Chapter Two

# Refined Sequence Diagrams

## 2.2 Components of a High-Level Sequence Diagram

A high-level sequence diagram shows how actors interact with the system through a sequence of messages. It supports system analysis by visualizing workflows for specific use cases.

### Key Components:

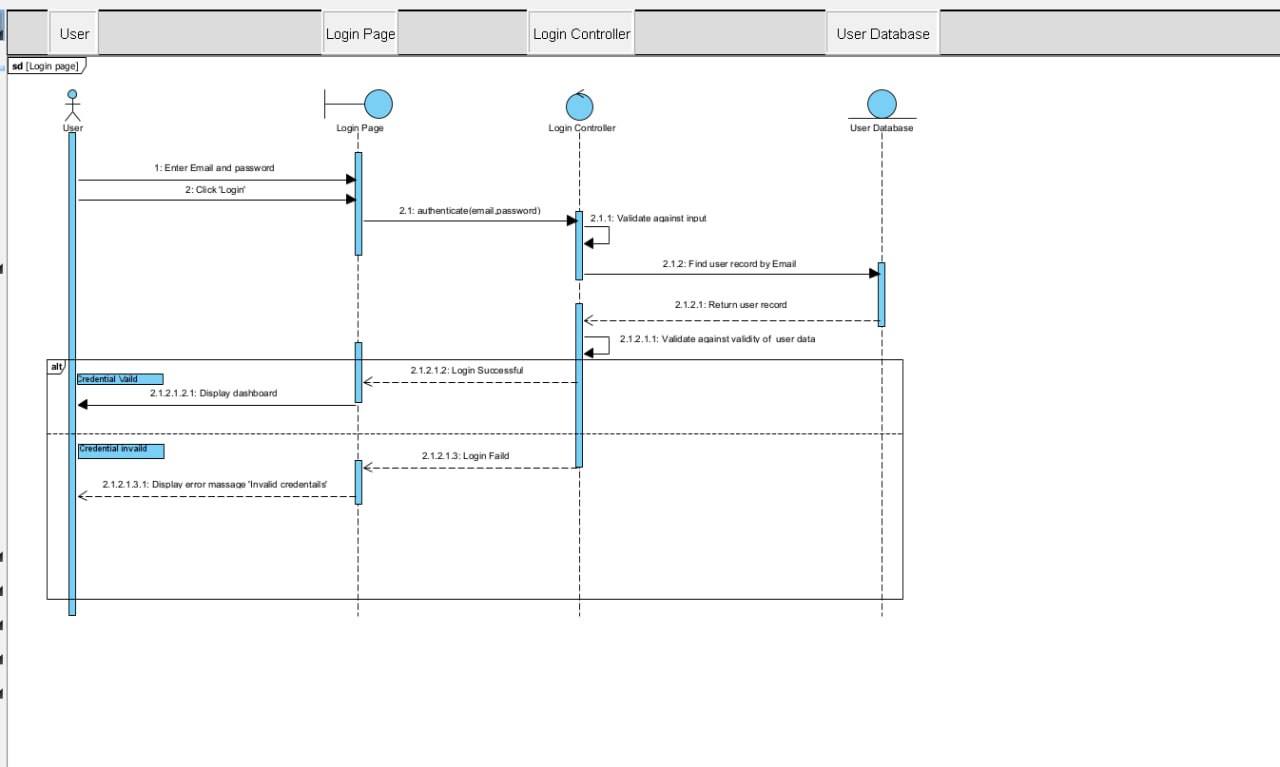
* Actors: External users or systems (e.g., User, Admin, Instructor). Represented by stick figures.
* Objects/Participants: Internal system elements (e.g., Login Form, Auth Controller, UserDB). Represented by underlined rectangles.
* Lifelines: Dashed vertical lines under each participant showing their active time in the scenario.
* Messages: Horizontal arrows showing interaction.  
  \* Solid arrows: synchronous messages.  
  \* Open arrows: asynchronous messages.  
  \* Return messages: dotted arrows with open heads.
* Activation Bars: Thin vertical rectangles over lifelines to indicate execution.
* Combined Fragments (alt, loop): Show conditionals and repetition.
* Boundary-Control-Entity Pattern: \* Boundary: UI layers (e.g., LoginForm).  
  \* Control: Logic handlers (e.g., AuthController).  
  \* Entity: Data stores (e.g., UserDB).

## 2.3 Refined Examples from Use Cases

### UC-1: Login

Actors: User

System Components: LoginForm (UI), AuthController, UserDB



1. User → LoginForm: Enter credentials

2. User → LoginForm: Click "Login"

3. LoginForm → AuthController: authenticate(username, password)

4. AuthController → UserDB: fetch user by username

5. UserDB → AuthController: return user record

6. AuthController: verify password hash

alt Credentials valid:

AuthController → LoginForm: login successful

LoginForm → User: show dashboard

else Credentials invalid:

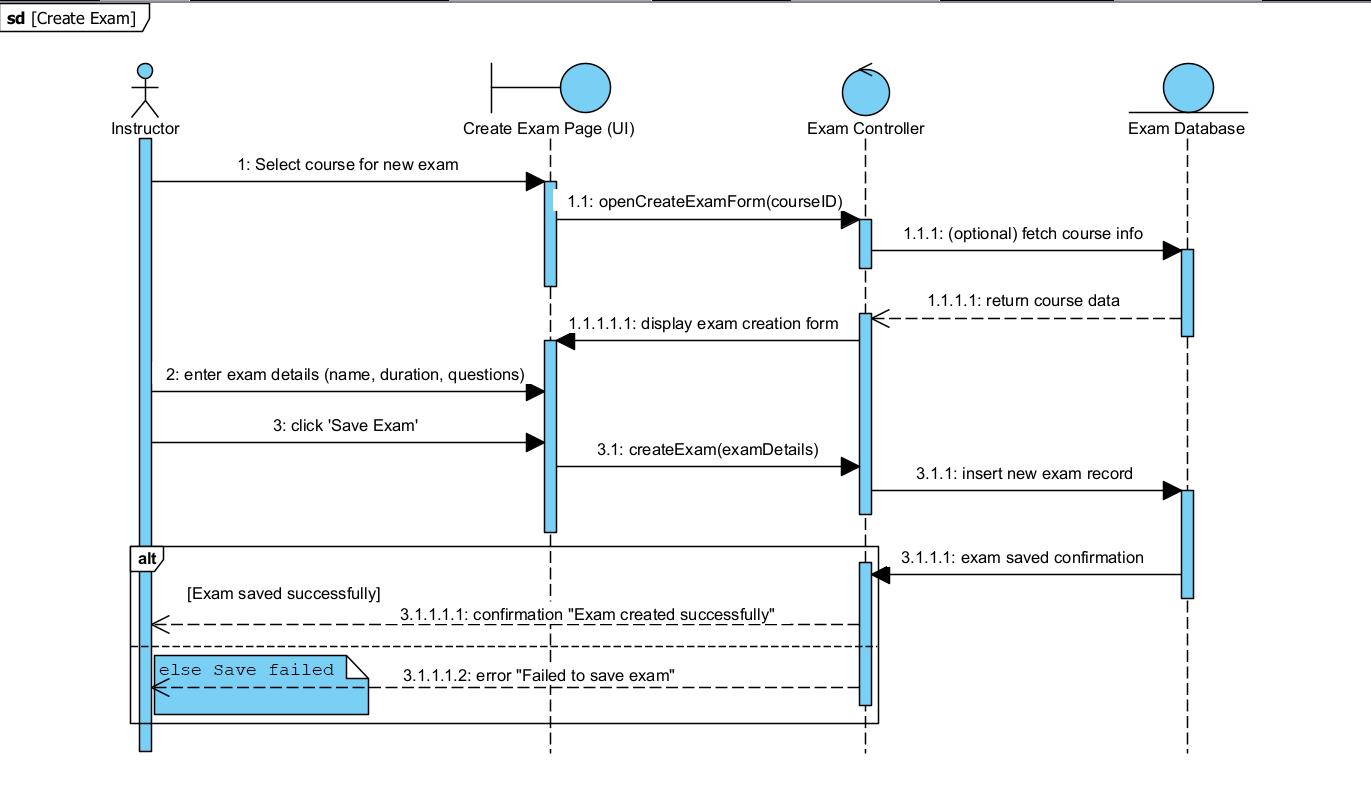
AuthController → LoginForm: login failed

LoginForm → User: display error

### UC-2: Create Exam

Actors: Instructor

System Components: CreateExamPage (UI), ExamController, ExamDB



1. Instructor → CreateExamPage: Select course

2. CreateExamPage → ExamController: openCreateExamForm(courseID)

3. ExamController → ExamDB: (optional) fetch course info

4. ExamDB → ExamController: return course data

5. ExamController → CreateExamPage: display creation form

6. Instructor → CreateExamPage: input exam details

7. CreateExamPage → ExamController: createExam(details)

8. ExamController → ExamDB: insert exam record

9. ExamDB → ExamController: save confirmation

alt Save Successful:

ExamController → CreateExamPage: "Exam created successfully"

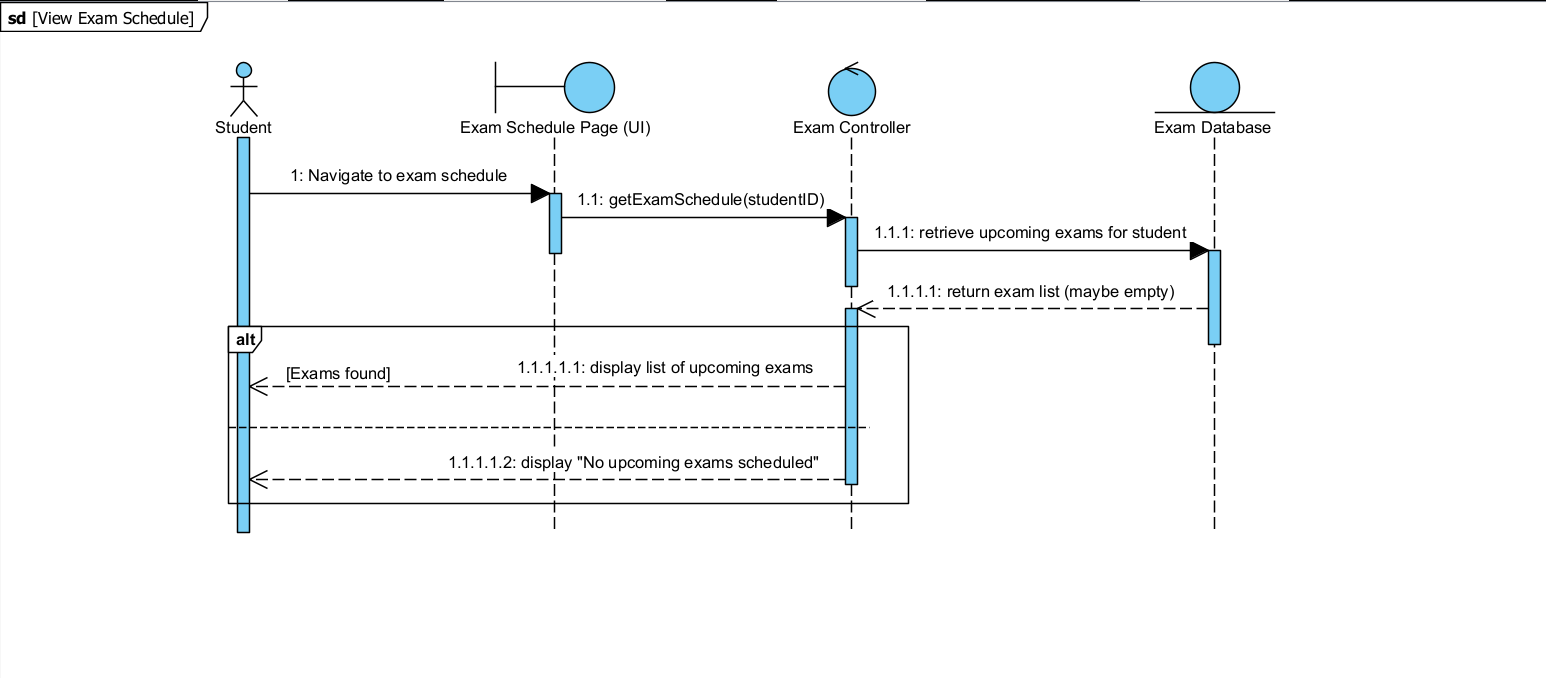
else Save Failed:

ExamController → CreateExamPage: "Failed to save exam"

### UC-3: View Exam Schedule

Actors: Student

System Components: ExamSchedulePage (UI), ExamController, ExamDB



1. Student → ExamSchedulePage: navigate to schedule

2. ExamSchedulePage → ExamController: getExamSchedule(studentID)

3. ExamController → ExamDB: retrieve exams

4. ExamDB → ExamController: return exam list (possibly empty)

alt Exams Found:

ExamController → ExamSchedulePage: display upcoming exams

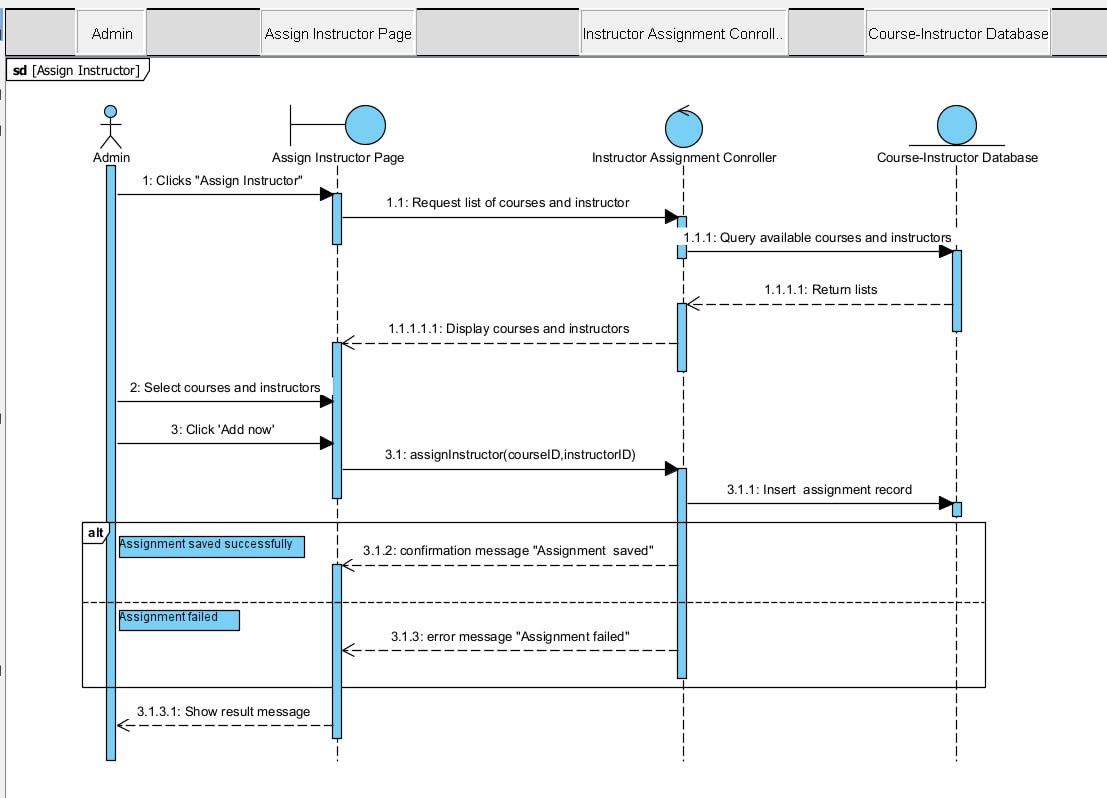
else No Exams:

ExamController → ExamSchedulePage: display "No upcoming exams scheduled"

### UC-5: Assign Instructors

Actors: Admin

System Components:AssignInstructorPage (UI), AssignmentController, CourseInstructorDB



1. Admin → AssignInstructorPage: Select "Assign Instructor"

2. AssignInstructorPage → AssignmentController: request lists

3. AssignmentController → CourseInstructorDB: query data

4. CourseInstructorDB → AssignmentController: return lists

5. AssignmentController → AssignInstructorPage: show options

6. Admin → AssignInstructorPage: select course & instructor

7. AssignInstructorPage → AssignmentController: submit assignment

8. AssignmentController → CourseInstructorDB: save assignment

alt Success:

AssignmentController → Admin: confirmation message

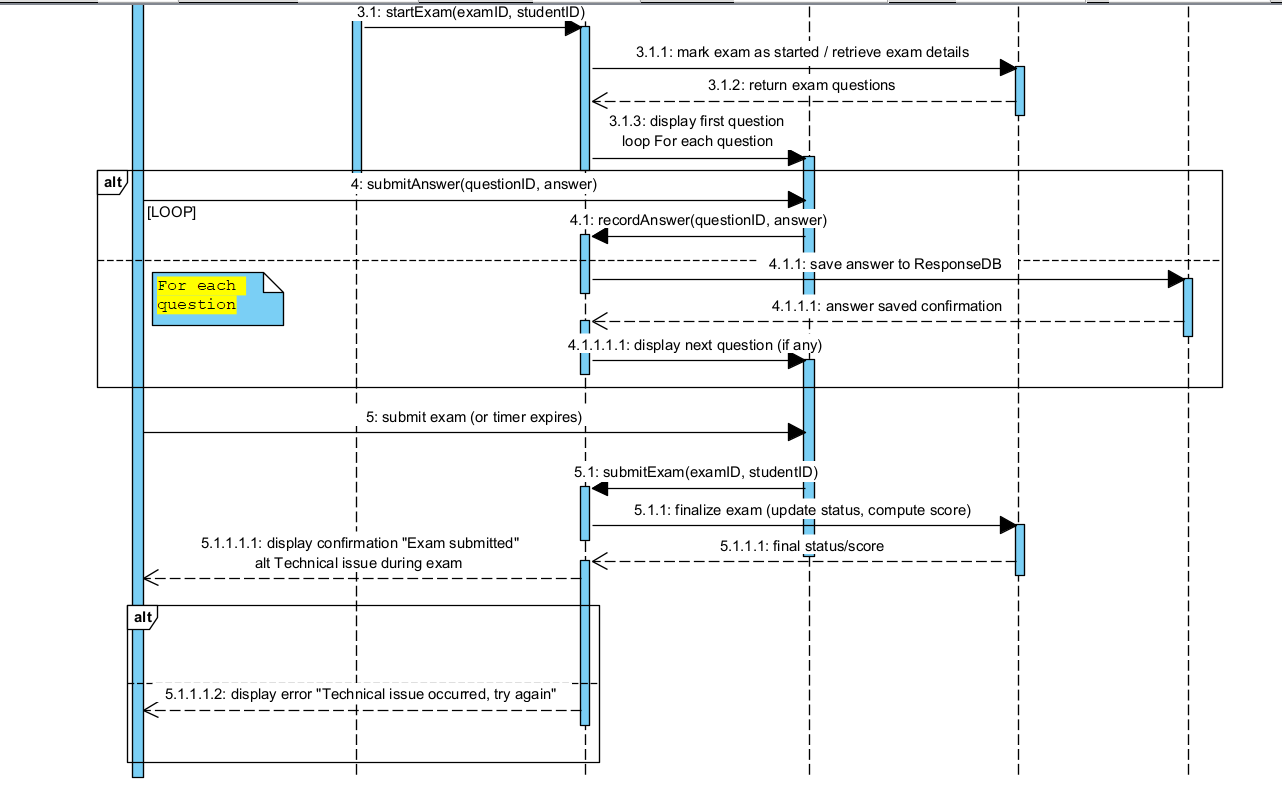
else Failure:

AssignmentController → Admin: show error

### UC-22: Take Exam

Actors: Student

System Components: ExamListPage (UI), ExamController, QuestionPage (UI), ExamDB, ResponseDB



1. Student → ExamListPage: navigate to available exams

2. ExamListPage → ExamController: getAvailableExams(studentID)

3. ExamController → ExamDB: query exams

4. ExamDB → ExamController: return list

5. ExamController → ExamListPage: display options

6. Student → ExamListPage: select and start exam

7. ExamListPage → ExamController: startExam(examID, studentID)

8. ExamController → ExamDB: retrieve questions

9. ExamDB → ExamController: return questions

10. ExamController → QuestionPage: display first question

[loop for each question]

Student → QuestionPage: submitAnswer(questionID, answer)

QuestionPage → ExamController: recordAnswer

ExamController → ResponseDB: save answer

ResponseDB → ExamController: confirmation

ExamController → QuestionPage: display next question

11. Student → QuestionPage: submit exam (or timeout)

12. QuestionPage → ExamController: submitExam(examID, studentID)

13. ExamController → ExamDB: finalize exam (status, score)

14. ExamDB → ExamController: return result

alt Success:

ExamController → QuestionPage: "Exam submitted"

else Technical Issue:

ExamController → QuestionPage: "Try again later"

**2.4 Tools and Steps to Draw High Level Sequence Diagram**

* Tool Used: Visual Paradigm

1. Open Visual Paradigm and create or open your project.
2. Go to Diagram Navigator → Right-click Diagrams → New → UML → Sequence Diagram.
3. Name the diagram (e.g., 'Login Flow', 'Create Exam').
4. Drag 'Actor' from the toolbar to represent external users (e.g., Student, Instructor).
5. Drag 'Lifeline' for each system component (e.g., UI Page, Controller, DB).
6. Use solid arrows for synchronous messages and dashed arrows for return messages.
7. Insert 'Activation Bars' on lifelines to show active operations.
8. Use 'Combined Fragments' (alt/loop) to represent conditions and loops.
9. Align and format the layout for clarity.
10. Export the diagram as PNG/JPG to embed in documentation.