admin choose to send notification
    1. **Compose Message**: The administrator then proceeds to write the content and select the receiver.
    2. **Send Action**: After composing the message, the administrator clicks Send.
    3. **Distribute Notification**: The System receives the command and distributes the notification to the selected receivers.
- If the admin choose maintenance
    1. **Display State, Logs**: The system displays the state and logs at the current time.
    2. **Select Task**: The administrator is then prompted to select a maintenance task from a list of available options.
    3. **Execute Task**: The System receives the selection and proceeds to execute the selected task and report the result.

4 State-chart

4.1 Statechart Diagram – Assignment Lifecycle

Purpose The **Assignment Lifecycle Statechart Diagram** models the various states an assignment passes through within the Tutor Support System (TSS). It captures both tutor and student interactions from assignment creation to review, including exceptional conditions such as late submissions, upload errors, and grading appeals.

Description The diagram defines how the system dynamically transitions between assignment states in response to user actions and system triggers. Each state represents a meaningful stage in the lifecycle of an assignment — ensuring clear traceability of submission and evaluation events across the platform.

States and Transitions

- **Created:** The tutor creates a new assignment. The system initializes assignment metadata such as title, description, deadline, and associated course.
- **Assigned:** The system assigns the created assignment to the respective students and dispatches notifications. Students can view assignment details and begin working toward submission. Two possible transitions emerge:
  - *Late:* If the deadline passes before submission, the assignment transitions to the `Overdue` state.
  - *Submit before deadline:* If the student successfully submits work, the system transitions to `Submitted`.
- **Submitted:** The student has submitted the assignment before or after the deadline. In the case of an upload error, the system permits re-uploading before marking the submission as final. Tutors then review and evaluate the submission, transitioning it to `Graded`.
- **Graded:** The tutor completes the review and assigns a score. If a student fails to submit, the system automatically assigns a grade of zero and terminates the process. A student may file an appeal, transitioning the assignment into `UnderReview`.
- **UnderReview:** The coordinator or tutor revisits the evaluation after an appeal, potentially updating the score. Once the review is finalized, the assignment lifecycle concludes.
- **Overdue:** The system marks assignments that missed the deadline. Students can still submit late, but the submission is flagged and may result in partial credit or administrative review.

Design Notes

- The diagram follows the **finite state machine (FSM)** paradigm, ensuring that every assignment has a clearly defined state at all times.

- Circular transitions such as `uploadError → Submitted` allow recovery from common user errors without data loss.
- Late submissions and appeals introduce realistic academic governance rules handled automatically by the system.
- The lifecycle closes only after all grading and review actions are completed, maintaining system consistency and audit traceability.

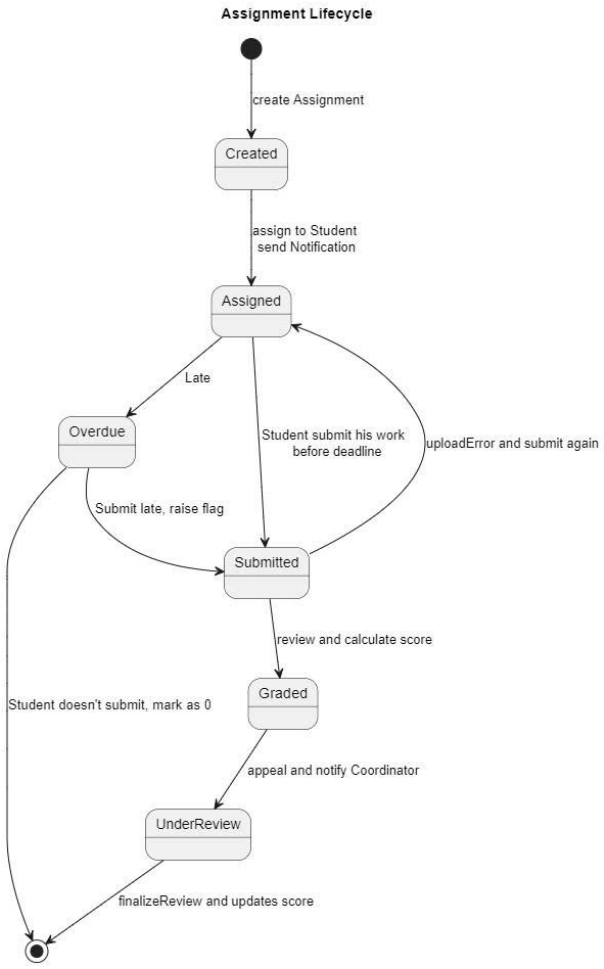


Figure 4.1: Statechart Diagram – Assignment Lifecycle in TSS

Summary This statechart encapsulates the behavioral flow of an assignment entity within the TSS ecosystem, bridging tutor evaluation workflows and student submission activities. By enforcing explicit transitions between states, the system maintains robust status tracking, late-handling mechanisms, and transparent feedback loops to ensure academic integrity and operational reliability.

4.2 Statechart Diagram – Session Lifecycle

Purpose The **Session Lifecycle Statechart Diagram** illustrates the dynamic behavior of a tutoring session within the Tutor Support System (TSS), capturing how each session transitions between states from scheduling to completion or cancellation. This model ensures traceability of all tutor–student interactions, reflecting real-world scheduling, confirmation, and attendance patterns.

Description Each session in TSS is instantiated after the matching process between a tutor and a student. The diagram tracks subsequent transitions based on confirmations, cancellations, and the passage of scheduled time. It provides a clear behavioral definition for managing active sessions and maintaining historical consistency in reports and analytics.

States and Transitions

- **Draft:** The initial state of a session created automatically after tutor–student matching. The session details (time, location, format) are editable by the tutor until confirmation.
- **Confirmed:** When the tutor confirms the schedule, the session transitions to the `Confirmed` state. Both the tutor and the student receive notifications. At this stage:
  - The tutor or student may still cancel before the session starts (*Cancel before start*).
  - If the tutor cancels prior to confirmation, the session reverts or is removed (*Cancel before confirmation*).
- **Ongoing:** When the scheduled start time is reached, the session transitions automatically to the `Ongoing` state. It represents an active tutoring session currently in progress.
- **Cancelled:** Either the tutor or the student may cancel at any point before the session starts or during its progress. Once cancellation is confirmed, the system marks the session as `Cancelled` and updates notifications accordingly.
- **Completed:** Upon successful conclusion of a session, the system transitions it to `Completed`. Attendance records are finalized, and both tutor and student may proceed to evaluation and rating activities.

Design Notes

- The lifecycle enforces distinct paths for **confirmation, execution, and cancellation**, preventing ambiguous session states.
- Transitions are governed by role-based triggers: tutors control schedule confirmation; both parties may initiate cancellation requests.
- Once a session reaches `Completed`, it becomes immutable to preserve attendance integrity for analytics and performance evaluation.
- The model aligns with asynchronous workflows between tutors and students, reflecting realistic timing and communication constraints.

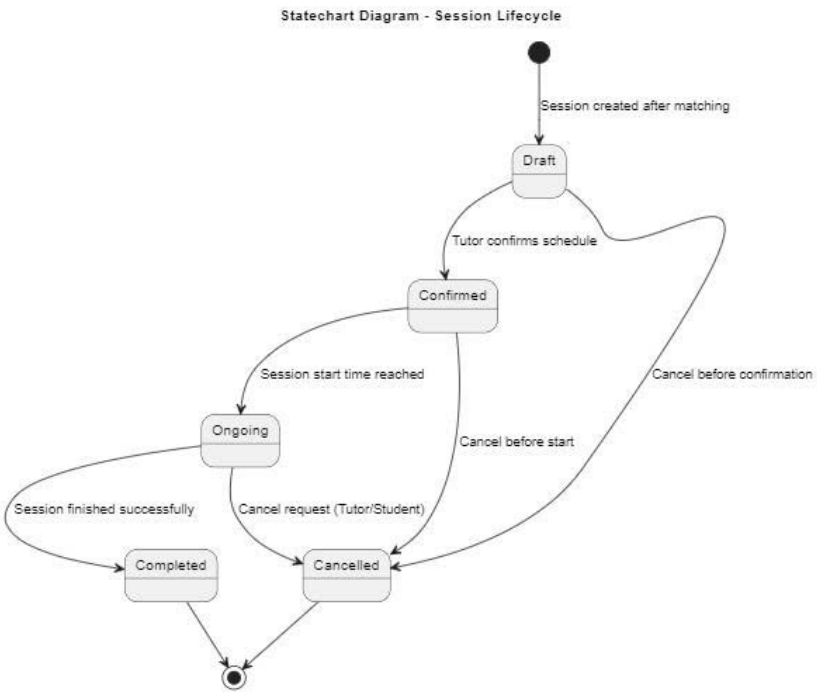


Figure 4.2: Statechart Diagram – Session Lifecycle in TSS

Summary This statechart defines the full behavioral lifecycle of tutoring sessions in the TSS environment. By formalizing transitions from creation to completion, it ensures consistency between scheduling logic, calendar synchronization, and user notification flows. The model guarantees transparent, accountable session management and supports downstream operations such as attendance tracking and feedback collection.

5 Materials

- UI mockup link: Figma Design Mockup
- Activity diagram link: Activity Diagram folder
- State chart diagram link: State chart diagram folder
- Sequence diagram link: Sequence diagram folder