# NegevNerds

## Collaborative Exam Preparation Platform

## Application Design Document


Ben-Gurion University

Software Engineering Department

**Project Advisor:**

Dr. Elior Sulem

**Team Members:**

Shachar Cohen

Nadav Ktalav

David Volodarsky

Noa Aboody

# Contents

# 1. Use Cases

Use-Cases Diagram:

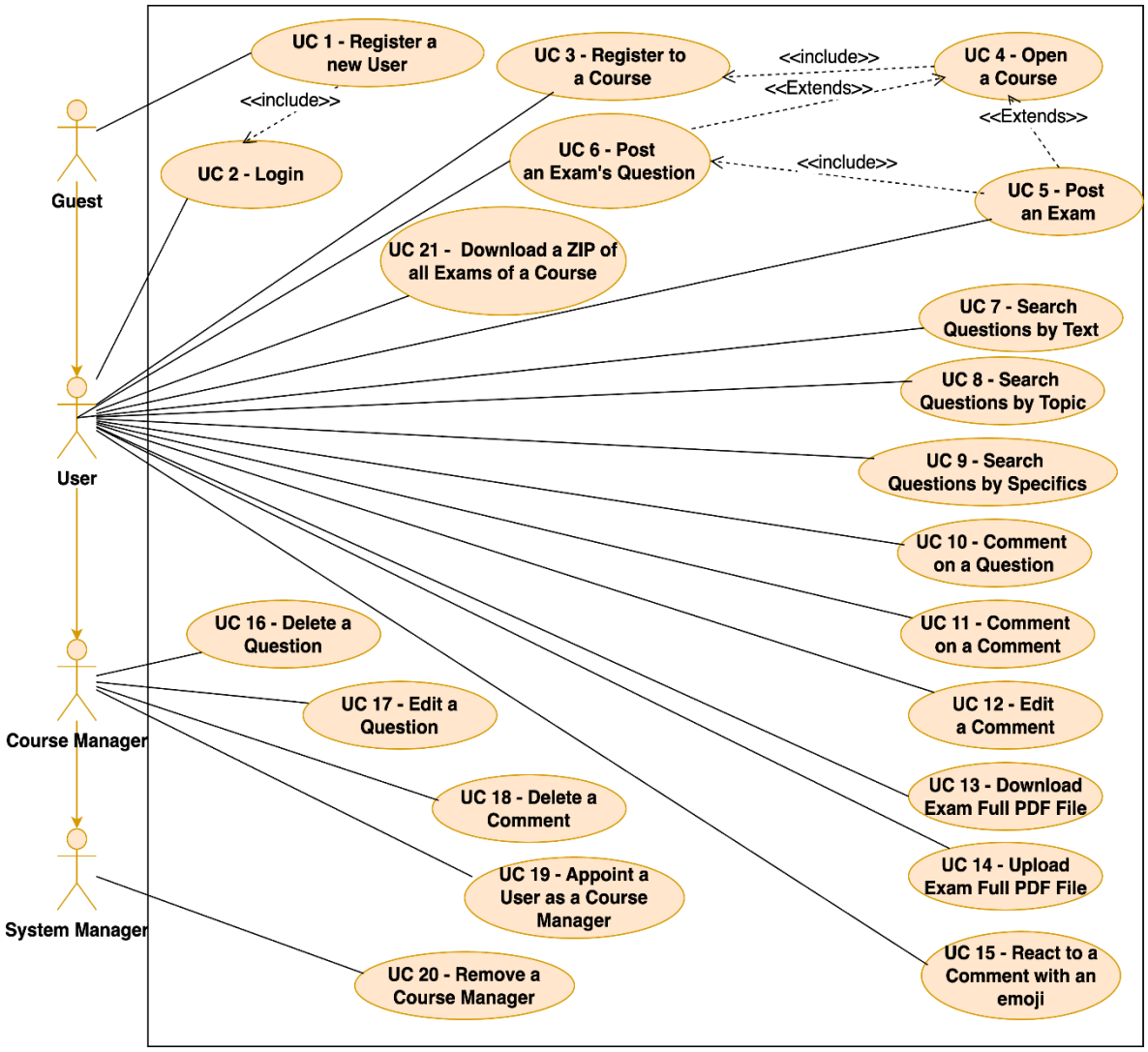



*Figure 1: Use Cases Diagram*

### 1.1 UC 1 – Register a new user

Description: A guest is trying to access the platform. This is the guest first time in the platform therefore he will have to register before getting an access to the website.

Actor: Guest

Parameters: name, last name, mail address, password, password confirm.

Pre-conditions:

1. The guest has no access to the website content since he is not registered yet.

Post-Conditions:

1. The guest details are being saved in the system database.
2. The user receives a confirmation email with registration details.
3. The guest, which is now a user, is being transported to the homepage of the website.

Main success scenario:

1. The guest enters the “NegevNerds” website.
2. The guest will fill the following details – name, last name, mail (BGU mail only) password and password confirm.
3. The guest will have to accept the platform terms of use.
4. The system validates the information provided.
5. A verification code will be sent to the guest email to verify is identity.
6. The guest will type the verification code that he got in his mail.
7. The system checks the provided code:
8. For a matching code, the registration will end successfully with the guest details are stored in the system database and the guest has become a user.

Alternative flows:

1. The guest is trying to register with a mail address that already in the system database - the system returns a relevant error message and registration fails.
2. The guest is trying to register with a non-BGU mail address, the system returns a relevant error message and registration fails.
3. There is no matching between the password and password confirm, the system returns a relevant error message and registration fails.
4. The guest typed an unmatching verification code, the system returns a relevant error message and registration fails.

### 1.2 UC 2 – Login

Description: A user wants to log in to the platform to access its features and content.

Actor: User / Course Manager / System Manager

Parameters: email, password

Pre-conditions:

1. The use is not logged in.

Post-conditions:

1. The user is authenticated and redirected to the homepage.
2. User's login session is active.

Main success scenario:

1. The user navigates to the "NegevNerds" login page.
2. The user enters their email and password.
3. The system validates the credentials:
    a. If valid, the system grants access and redirects the user to the homepage.
    b. The user's session begins.

Alternative flows:

1. **Invalid Credentials:** If the entered email or password is incorrect, the system displays an error message: *"Invalid email or password. Please try again."*
2. **Forgot Password:** The user clicks "Forgot Password?" and is redirected to the password recovery flow.

### 1.3 UC 3 – Register to a Course

Description: A user wants to register for a course on the platform to access its content and participate in discussions.

Actor: User / Course Manager / System Manager

Parameters: courseID, userID

Pre-conditions:

1. The user is registered and logged into the platform.
2. The course exists in the system.

Post-Conditions:

1. The user's details are added to the course's participant list.
2. The course appears now in the user homepage.

Main success scenario:

1. The user search for a specific course by its name or id.
2. The user clicks on the register button of the course.
3. The system confirms the registration by displaying a success message.
4. The course can be seen on the user homepage now.

Alternative flows:

1. **The user is already registered for the course:** The system displays an error message stating the user is already enrolled.
2. **The course is not available on the site**: The system displays an error message stating that there is no course such course in the platform and suggest the user the option to open it.

### 1.4 UC 4 – Open a Course

Description: A user wants to create and open a new course on the platform.

Actor: User / Course Manager / System Manager

Parameters: user id, course id, course name, course syllabus file

Pre-conditions:

1. The user is logged into the system.

Post-Conditions:

1. The new course is added to the system and is visible to students.
2. The user who opened the course is automatically become course-manager for this course.
3. The user is automatically being registered to the course.

Main success scenario:

1. The user clicks on the “Open course” bottom in his homepage.
2. The user provides the course details.
3. The system verifies the provided details.
4. The course is successfully created and becomes visible to students.
5. The user is automatically appointed as the course manager and being registered to the course.

Alternative flows**:**

1. **The course has already existed:** The system displays an error message stating the course is already in the platform.
2. **There is no such course in Ben Gurion University**: The system displays an error message stating that there is no such course being studied in Ben Gurion University.

### 1.5 UC 5 – Post an Exam

Description**:** A user wants to upload an exam to website.

Actor**:** User / Course Manager / System Manager

Parameters: exam file

Pre-conditions:

1. The user is registered and logged into the system.

Post-Conditions**:**

1. The exam is stored in the system database.

Main success scenario:

1. The user clicks on the “Upload exam” bottom in his homepage.
2. The user uploads the exam file.
3. The system automatically extracts the exam details – course name, year, semester (A/B/C) and moed (A/B/C).
4. The exam is being stored in the system database according to the details that haven extracted.
5. The exam is being listed and now can be seen in the course page on the website.

Alternative flows:

1. **The uploaded file format is invalid:** The system displays an error message and rejects the upload.
2. **The course which the exam belongs to, is not on the website**: The system displays an error message and suggest the option to open the course on the platform.
3. **The exam is already on the website:** The system displays an error message.

4. **The exam's details have not been extracted successfully**: The system displays an error message and send a notification to the course manager.

### 1.6 UC 6 – Post an Exam's Question

Description: A user posts a single question from a specific exam.

Actor: User / Course Manager / System Manager

Parameters: course id, year, semester, moed, question's number, is American, the question file – pdf or photo, the question file – pdf or photo (optional).

Pre-conditions:

1. The user is registered and logged into the platform.

Post-Conditions:

1. The question is visible to other users on the platform.

Main success scenario:

1. The user clicks on the "post question" bottom in the homepage/relevant course or exam page.
2. The user provides all the needed details of the question.
3. The user confirms and clicks "post".
4. The system confirms the question has been successfully posted.

Alternative flows:

1. **The course doesn't exist on the website:** The system displays an error message and suggest the option to open the course on the platform.
2. **The question already exists in the system:** A relevant error message will be displayed.

### 1.7 UC 7 – Search Questions by Text

Description: A user wants to find all the questions on the website that contains questions with text the user types in.

Actor: User / Course Manager / System Manager

Parameters: Text

Pre-conditions:

1. The user is logged into the platform.

Post-Conditions:

1. A list of questions matching the text is displayed.

Main success scenario:

1. The user goes to the search bar at the homepage.
2. The user chooses the option of searching by text.
3. The user enters text related to the question he is looking for.
4. The system processes the text and displays a list of matching questions.

Alternative flows:

1. **No questions match the keywords:** The system displays a message stating no results were found.

### 1.8 UC 8 – Search Questions by Topic

Description: A user wants to find all the questions on the website that contains questions with the topics the user types in.

Actor: User / Course Manager / System Manager

Parameters: Topics

Pre-conditions:

1. The user is logged into the platform.

Post-Conditions:

1. A list of question matching the topics is displayed.

Main success scenario:

1. The user goes to the search bar at the homepage.
2. The user chooses the option of searching by topic.
3. The user enters topics related to the exam they are looking for.
4. The system processes the keywords and displays a list of matching exams/ questions.

Alternative flows:

1. **No exams/questions match the keywords:** The system displays a message stating no results were found.

### 1.9 UC 9 – Search a Question by Specifics

Description: A user searches for a specific question using specific details.

Actor: User / Course Manager / System Manager

Parameter: course name, year(optional), semester (optional), moed(optional), question number(optional).

Pre-conditions:

1. The user is logged into the platform.

Post-Conditions:

1. The system navigates to the exam homepage.

Main success scenario:

1. The user goes to the search bar at the homepage.
2. The user chooses the option of searching by specific details.
3. The user applies the specific details.
4. The system displays the specific question homepage.

Alternative flows:

1. **No question matches the filters:** The system displays a message stating no results were found.

### 1.10 UC 10 – Comment on a Question

Description**:** A user wants to ask about a specific question or share his thought and opinion about it.

Actor: User / Course Manager / System Manager

Parameters: optional – photo or pdf file

Pre-conditions:

1. The user is logged into the platform.
2. The question exists in the system.

Post-Conditions:

1. The comment is visible under the question.

Main success scenario:

1. The user navigates to the question homepage.
2. The user presses “Add Comment” option.
3. The user writes / uploads file and submits the comment.
4. A notification is sent to all the user who have already commented on this question.
5. The system confirms the comment is added successfully.

Alternative flows:

1. **The comment is empty:** The system displays an error message prompting the user to add content to the comment.
2. **Violation of the platform policy**: The comment doesn't stand with the platform policy therefore will not be posted.

### 1.11 UC 11 – Comment on a comment

Description**:** A user wants to comment on another user’s comment in a discussion about a question.

Actor: User / Course Manager / System Manager

Parameters: optional – photo or pdf file

Pre-conditions:

1. The user is logged into the platform.
2. The question exists in the system.
3. The comment which the user comments on is exists.

Post-Conditions:

1. The comment is visible under the question.

Main success scenario:

1. The user navigates to the question homepage.
2. The user presses “Add Comment” option.
3. The user writes / uploads file and submits the comment.
4. A notification is sent to the user who wrote the comment the user commented on.
5. The system confirms the comment is added successfully.

Alternative flows:

1. **The comment is empty:** The system displays an error message prompting the user to add content to the comment.
2. **Violation of the platform policy**: The comment doesn't stand with the platform policy therefore will not be posted.

### 1.12 UC 12 – Edit a Comment

Description**:** A user wants to edit their previously added comment to correct or update its content.

Actor: User / Course Manager / System Manager

Parameters: optional – photo or pdf file

Pre-conditions:

1. The user is logged into the platform.
2. The question exists in the system.
3. The comment which the user wants to edit exists and belongs to the user.

Post-Conditions:

1. The comment is updated and visible under the question.

Main success scenario:

1. The user navigates to the question homepage.
2. The user locates their existing comment under the question.
3. The user presses the "Edit Comment" option associated with their comment.
4. The user edits the content (text, photo, or PDF) and submits the changes.
5. The system validates the changes (e.g., checks file format or comment length).
6. The system confirms that the comment was successfully updated.

Alternative flows:

1. **The comment is empty:** The system displays an error message prompting the user to add content to the comment.
2. **Violation of the platform policy**: The comment doesn't stand with the platform policy therefore will not be posted.

### 1.13 UC 13 – Download Exam Full PDF File

Description**:** A user wants to download the entire exam PDF file to their device for offline access or review.

Actor: User / Course Manager / System Manager

Parameters: None

Pre-conditions:

1. The user is logged into the platform.
2. The exam file exists in the system and is accessible.

Post-Conditions:

1. The exam PDF is successfully downloaded to the user's device.

Main success scenario:

1. The user navigates to the exam homepage.
2. The user clicks the “Download Exam” button.
3. The system retrieves the PDF file of the exam.
4. The exam is downloaded and saved on the user’s device.

Alternative flows:

1. **File Not Found:** The PDF file for the exam is missing or corrupted in the system.

### 1.14 UC 14 – Upload Exam Full PDF File

Description**:** A user wants to upload a full exam PDF file to the system for students to access.

Actor: User / Course Manager / System Manager

Parameters: exam file pdf

Pre-conditions:

1. The user is logged into the platform.
2. The exam file does not exist in the system.

Post-Conditions:

1. The exam file is successfully uploaded and associated with the specified exam entry.
2. The file available for users to download.

Main success scenario:

1. The user navigates to the exam homepage.

2. The user clicks the "Upload Exam" button and selects the pdf file from their device.
3. The user clicks the "Upload" button.
4. The system processes and uploads the file to the server
5. The system associates the file with the specified course or exam entry.
6. The file available for users to download.

Alternative flows:

1. **Unsupported File Format:** The user selects a file that is not in PDF format. The system displays relevant error message.

### 1.15 UC 15 – React to a Comment with an emoji

Description**:** A user wants to react to an existing comment by selecting an emoji (e.g., 👍, ❤️, 😂) to express their feedback or opinion

Actor: User / Course Manager / System Manager

Parameters: The emoji selected for the reaction.

Pre-conditions:

1. The user is logged into the platform.
2. The comment exists in the system.

Post-Conditions:

1. The selected emoji is displayed as a reaction under the comment.

Main success scenario:

1. The user navigates to the question page where the comment is located
2. The user finds the specific comment they want to react to.
3. The user clicks the "React" button or icon next to the comment.
4. The system displays a list of available emojis.

5. The user selects an emoji from the list.
6. The reaction is visible to all users.
7. A notification is sent to the comment's writer.

Alternative flows:

1. **Reaction Changed:** The user selects a different emoji after previously reacting to the comment.

### 1.16 UC 16 – Delete a Question

Description**:** A Course Manager wants to delete a question from the system to ensure the platform contains only right/ relevant or appropriate content.

Actor: Course Manager / System Manager

Parameters: None

Pre-conditions:

1. The Course Manager is logged into the platform.
2. The question exists in the system.

Post-Conditions:

1. The question is removed from the system.
2. All associated comments and reactions to the question are also removed.

Main success scenario:

1. The Course Manager navigates to the question page.
2. The Course Manager locates the "Delete Question" button on the page and clocks on it.
3. The system displays a confirmation dialog to confirm the deletion.
4. The Course Manager confirms the deletion


5. The system removes all associated comments and reactions
6. The system confirms that the question was successfully deleted.

Alternative flows:

1. **Course Manager Cancels Deletion:** The Course Manager chooses "Cancel" in the confirmation dialog.

### 1.17 UC 17 – Edit a Question

Description**:** A Course Manager wants to edit the details of an existing question, such as the year, semester, moed, question number, topics, question path, or answer path, to ensure the information is accurate and up-to-date.

Actor: Course Manager / System Manager

Parameters: year(optional), semester(optional), moed(optional), question number(optional), topics(optional), question path(optional), answer path(optional)

Pre-conditions:

1. The Course Manager is logged into the platform.
2. The question exists in the system.

Post-Conditions:

1. The question attributes are updated.
2. The changes are visible to all users who can access the question.

Main success scenario:

1. The Course Manager navigates to the question page.
2. The Course Manager locates the " Edit Question" button.
3. The system displays an editable form with the current details of the question.

4. The Course Manager updates one or more of the following fields:
    I. Year
    II. Semester
    III. Moed
    IV. Topics
    V. Question Number
    VI. Question Path
    VII. Answer Path
5. The Course Manager clicks the "Save Changes" button.
6. The system validates the changes.
7. The system updates the question details in the database.
8. The system confirms that the changes were successfully saved.
9. The updated question can be seen on the website.

Alternative flows:

1. **Duplicate:** The Course Manager updated the question's attributes and now they are matching another question's attributes.

### 1.18 UC 18 – Delete a Comment

Description**:** A Course Manager wants to delete an existing comment on a question or to a comment. Once deleted, all reactions to the comment are removed, and the comment is replaced with a message stating: *"This comment has been deleted"*. Replies to the deleted comment remain visible.

Actor: Course Manager / System Manager

Parameters: None

Pre-conditions:

1. The Course Manager is logged into the platform.

2. The comment exists in the system.

Post-Conditions:

1. The comment is marked as deleted in the system.
2. All reactions to the comment are removed.
3. A placeholder message (*"This comment has been deleted"*) replaces the comment content.
4. Replies to the deleted comment remain accessible and visible.

Main success scenario:

1. The Course Manager navigates to the question page.
2. The Course Manager locates the "Delete Comment" button and clicks it.
3. The system prompts the Course Manager for confirmation (e.g., *"Are you sure you want to delete this comment?"*).
4. The Course Manager confirms the deletion.
5. The system performs the following actions:
   I. Marks the comment as deleted in the database.
   II. Removes all reactions associated with the comment.
   III. Replaces the comment's content with the placeholder message: *"This comment has been deleted"*.
6. The deleted comment and its placeholder are visible to all users. Replies to the deleted comment remain visible.
7. The system confirms the comment has been successfully deleted (e.g., *"Comment deleted successfully"*).

Alternative flows:

1. **Deletion Not Confirmed:** If the Course Manager chooses not to confirm the deletion, the system cancels the deletion process, the comment remains unchanged.

### 1.19 UC 19 – Appoint a User as a Course Manager

Description**:** A Course Manager wants to appoint an existing user as an additional Course Manager for a specific course. Once the appointment is initiated, the user receives a notification to either accept or decline the role.

Actor: Course Manager / System Manager

Parameters: user's email

Pre-conditions:

1. The Course Manager is logged into the platform.
2. The user exists in the system.
3. The course exists in the system.
4. The course manager is a manager in this course.

Post-Conditions:

1. If the user accepts the appointment:
   I. The user is added as a Course Manager for the course.
   II. The user gains all permissions associated with a Course Manager role for that course.
2. If the user declines the appointment:
   I. The appointment request is cancelled, and no changes are made to the user's role.

Main success scenario:

1. The Course Manager navigates to the course page.
2. The Course Manager click on the "Appoint new Course Manager" button.
3. The Course Manager submits the user's email.
4. The system displays a confirmation dialog (e.g., *"Are you sure you want to appoint this user as a Course Manager?"*).
5. The Course Manager confirms the appointment.

6. The system sends a notification to the user stating: *"You have been appointed as a Course Manager for [Course Name]. Do you accept this role?"*
7. The user views the notification and chooses to accept or decline:
8. If the user accepts:
    a. The system updates the user's role to include Course Manager permissions for the specific course.
    *b*. The system displays a confirmation message to the user: *"You are now a Course Manager for [Course Name]."*
9. If the user declines:
    a. No changes are made to the user's role.

Alternative flows:

1. **User Already a Course Manager**: If the selected user is already a Course Manager for the course, the system will show relevant message.
2. **Invalid User**: The email doesn't exist in the system.

### 1.20 UC 20 – Remove a Course Manager

Description**:** A System Manager wants to remove a user from their role as a Course Manager for a specific course. Once removed, the user loses all Course Manager permissions for that course, and they receive a notification informing them of the removal.

Actor: System Manager

Parameters: course id, user’s email

Pre-conditions:

1. The System Manager is logged into the platform.
2. The user exists in the system.
3. The course exists in the system.

4. The user is currently assigned as a Course Manager for the course.

Post-Conditions:

1. The user is no longer a Course Manager for the specified course.
2. The user receives a notification about the removal.
3. All Course Manager-specific permissions for that course are revoked.

Main success scenario:

1. The System Manager selects the "Manage Course Managers" option for the desired course.
2. The System Manager identifies the user to be removed and clicks the "Remove Course Manager" button.
3. The system prompts the System Manager for confirmation (e.g., *"Are you sure you want to remove [User Name] as a Course Manager for [Course Name]?"*).
4. The System Manager confirms the removal.
5. The system performs the following actions:
   I. Revokes the user's Course Manager role for the specified course.
   II. Sends a notification to the user informing them of the removal (e.g., *"You have been removed as a Course Manager for [Course Name]."*).
6. The System Manager sees a confirmation message: *"The user has been successfully removed as a Course Manager for [Course Name]."*

Alternative flows:

1. **User is Not a Course Manager**: f the selected user is not a Course Manager for the specified course. The system notifies the System Manager: *"[User Name] is not a Course Manager for [Course Name]."*

### 1.21 UC 21 – Download a ZIP of All Exams of a Course

Description**:** A user wants to download a ZIP file containing all exam files for a specific course.

Actor: User / Course Manager / System Manager

Parameters: None

Pre-conditions:

1. The user is logged into the platform.
2. The course exists in the system.
3. The course has at least one exam with uploaded files.

Post-Conditions:

1. A ZIP file is generated containing all available exams for the course.
2. The user receives the ZIP file as a download.

Main success scenario:

1. The user navigates to the course page.
2. The user selects the "Download All Exams" option.
3. The system processes the request and performs the following:
    I. Locates all exams associated with the course.
    II. Organizes files into a ZIP
4. The system generates a ZIP file and sends it as a downloadable response.
5. The user receives the ZIP file and saves it locally.

Alternative flows:

1. **No Exams in this course yet**: The user tries to download a zip of all the exams of a course, however, there are no exams of this course it the system yet, the system will show relevant message.

# 2. System Architecture

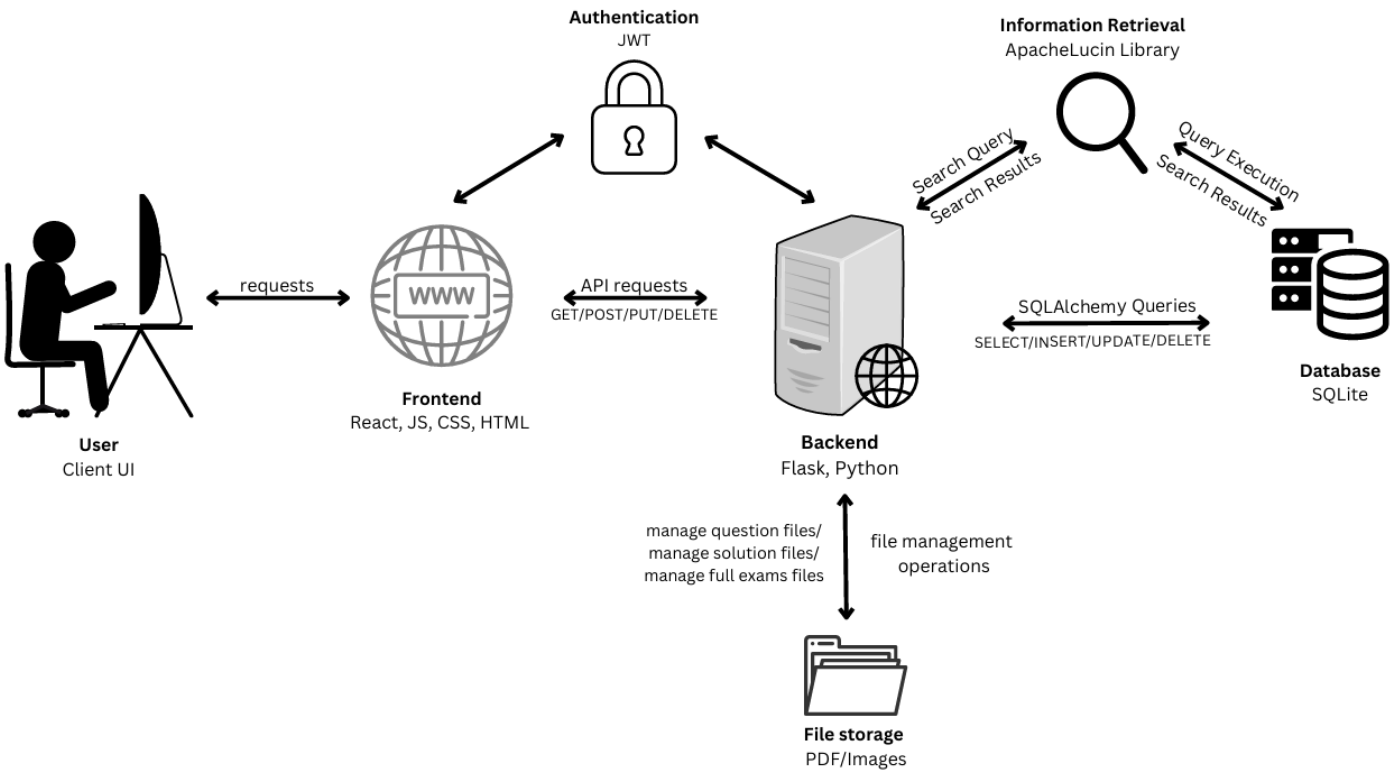



*Figure 2: System Architecture*

NegevNerds system contains 6 main components:

1. **<u>Frontend Component (Client-side):</u>**

   The Frontend is the user interface of the system, built using React. It provides an interactive experience where users can submit queries, view results, and interact with content. Deployed client-side, it runs in the user's web browser and communicates with the Backend through REST APIs and HTTP requests (via Axios). The Frontend manages the states dynamically using React hooks like useState and useEffect, and sends GET, POST, PUT, and DELETE requests to the Backend.

2. **<u>Backend component (Server-side):</u>**

   The Backend is built with Flask and handles API requests, business logic, and interactions between the Frontend, Database, Authentication, and File Storage. Deployed on a remote server, it processes requests, manages user

authentication using JWT, and handles data transactions with the Database and File Storage. It exposes RESTful APIs, implements business logic, and manages file uploads and retrievals.

3. **Authentication component:**

   The Authentication component ensures secure user login and access control using JWT. Upon login, a JWT token is generated and returned to the Frontend for secure storage. The Frontend sends the token with each request to the Backend, which verifies it before granting access to protected routes. This system eliminates the need for server-side sessions, ensuring secure and efficient user verification.

4. **Information Retrieval component:**

   The Information Retrieval component handles search queries and retrieves relevant data based on user input. It communicates with the Database to fetch results and can implement indexing algorithms by Apache Lucene for faster search performance. The component processes search queries, rank the results, and returns them to the Frontend for display.

5. **Database component:**

   The Database stores all persistent data, including questions, answers, and user information. Managed by the Backend, it is accessed via SQLAlchemy (an ORM) to perform data operations. The Database ensures data integrity, fast retrieval, and efficient querying. It is deployed on the same server as the Backend and is responsible for storing, retrieving, and updating system data.

6. **File Storage component:**

   The File Storage component manages files like PDFs and images that contain questions, solutions, or exams. These files are stored on the server and are accessible when needed. The Backend handles the uploading and retrieval of files from the File Storage, ensuring secure storage and easy access for the Frontend.

# 3. Data Model

## a. Description of Data Objects

### 3.1 CommentModel

Represents a comment entity in the system. A comment is associated with a specific question and may have reactions (Emojis).

**Attributes:**

- **comment_id** (Primary Key): String

  A unique identifier for the comment.
- **writer_name**: String

  The name of the user who wrote the comment. This field is required.
- **writer_id** (Foreign Key): String

  A unique identifier for the user who wrote the comment. This field is required.
- **Date**: DateTime

  The date the comment was created. This field is required.
- **prev_id**: String

  The identifier of the previous comment (if applicable), which could indicate a threaded or sequential comment structure. This field is required.
- **text**: String

  The content of the comment. This field is required.
- **Deleted**: Boolean

  A boolean flag indicating whether the comment has been marked as deleted. This field is required.
- **question_id**: String (Foreign Key)

  The unique identifier of the question this comment is associated with. This field references the question_id attribute in the QuestionModel.

## 3.2. CourseModel

Represents a course entity within the system. A course has users, topics, managers, and exams associated with it.

**Attributes:**

- **course_id** (Primary Key): String
  A unique identifier for the course.
- **name:** String
  The name of the course. This field is required.

## 3.3. CourseManagersModel

Represents the relationship between a course and its managers. This is a many-to-many association table linking courses and users who manage them.

**Attributes:**

- **course_id** (Primary Key, Foreign Key): String
  The unique identifier of the course being managed. This references the course_id in the CourseModel.
- **user_id** (Primary Key, Foreign Key): String
  The unique identifier of the user managing the course. This references the user_id in the UserModel.

## 3.4. CourseTopicsModel

Represents the relationship between a course and its syllabus topics. This model associates specific topics with the courses they belong to.

**Attributes:**

- **course_id** (Primary Key, Foreign Key): String
  The unique identifier of the course associated with the topic. This references the course_id in the CourseModel.

- **topic** (Primary Key): String

  A topic related to the course. This field is required and serves as part of the primary key.

## 3.5. ExamModel

Represents an exam entity in the system. Each exam is associated with a specific course and may have multiple questions.

**Attributes:**

- **year** (Primary Key): int

  The year in which the exam took place.
- **semester** (Primary Key): String

  The semester in which the exam was held (e.g. סתיו, אביב, קיץ).
- **moed** (Primary Key): String

  The specific exam "moed" (e.g. א, ב, ג, ד).
- **exam_id** (Unique): String

  A unique identifier for the exam.
- **course_id** (Foreign Key): String

  The unique identifier of the course this exam belongs to. This references the course_id in the CourseModel.
- **link**: Sting

  A link to the exam file in the server (optional).

## 3.6. Letter{letter}Model

Represents a table for storing words start with a specific letter (for each letter in the English and Hebrew alphabet), appear in the text of the corresponding question.

**Attributes:**

- **word** (Primary Key): String

  The word associated with the letter.

- **question_id** (Primary Key): String

  The unique identifier of the question associated with the word.

## 3.7. NotificationModel

Represents a notification entity in the system. Notifications are sent between users and may require approval.

**Attributes:**

- **notification_id** (Primary Key): String

  A unique identifier for the notification.
- **sender_user_id** (Foreign Key): String

  The unique identifier of the user who sent the notification. This references the user_id in the UserModel.
- **receiver_user_id** (Foreign Key): String

  The unique identifier of the user who received the notification. This also references the user_id in the UserModel.
- **message**: Sting

  The content of the notification. This field is required.
- **timestamp**: DateTime

  The date and time when the notification was created.
- **need_approval**: Boolean

  A boolean flag indicating whether the notification requires approval by the receiver.

## 3.8. QuestionModel

Represents a question entity in the system. A question is part of an exam and can have topics, comments, and related resources.

**Attributes:**

- **question_id**: String

  A unique identifier for the question.

- **year**: int

  The year the question was part of an exam. This field is required.

- **text**: String

  The text or content of the question. This field is required.

- **semester**: String

  The semester in which the question appeared (e.g. סתיו , אביב ,קיץ) This field is required.

- **moed**: String

  The specific exam "Moed" (e.g. א, ב, ג, ד.) This field is required.

- **question_number** (Primary Key): int

  The number of the question within the exam.

- **is_american**: Boolean

  A boolean flag indicating whether the question is a multiple-choice (American-style) question.

- **link_to_question:** String

  A link to the question's file in the server.

- **link_to_exam**: String

  A link to the exam file in the server(optional).

- **link_to_answer**: String

  A link to the solution file for the question in the server (optional).

- **exam_id** (Primary Key, Foreign Key): String

  The unique identifier of the exam this question belongs to. This references the exam_id in the ExamModel.

## 3.9. QuestionTopicsModel

Represents the relationship between a question and its associated topics. This model links questions to the topics they are related to.

**Attributes:**

- **question_id** (Primary Key, Foreign Key): Sting

  The unique identifier of the question associated with the topic. This references the question_id in the QuestionModel.

- **topic** (Primary Key): String

  A topic related to the question.

## 3.10. ReactionModel

Represents an emoji reaction to a comment from a user.

**Attributes:**

- **reaction_id** (Primary Key): String

  A unique identifier for the reaction.

- **user_id** (Foreign Key): String

  The unique identifier of the user who made the reaction. This references the user_id in the UserModel.

- **emoji**: String

  The emoji used in the reaction. This field is required.

- **comment_id** (Foreign Key): String

  The unique identifier of the comment being reacted to. This references the comment_id in the CommentModel.

## 3.11. UserModel

Represents a user in the system, storing essential user information and defining relationships with courses and course management.

**Attributes:**

- **user_id** (Primary Key): String

  A unique identifier for the user.

- **Email**: String

  The user's email address. This field must be unique and is required for user registration.

- **password**: String

  The user's encrypted password. This field is required for authentication.

- **first_name**: String

  The user's first name. This field is required.

- **last_name**: String

  The user's last name. This field is required.

- **logged_in**: Boolean

  A boolean flag indicating whether the user is currently logged in.

## 3.12. UserCoursesModel

Represents the relationship between a user and a course, indicating which users are enrolled in which courses.

**Attributes:**

- **user_id** (Primary Key, Foreign Key): String

  The unique identifier of the user enrolled in the course. This references the user_id in the UserModel.

- **course_id** (Primary Key, Foreign Key): String

  The unique identifier of the course the user is enrolled in. This references the course_id in the CourseModel.

## b. Data Objects Relationships

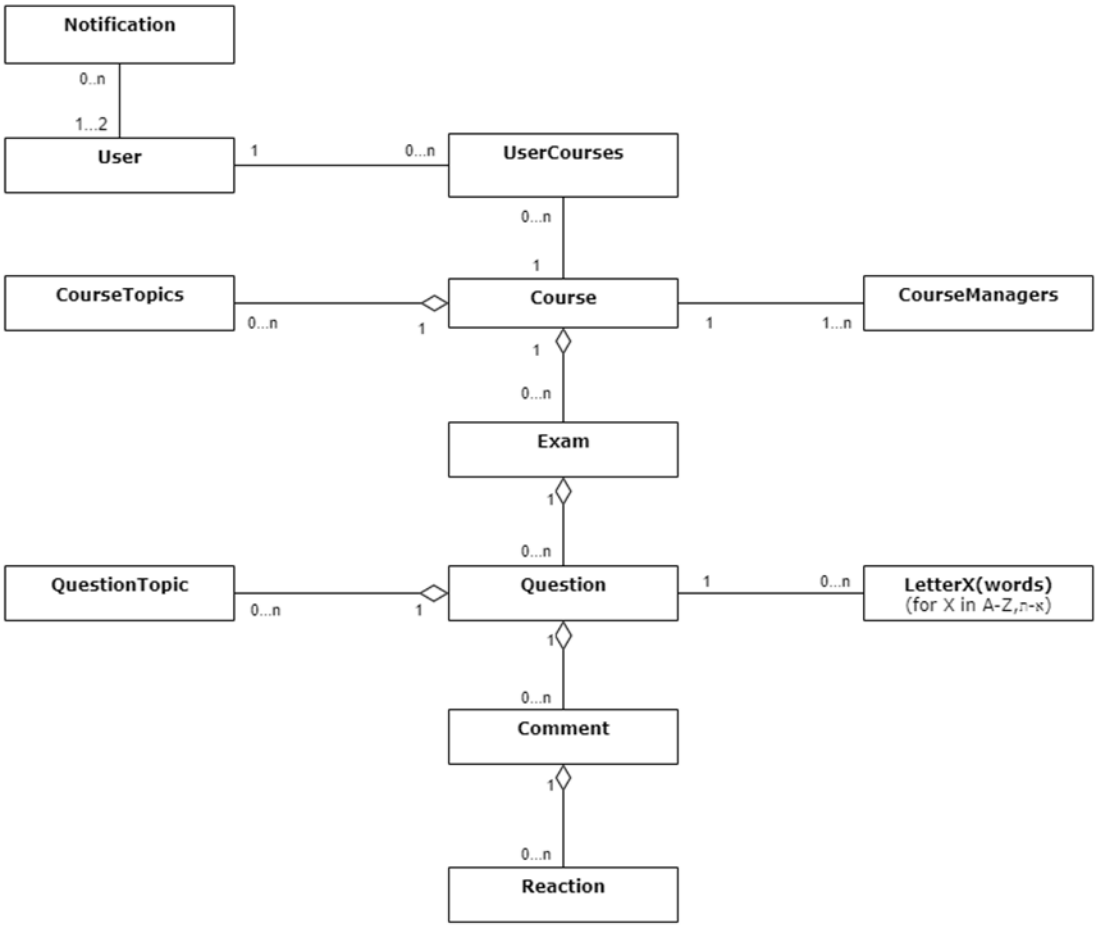
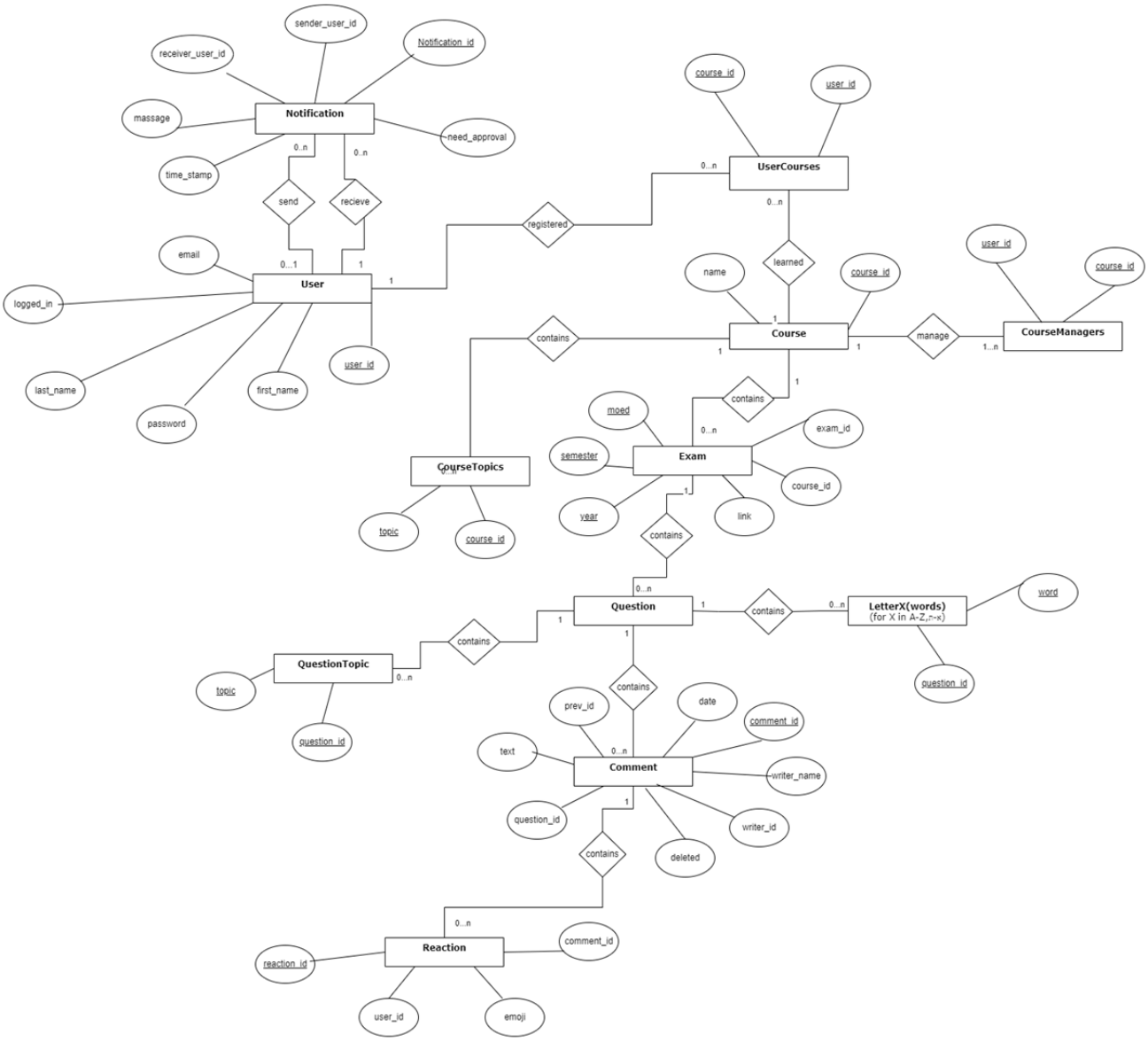



*Figure 3: Data Objects Relationships*

## c. Databases

*Figure 4: ERD*



# 4. Behavioural Analysis

## a. Sequence Diagram

### 4.1.1 Register a new User

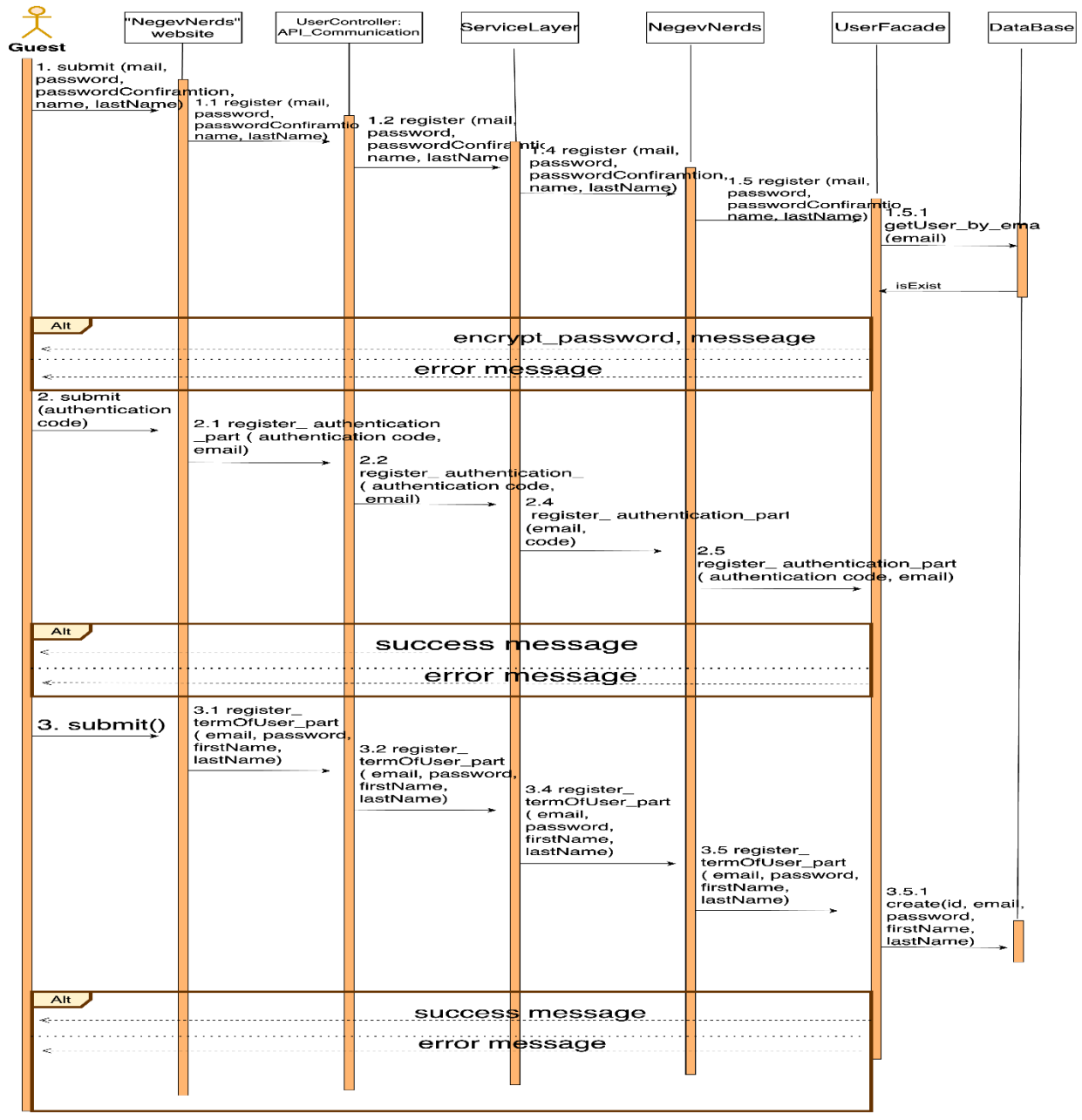



*Figure 5: Sequence Diagram of Register a new User*

### 4.1.2 Login

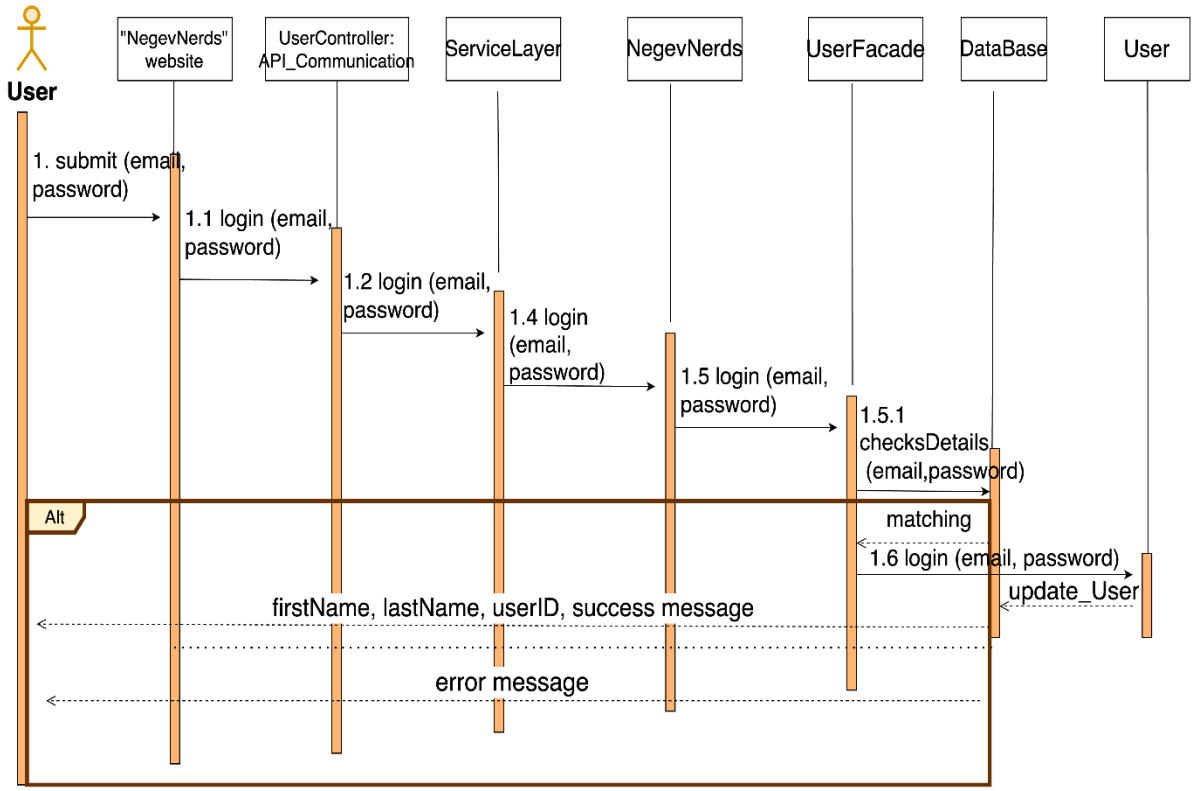



*Figure 6: Sequence Diagram of Login*

### 4.1.3 Register to Course

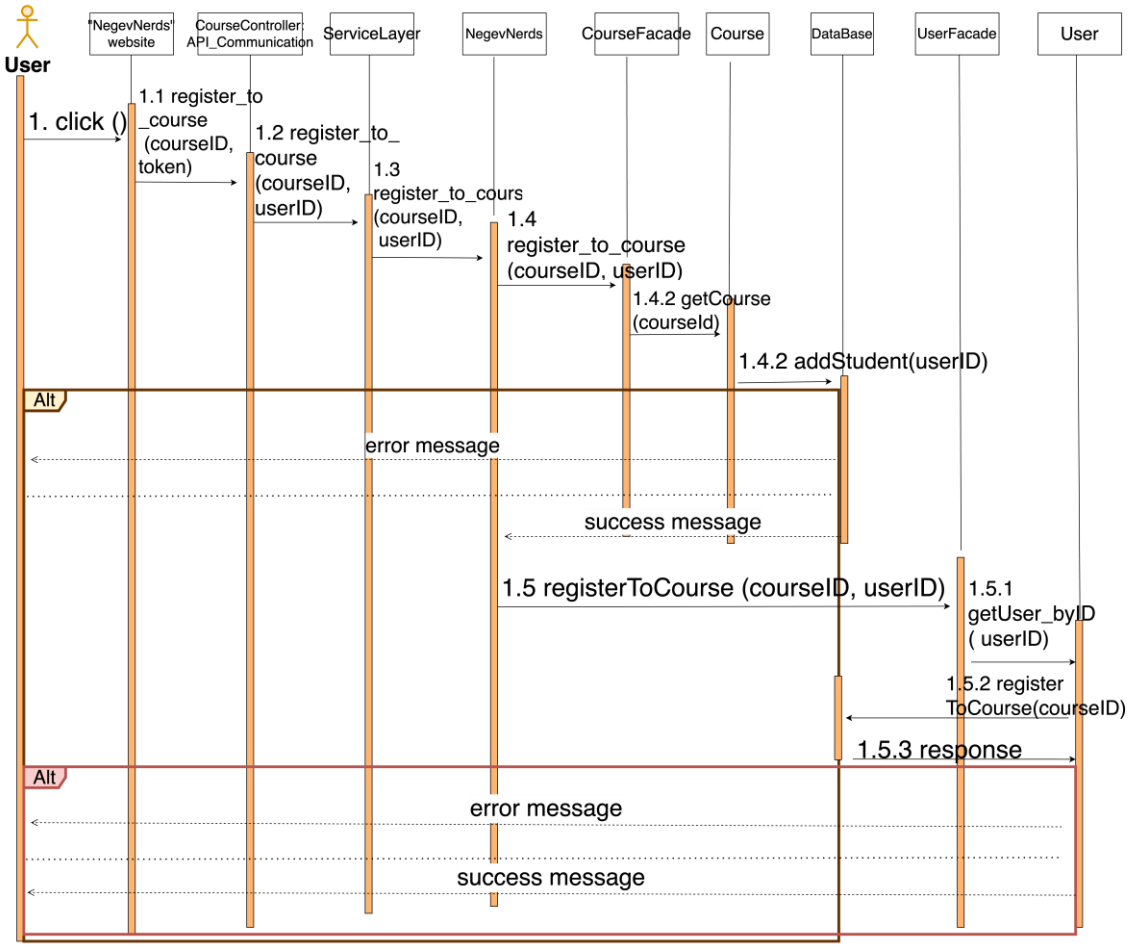



*Figure 7: Sequence Diagram of Register to Course*

### 4.1.4 Open a Course

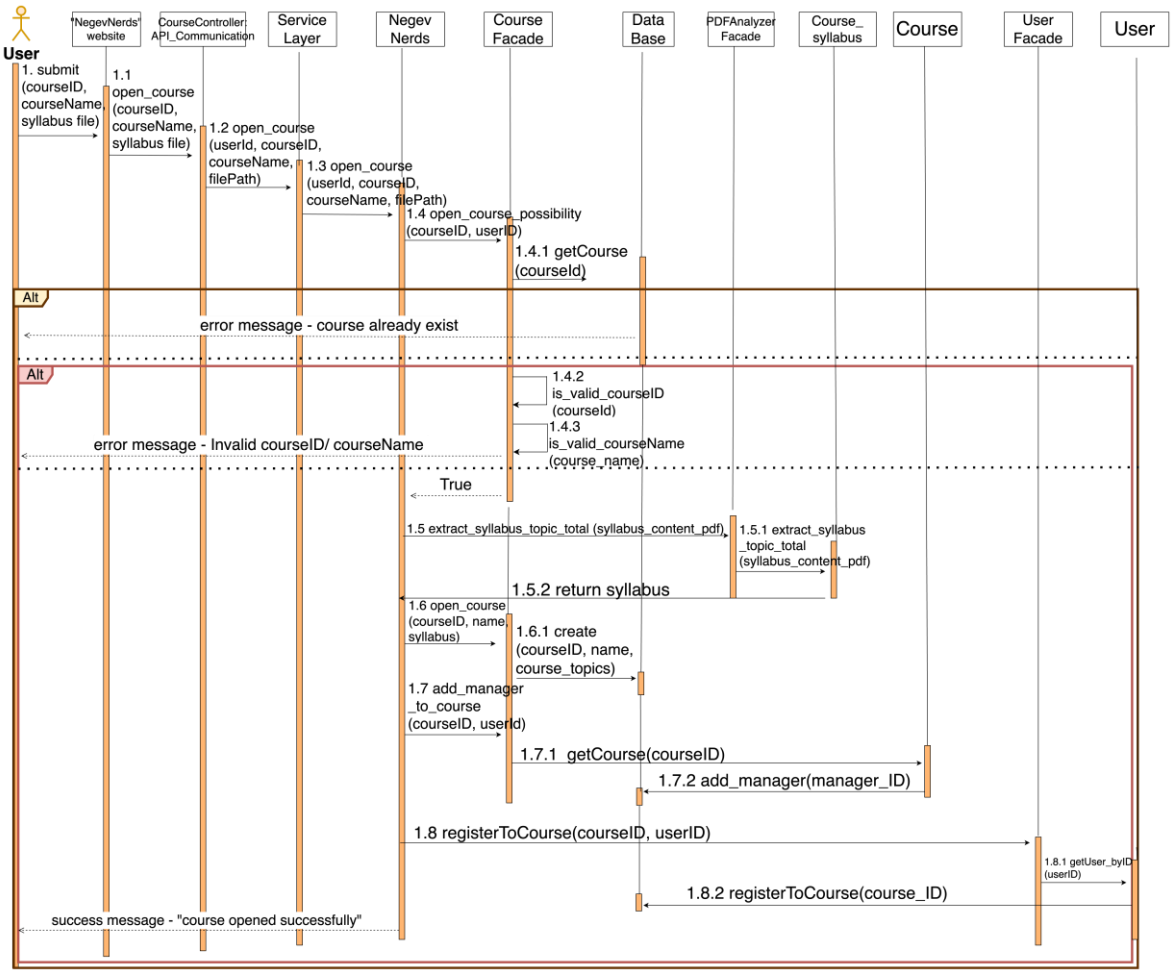



*Figure 8: Sequence Diagram of Open a Course*

### 4.1.5 Post an Exam

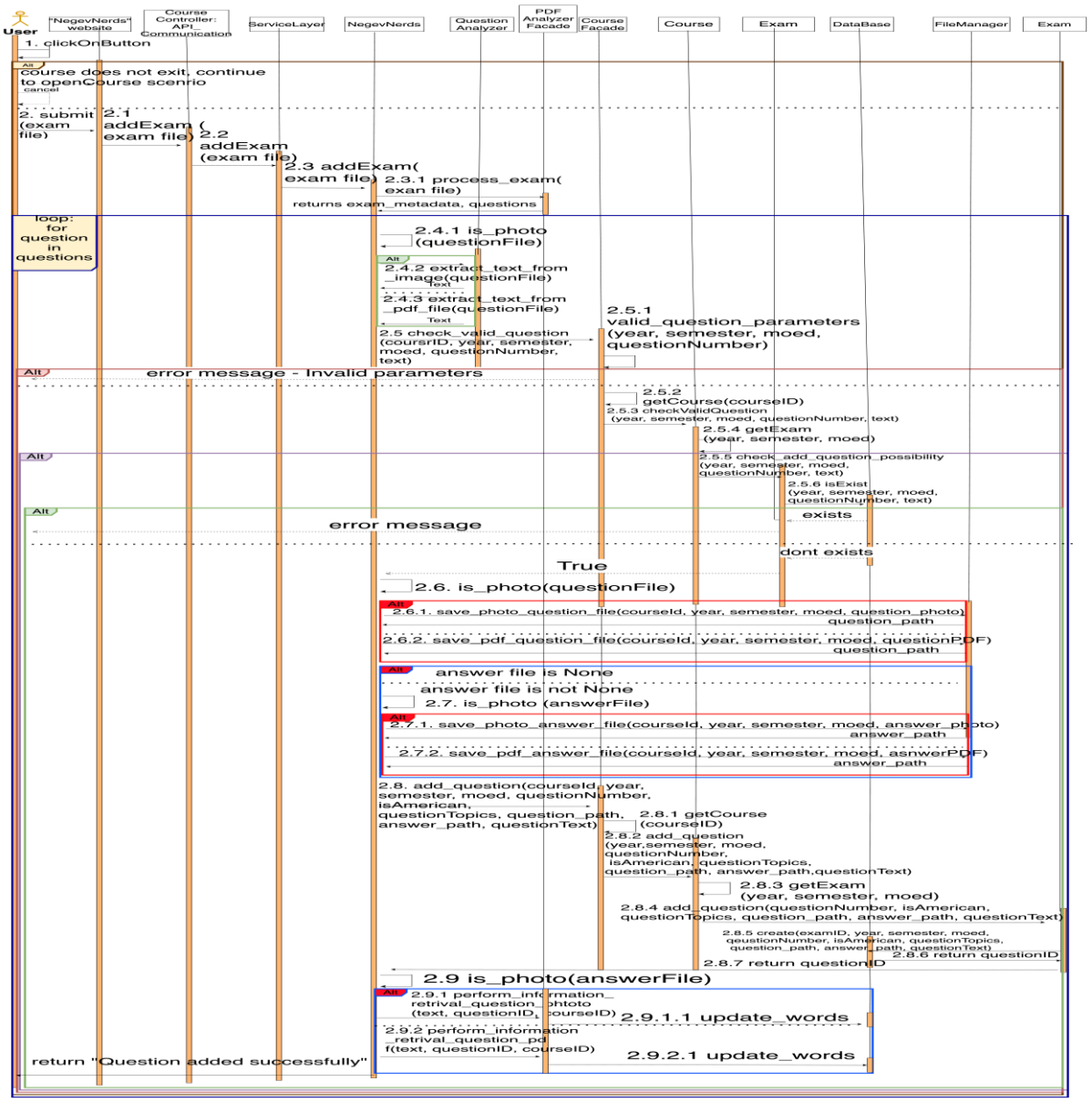



*Figure 9: Sequence Diagram of Post an Exam*

### 4.1.6 Post an Exam's Question

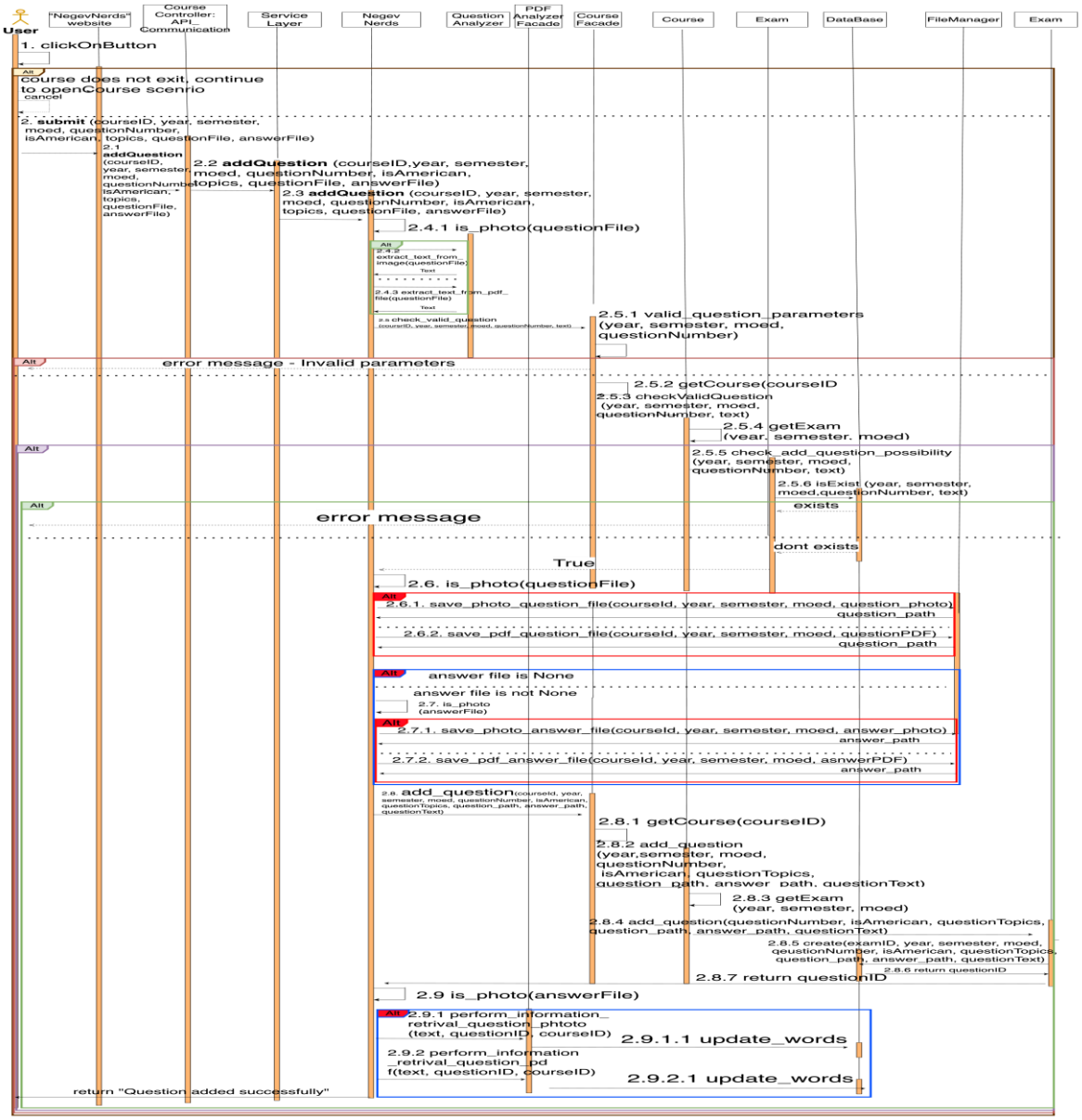



*Figure 10: Sequence Diagram of Post a Question*

### 4.1.7 Search Questions by Text

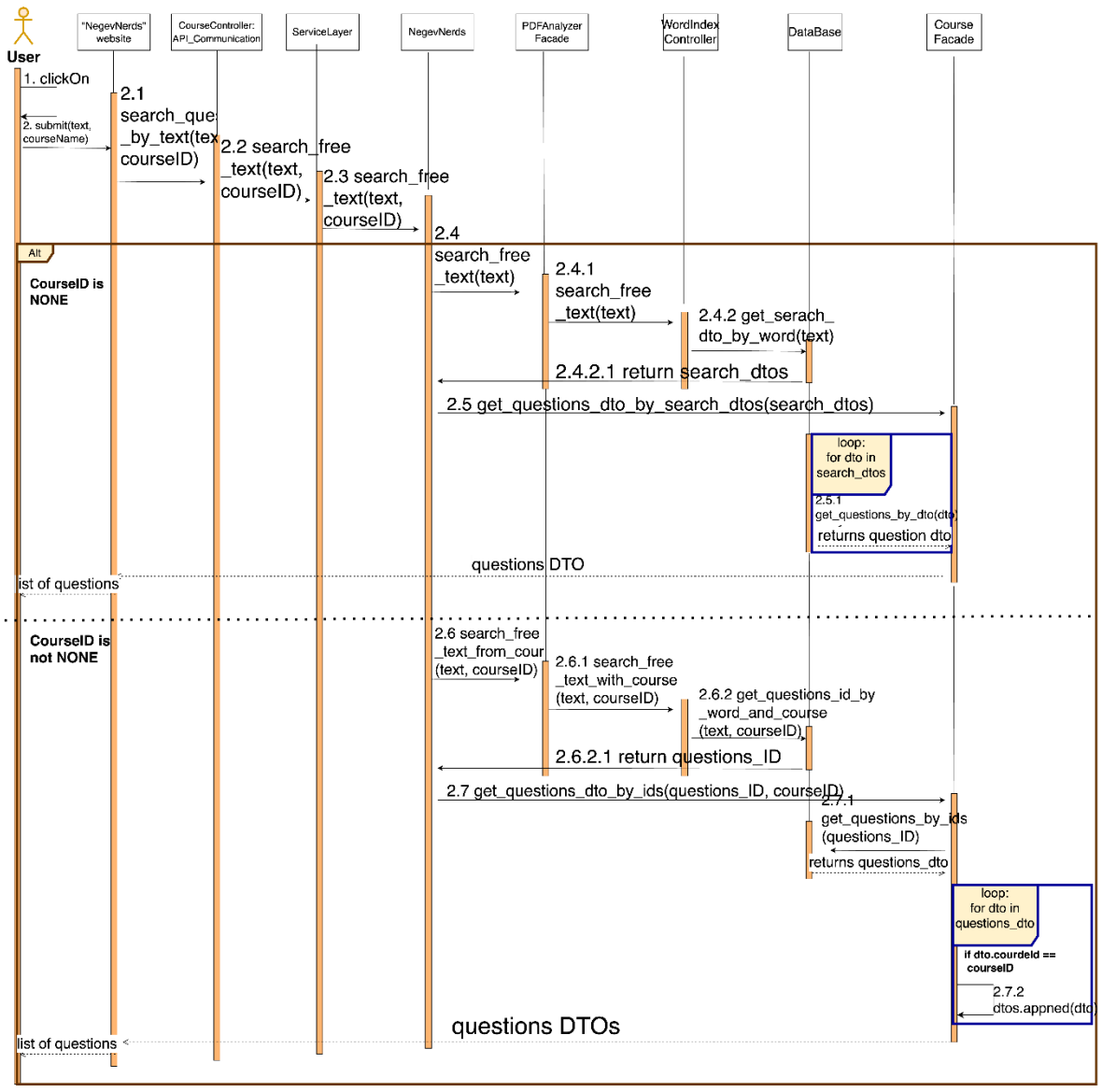



*Figure 11: Sequence Diagram of Search Question by Text*

### 4.1.8 Search Questions by Topic

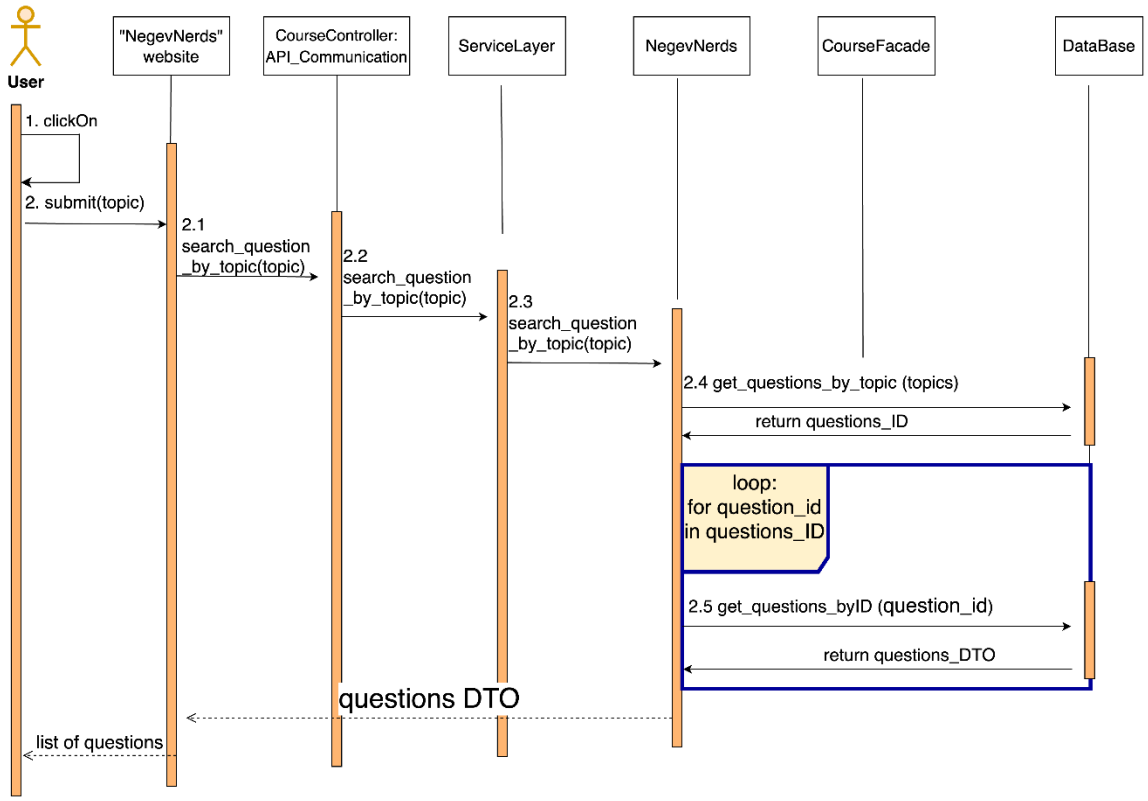



*Figure 12: Sequence Diagram of Search Question by Topic*

### 4.1.9 Search Question by Specifics

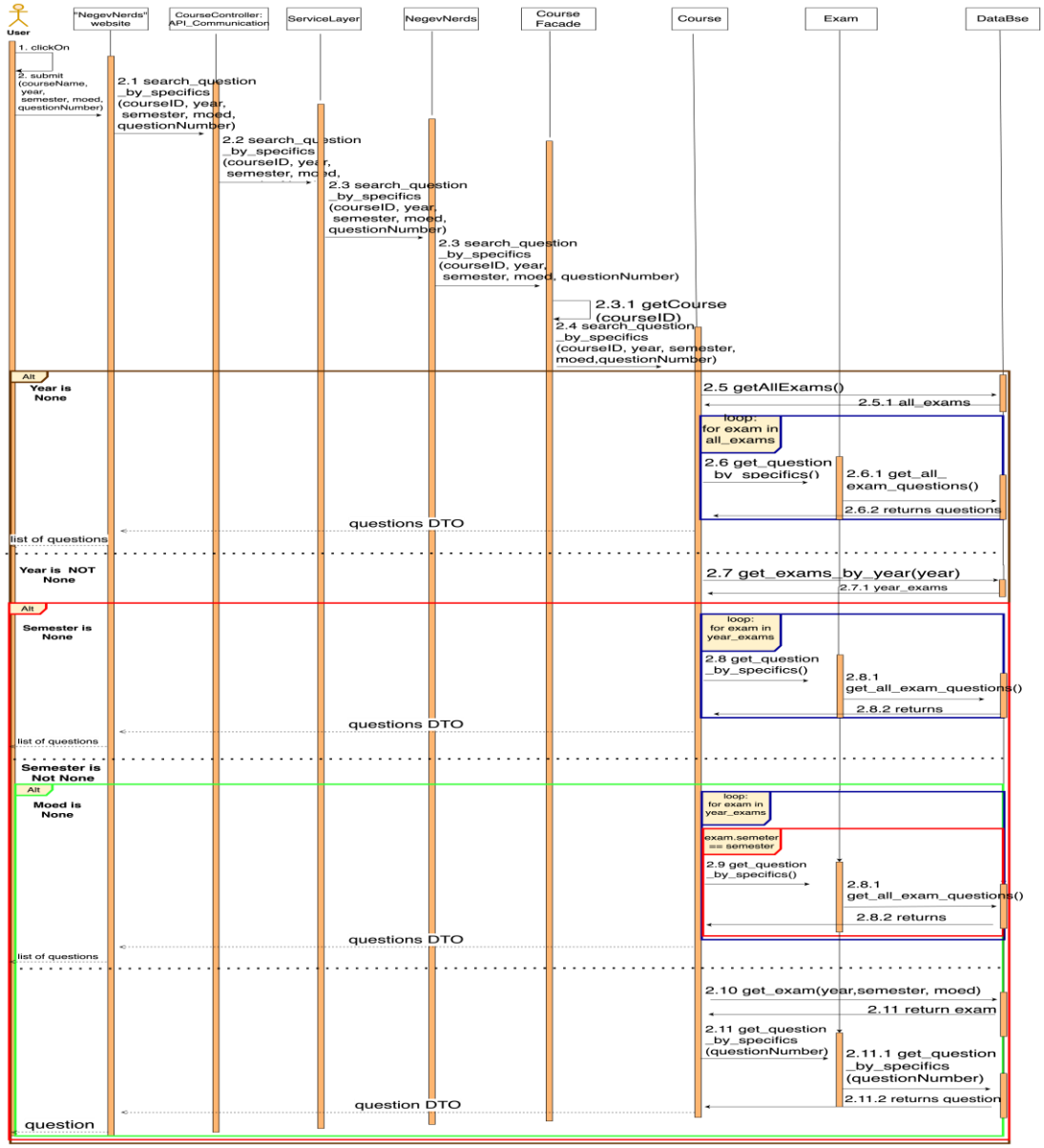



*Figure 13: Sequence Diagram of Search Question by Specifics*

## 4.1.10 Comment on a Question

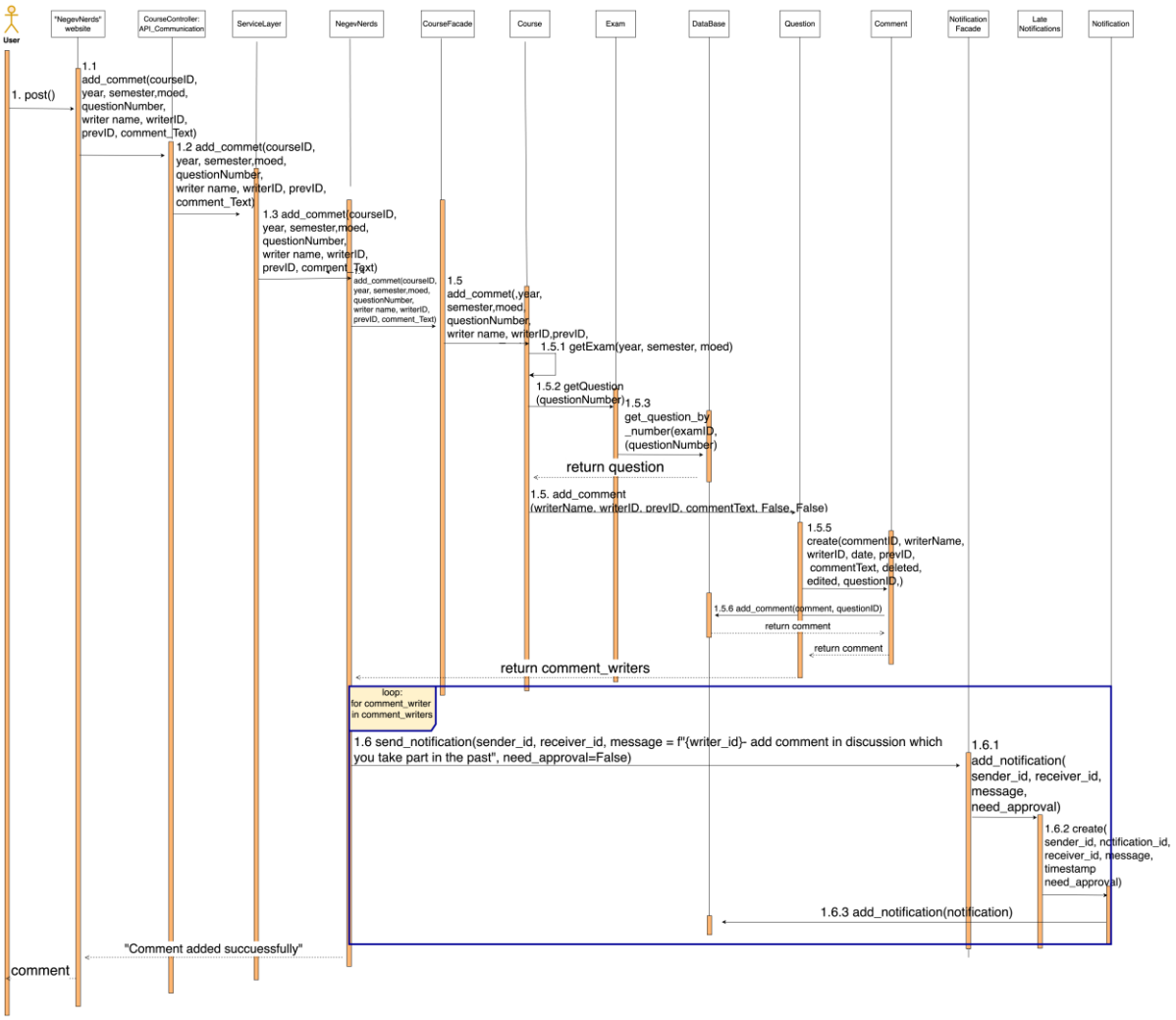



*Figure 14: Sequence Diagram of Comment on a Question*

### 4.1.11 Comment on a Comment

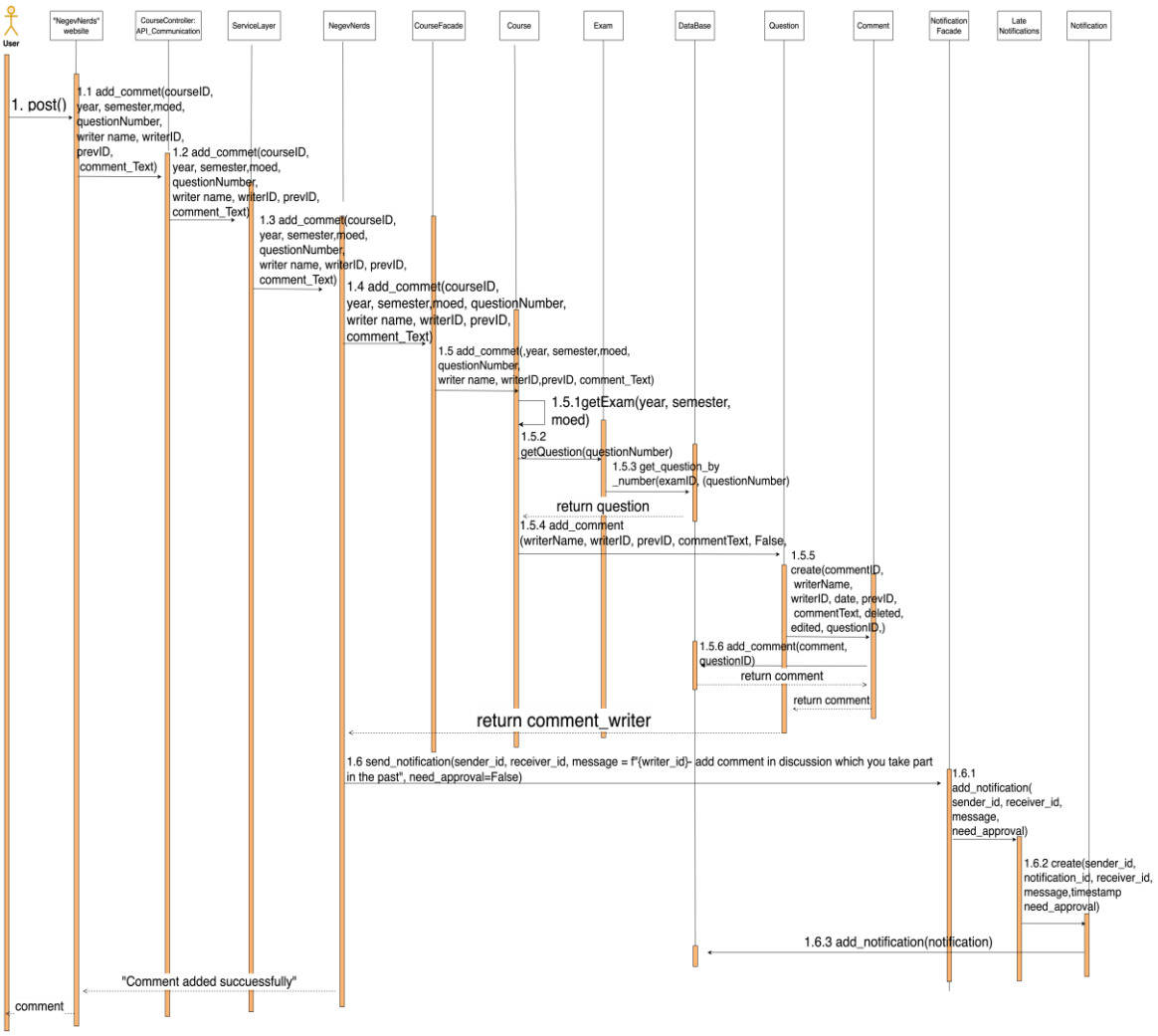



*Figure 15: Sequence Diagram of Comment on a Comment*

### 4.1.12 Edit a Comment

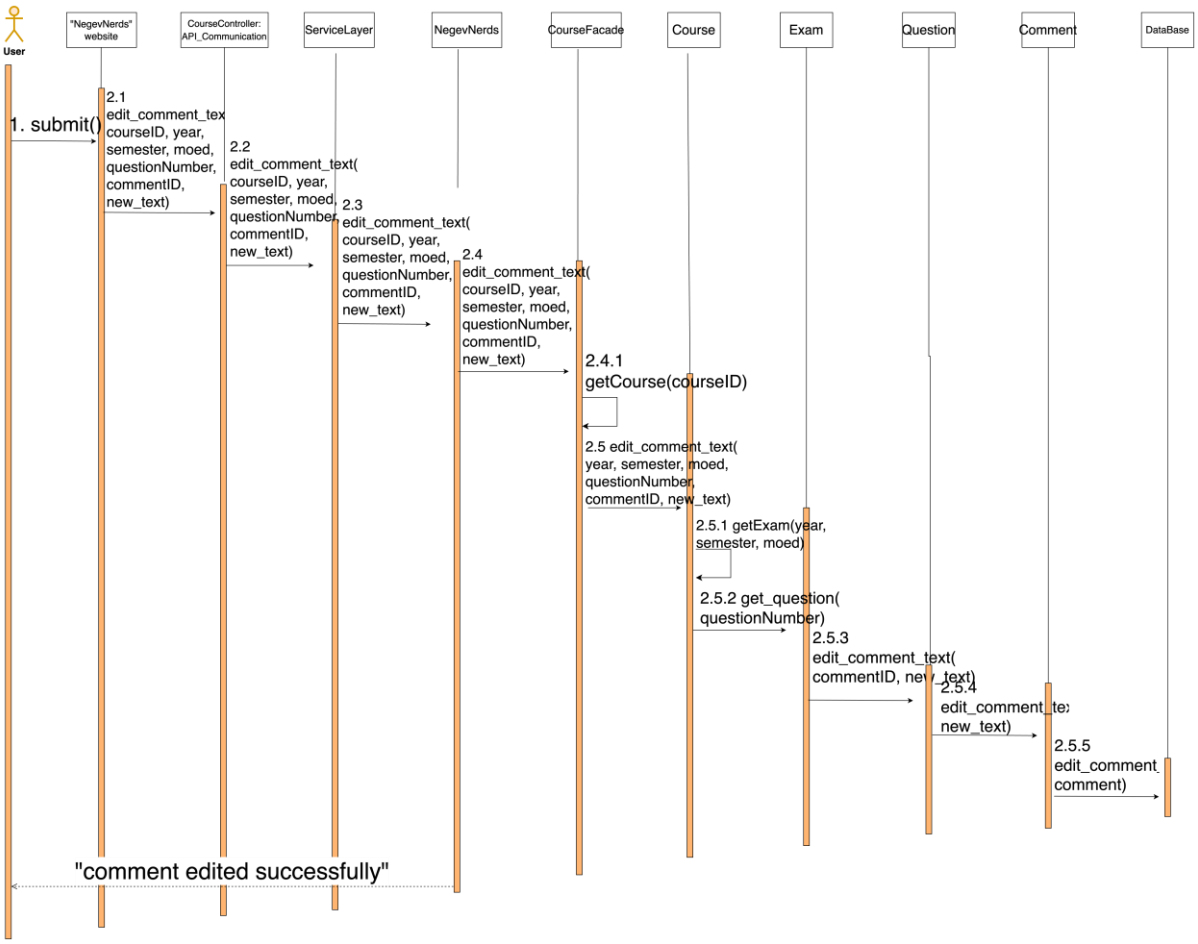



*Figure 16: Sequence Diagram of Edit a Comment*

### 4.1.13 Download Exam Full PDF File

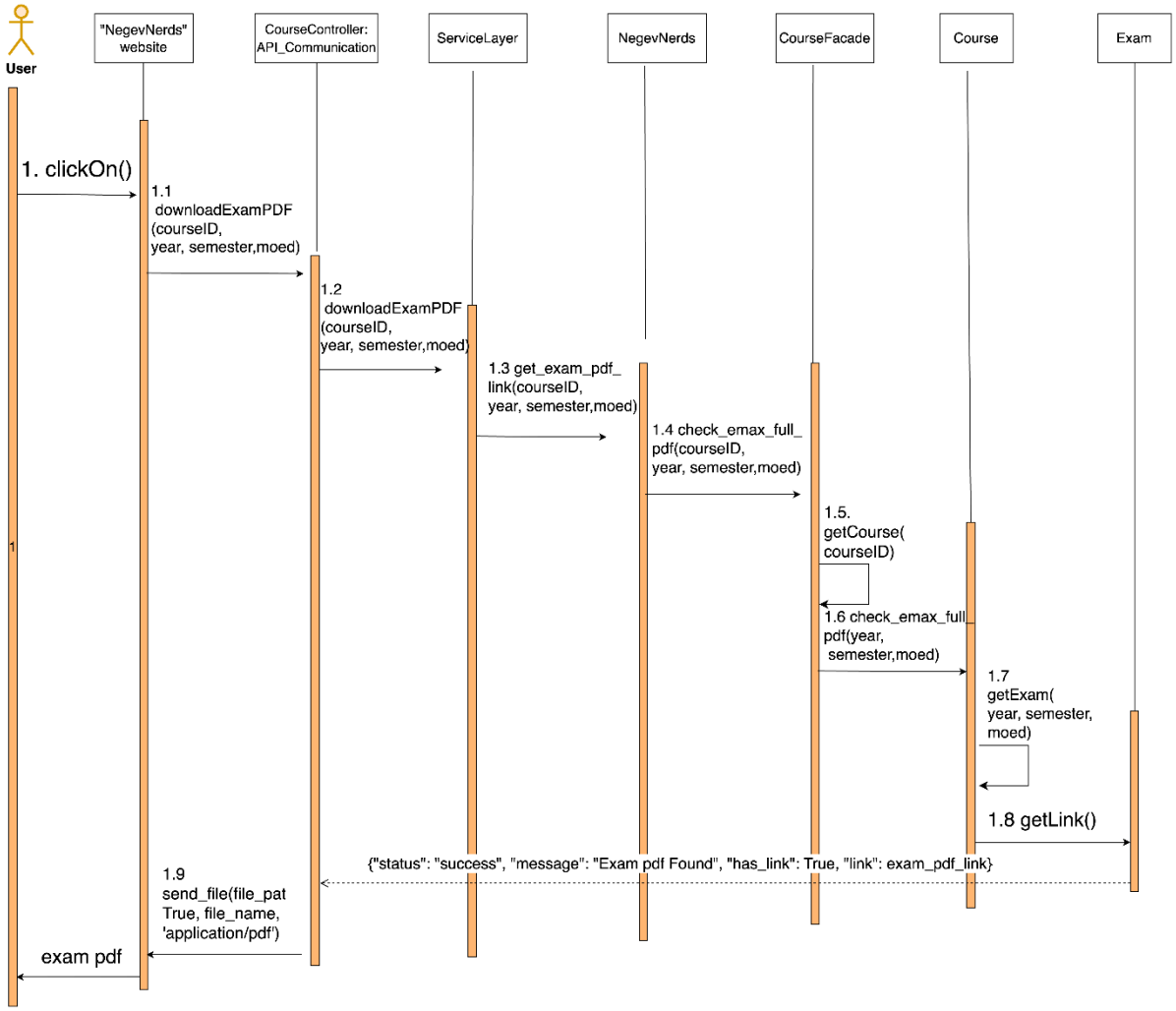



*Figure 17: Sequence Diagram of Download Exam Full PDF File*

### 4.1.14 Upload Exam Full PDF File

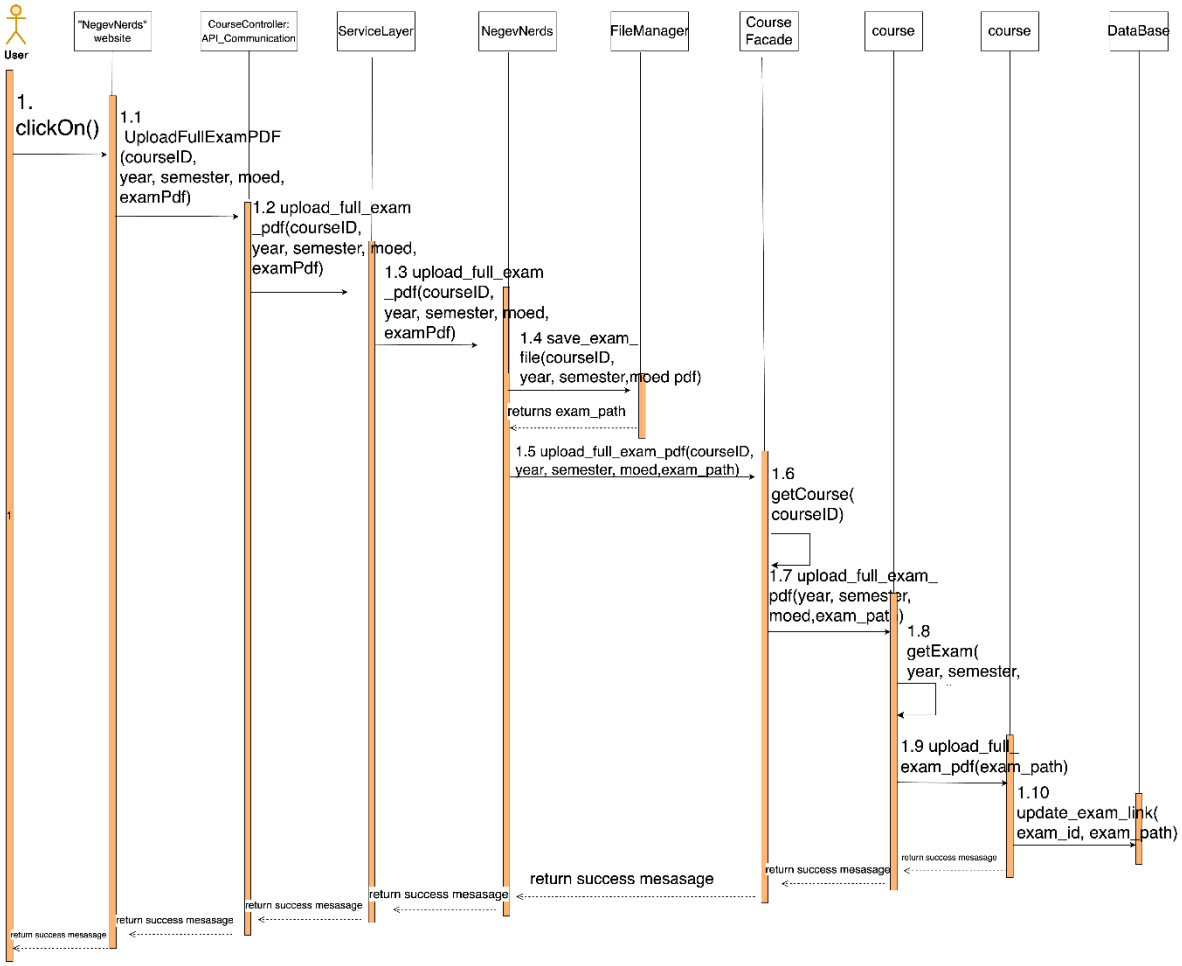



*Figure 18: Sequence Diagram of Upload Exam Full PDF File*

### 4.1.15 React to a Comment with an emoji

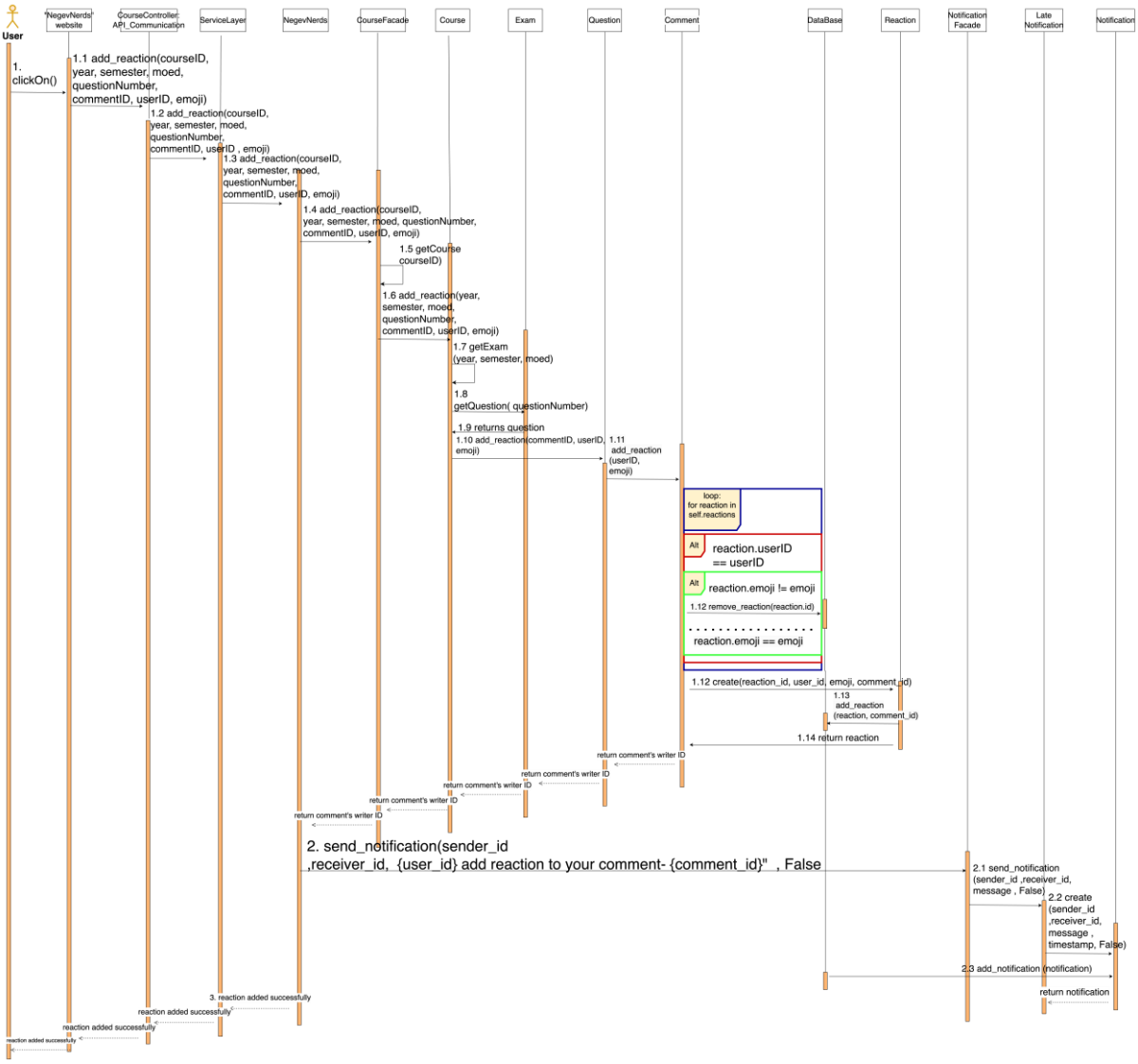



*Figure 19: Sequence Diagram of React to a Comment with an emoji*

### 4.1.16 Delete a Question

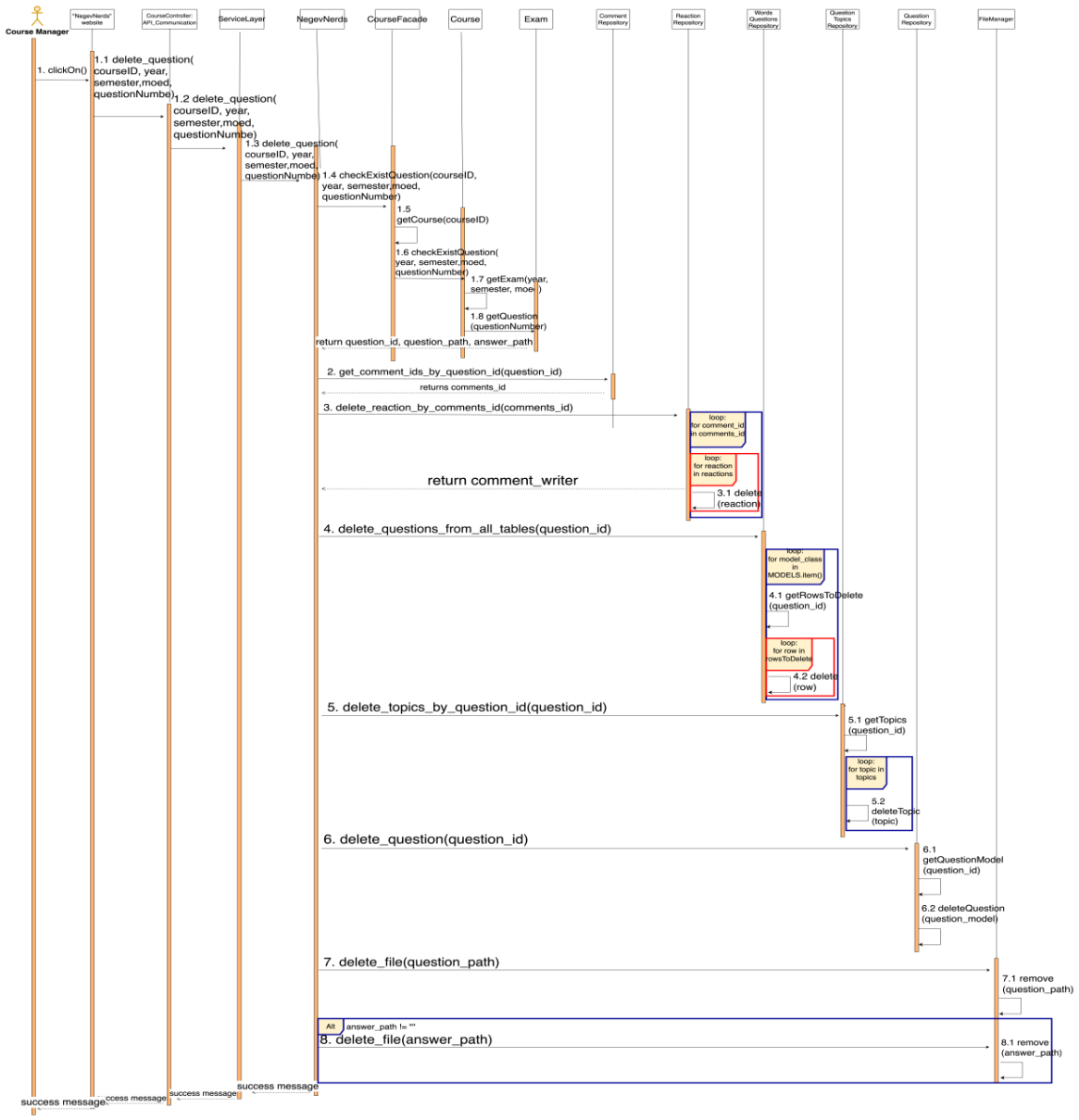



*Figure 20: Sequence Diagram of Delete a Question*

## 4.1.17 Edit a Question

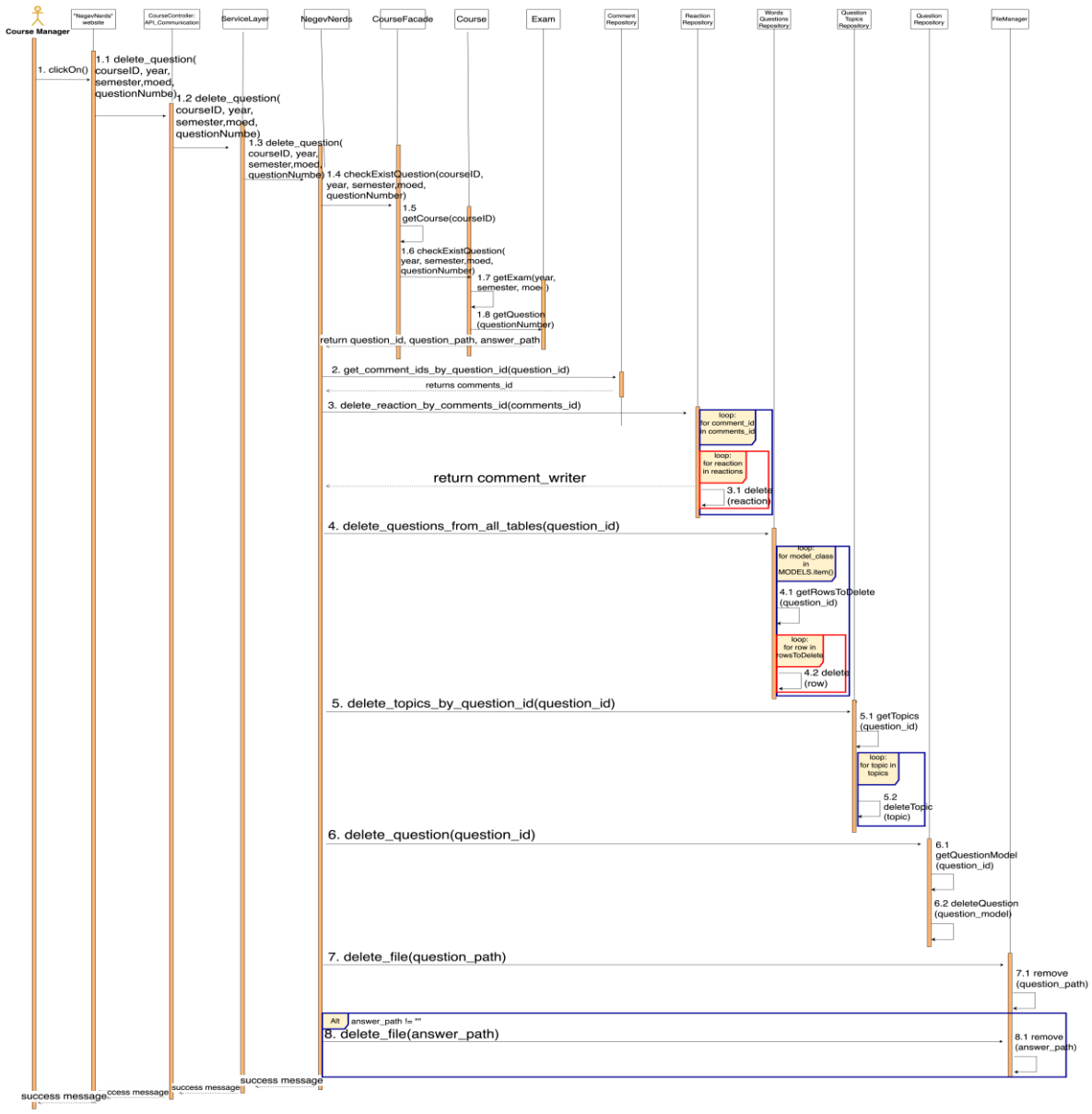



*Figure 21: Sequence Diagram of Edit a Question*

## 4.1.18 Delete a Comment

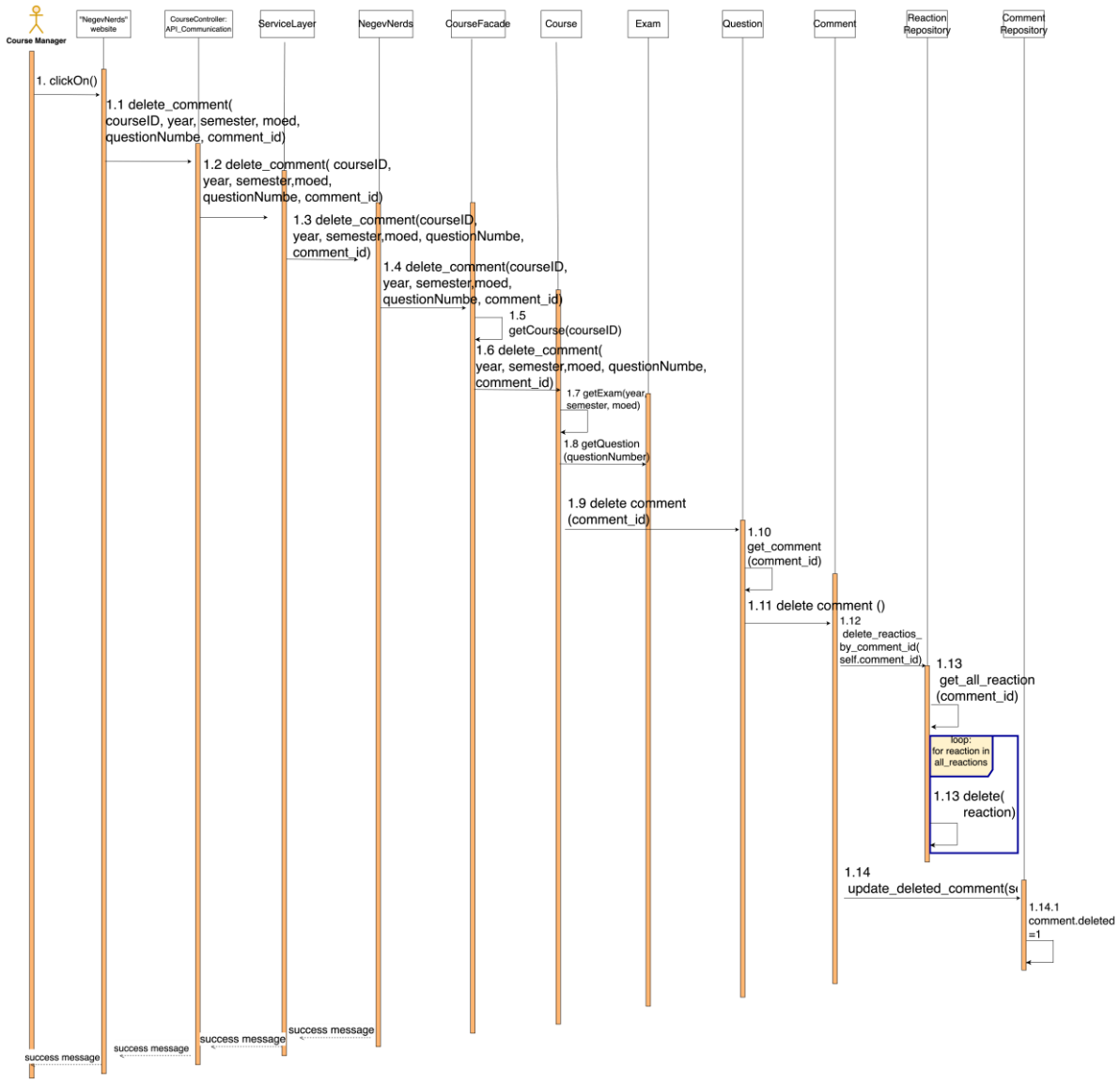



*Figure 22: Sequence Diagram of Delete a Comment*

### 4.1.19 Appoint a User as a Course Manager

Nominator side:

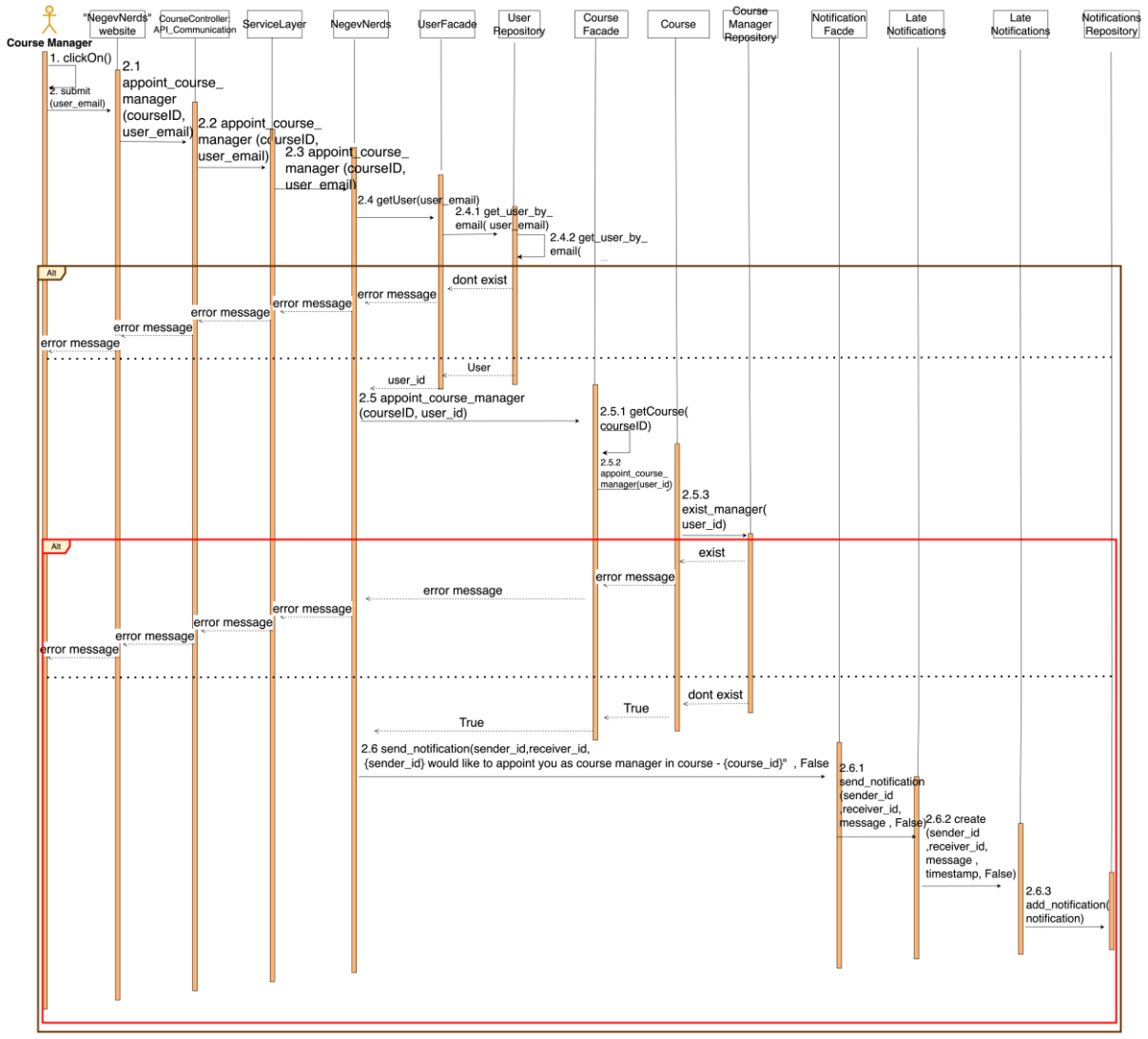



*Figure 23 Sequence Diagram of Appoint a User as a Course Manager, nominator side*

Nominee side:

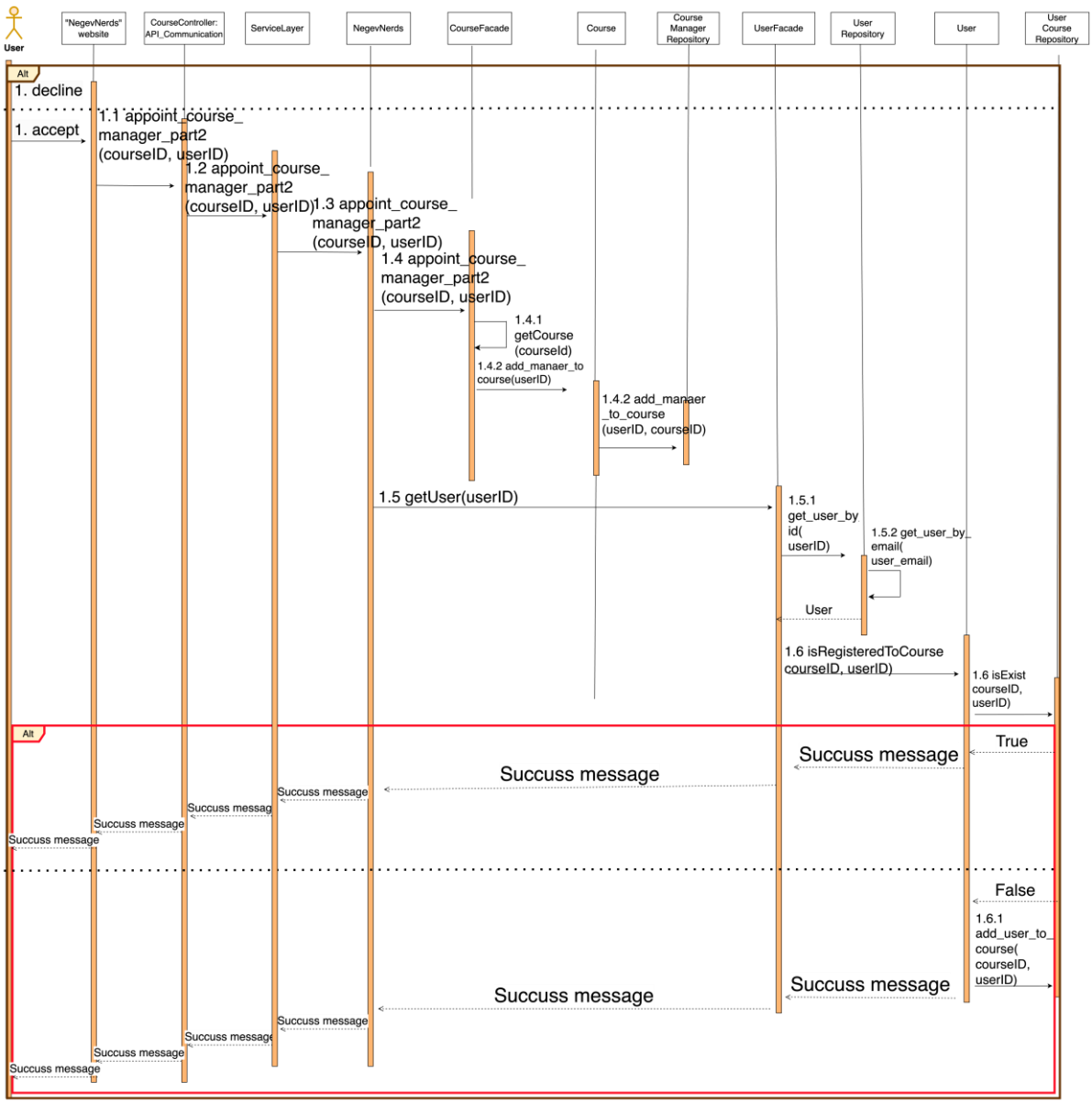



*Figure 24 Sequence Diagram of Appoint a User as a Course Manager, nominee side*

### 4.1.20 Remove a Course Manager

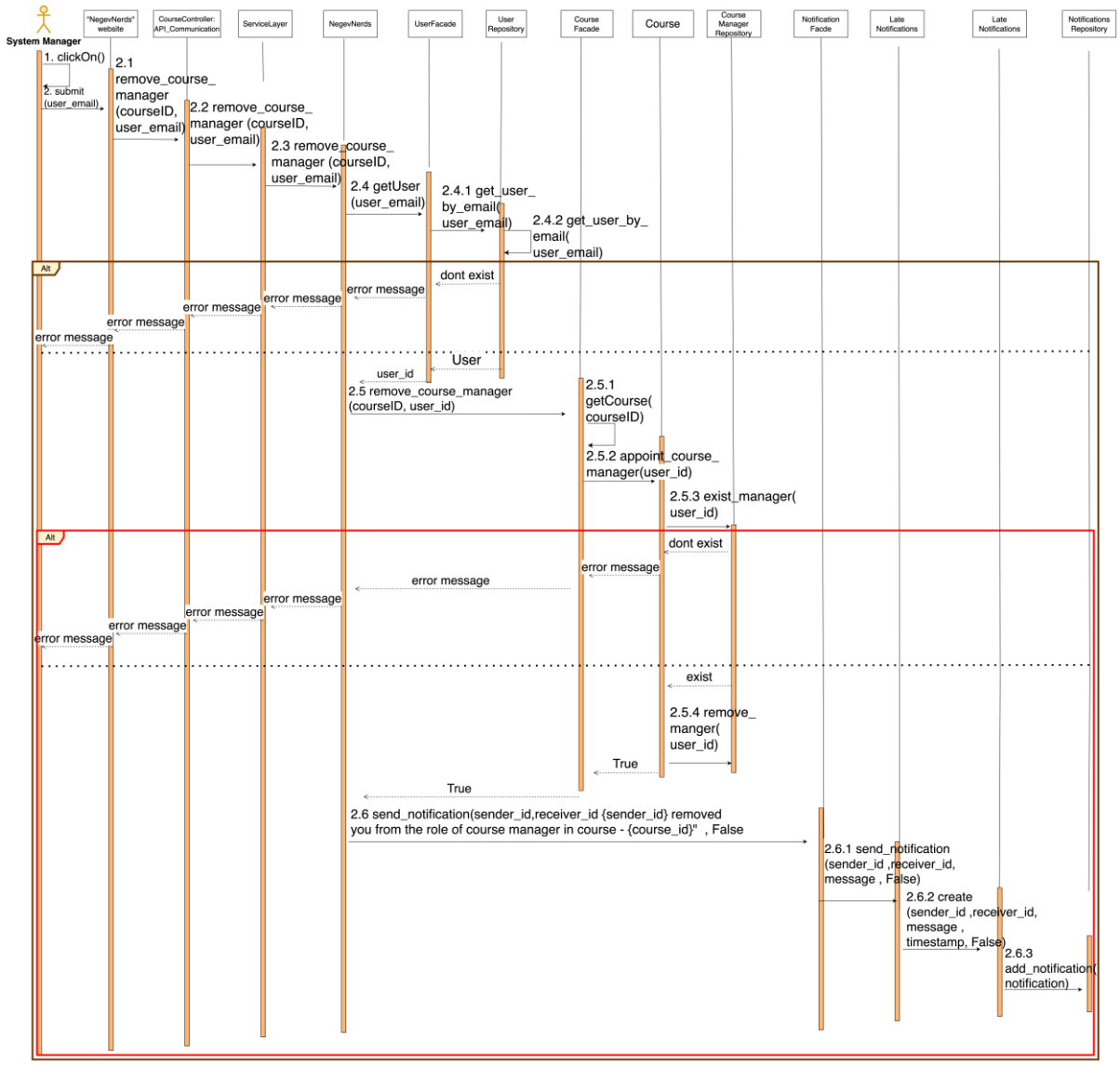



*Figure 25: Sequence Diagram of Remove a Course Manager*

### 4.1.21 Download a ZIP of All Exams of a Course

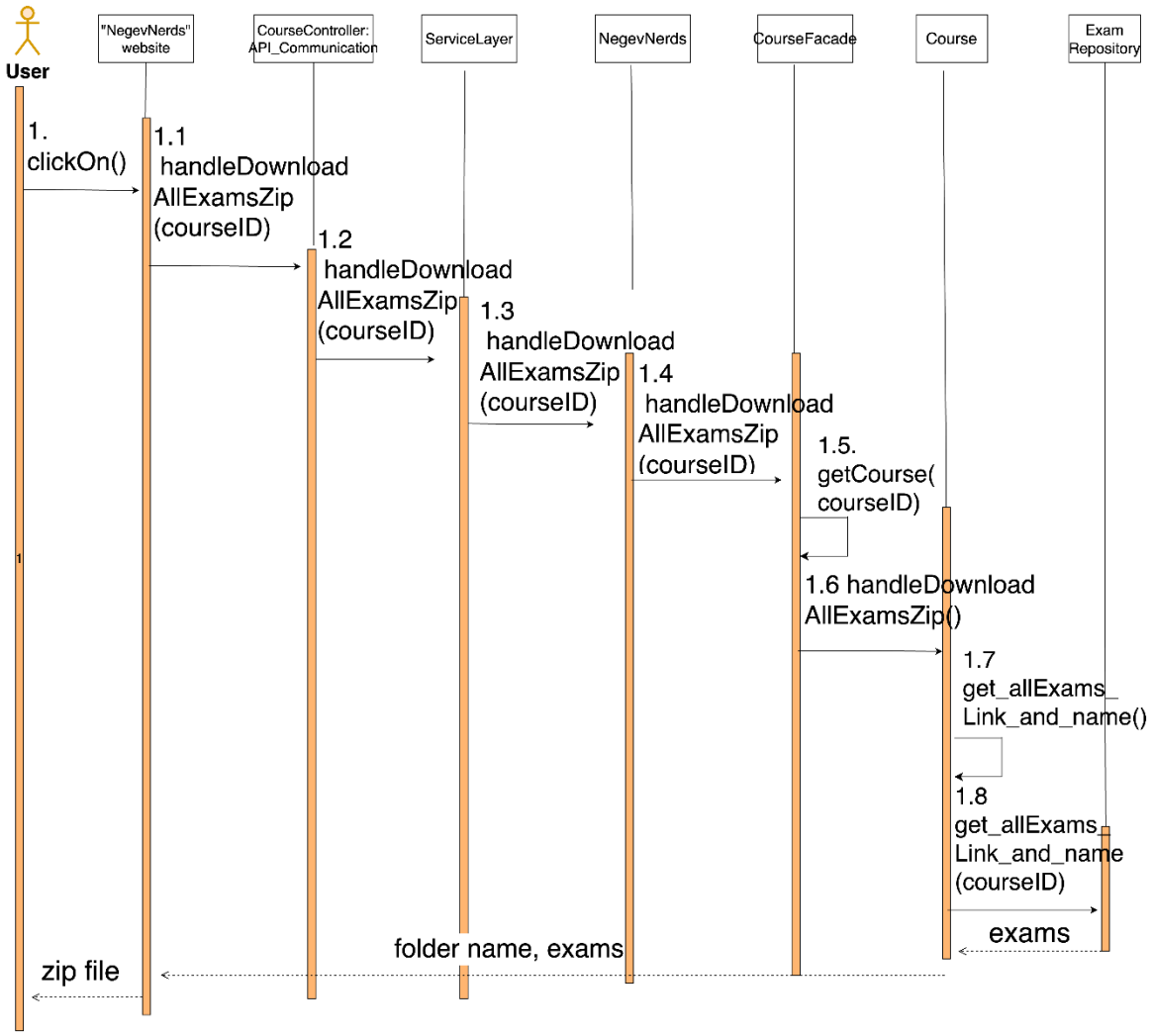



*Figure 26: Sequence Diagram of Download a ZIP of All Exams of a Course*

## b. Events

| Event | Description | Action |
|---|---|---|
| **Register a New User** | A guest registers for the first time to access the platform. | The system validates the provided details. If the email is not a BGU email or is already registered, the system displays an error message. The system sends a verification code to the guest's email.<br>The system validates the verification code provided by the guest. If valid, the system saves the guest's details in the database and registers the guest as a user.<br>The system redirects the user to the homepage. |
| **Login** | A user logs into the platform to access its features. | The system validates the provided credentials. If invalid, the system displays an error message. If valid, the system creates a session for the user and marks them as logged in.<br>The system redirects the user to the homepage. |
| **Register to a Course** | A user registers for a specific course to access its content. | The system checks if the user is already registered for the course. If the user is already registered, the system displays an error message. If the course doesn't exist, the system displays an error message and suggests creating the course. If valid, the system adds the user to the course's participant list. The system displays a success message and updates the user's homepage to show the course |
| **Open a Course** | A user creates a new course on the platform. | The system verifies the provided course details.<br>If the course already exists, the system displays an error message. If the course is invalid (not a BGU course), the system displays an error message.<br>If valid, the system creates the course and adds it |

| | | |
|---|---|---|
| | | to the platform. The system automatically appoints the user as the Course Manager and registers them for the course. The system makes the course visible to students. |
| **Post an Exam** | A user uploads an exam to the platform. | The system validates the uploaded file format. If the file format is invalid, the system displays an error message. If the course associated with the exam doesn't exist, the system displays an error message and suggests creating the course. The system extracts exam details from the file. If the exam already exists, the system displays an error message. If the details are valid, the system stores the exam in the database and displays it on the course page. |
| **Post an Exam's Question** | A user uploads a specific question from an exam. | The system validates the provided question details. If the course doesn't exist, the system displays an error message and suggests creating the course. If the question already exists, the system displays an error message. If valid, the system stores the question in the database and makes it visible on the course or exam page. The system confirms that the question has been successfully posted. |
| **Search Questions by Text** | A user searches for questions containing specific text. | The system processes the user's search query. The system searches the database for matching questions. If no matches are found, the system displays a message. If matches are found, the system displays a list of relevant questions for the user. |
| **Search Questions** | A user searches for questions | The system displays the search bar with a "Search by Topic" option. The system accepts the user's |

| | | |
|---|---|---|
| **by Topic** | that match specific topics. | input for topics. The system processes the topics and queries the database for matching questions. If matches are found, the system displays a list of relevant questions. If no matches are found, the system displays a message. |
| **Search a Question by Specifics** | A user searches for a specific question by entering detailed filters like course name, year, and moed. | The system displays the search bar with a "Search by Specifics" option. The system accepts the user's input for specific details. The system processes the details and queries the database for a matching question. If a match is found, the system redirects the user to the question homepage. If no match is found, the system displays a message. |
| **Comment on a Question** | A user adds a comment on a specific question. | The system accepts the user's comment and optional file upload (PDF or photo). The system validates the comment and uploaded file. If valid, the system saves the comment in the database and associates it with the question. The system notifies all users who have previously commented on the question.<br>The system confirms the comment has been successfully added and displays it under the question.<br>If the comment is empty or violates platform policies, the system displays an error message. |
| **Comment on a Comment** | A user replies to another user's comment in a discussion. | The system displays a "Reply" button under each comment. The system accepts the user's reply and optional file upload (PDF or photo). The system validates the reply and uploaded file.<br>If valid, the system saves the reply in the database |

| | | |
|---|---|---|
| | | and associates it with the parent comment. The system notifies the user who wrote the original comment.<br>The system confirms the reply has been successfully added and displays it under the original comment.<br>If the reply is empty or violates platform policies, the system displays an error message. |
| **Edit a Comment** | A user edits their existing comment to correct or update its content. | The system accepts the user's updates. The system validates the updates (e.g., file format, comment length). If valid, the system updates the comment in the database. The system confirms the update and displays the modified comment under the question. If the updated comment is empty or violates platform policies, the system displays an error message. |
| **Download Exam Full PDF File** | A user downloads the full PDF file of an exam for offline access or review. | When clicked, the system retrieves the PDF file from the database or server. The system initiates the download and saves the file on the user's device. If the file is missing or corrupted, the system displays an error message. |
| **Upload Exam Full PDF File** | A user uploads a full exam PDF file to the system. | The system accepts the uploaded PDF file from the user. The system validates the file format.<br>If valid, the system uploads the file to the server and associates it with the specified exam entry.<br>The system confirms the upload and makes the file available for download. If the file format is invalid, the system displays an error message. |

| | | |
|---|---|---|
| **React to a Comment with an Emoji** | A user reacts to a comment with an emoji. | The system shows a list of emojis for the user to choose from. The system saves the selected emoji as a reaction in the database. The system displays the emoji reaction under the comment and updates the total count for that emoji. The system notifies the comment's writer about the reaction. |
| **Delete a Question** | A Course Manager deletes a question from the system. | The system prompts the Course Manager to confirm the deletion. Upon confirmation, the system removes the question from the database. The system removes all associated comments and reactions. The system confirms the deletion and updates the question list on the website. If the deletion is cancelled, the system takes no action. |
| **Edit a Question** | A Course Manager modifies the details of an existing question to ensure the information is accurate. | The Course Manager updates one or more of the following fields. The system validates the updated fields. If valid, the system updates the question's details in the database. The system confirms the successful update and displays the revised question to users. If the updated details conflict with another question, the system displays an error message. |
| **Delete a Comment** | A Course Manager deletes an existing comment and removes its associated reactions. | The system prompts the Course Manager for confirmation before deletion. Upon confirmation: The system marks the comment as deleted in the database. The system removes all reactions linked to the comment. The system replaces the comment text with a placeholder. The system confirms the successful deletion and updates the UI for all users. |

| | | |
|---|---|---|
| **Appoint a User as a Course Manager** | A Course Manager assigns another user as a Course Manager for a specific course. | The system validates that the selected user and course exist. The system sends a notification to the user:<br>If the user accepts: The system updates the user's role in the database, granting Course Manager permissions for the specified course. The system confirms the update with a message to both the original and newly appointed Course Managers.<br>If the user declines: The system cancels the appointment and notifies the original Course Manager.<br>If the user is already a Course Manager, the system displays an error message. |
| **Remove a Course Manager** | A System Manager removes a user's Course Manager role for a specific course. | The system validates that the user is currently a Course Manager for the course. The system displays a confirmation prompt: Upon confirmation:<br>The system revokes the user's Course Manager permissions for the specified course in the database.<br>The system sends a notification to the user about the removal. The system confirms the successful removal with a message to the System Manager.<br>If the selected user is not a Course Manager for the course, the system displays an error message. |

*Table: 1 Events*

## c. States

A user in NegevNerds has the following States:

1. **Guest**

   A user who hasn’t registered or logged into the platform.

2. **Registered User**

   A user who has completed the registration process.

3. **Logged In**

   A user who has logged into the platform and can access its features.

4. **Course Participant**

   A user who has registered for a course.

5. **Course Manager**

   A user with Course Manager permissions for specific courses.

6. **System Manager**

   A user with administrative privileges across the platform.

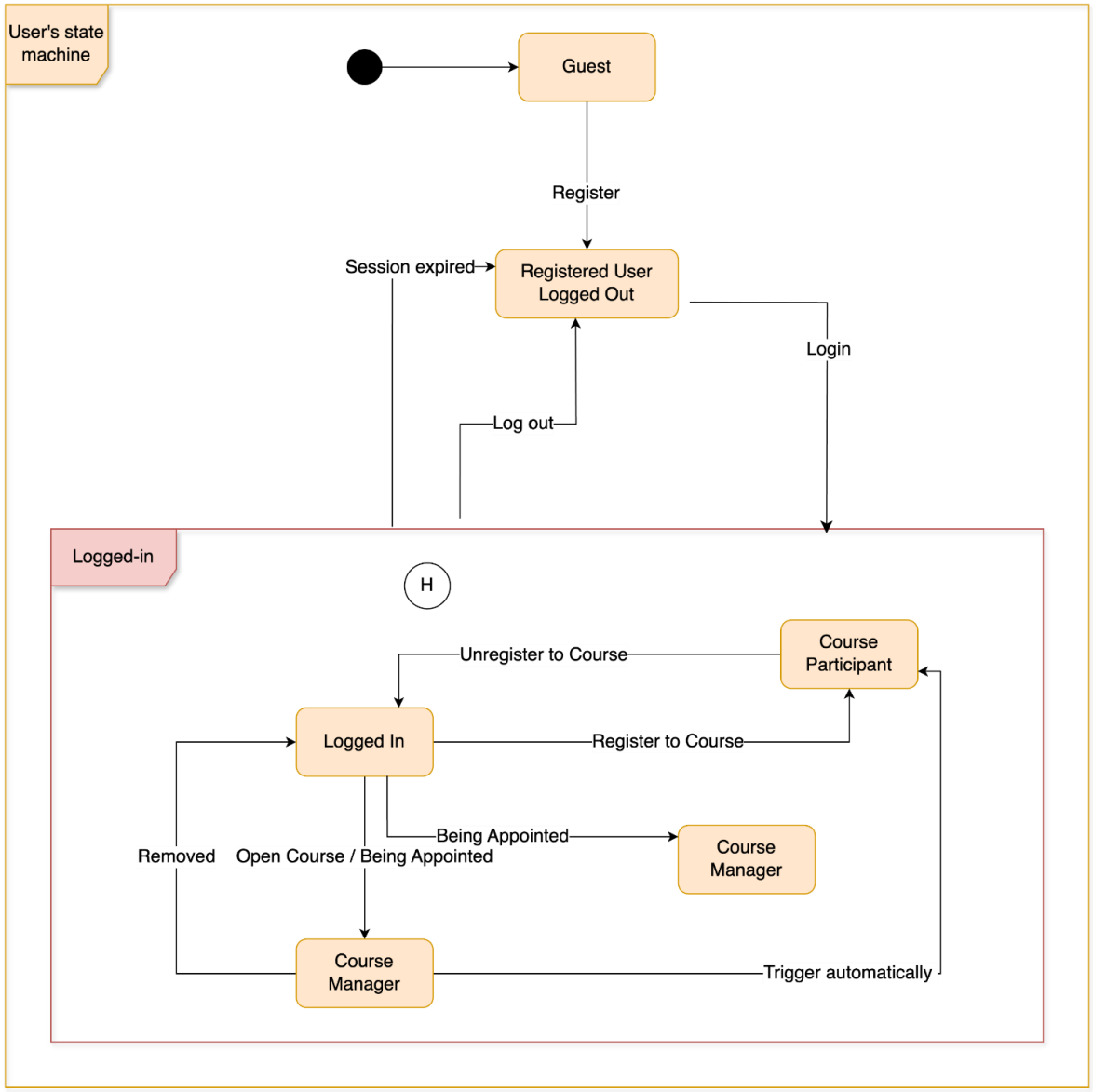



*Figure 27: User's state machine – describes the user states in NegevNerds*

# 5. Object-Oriented Analysis

## a. Class Diagram

(Each component will be displayed on Zoom In later)

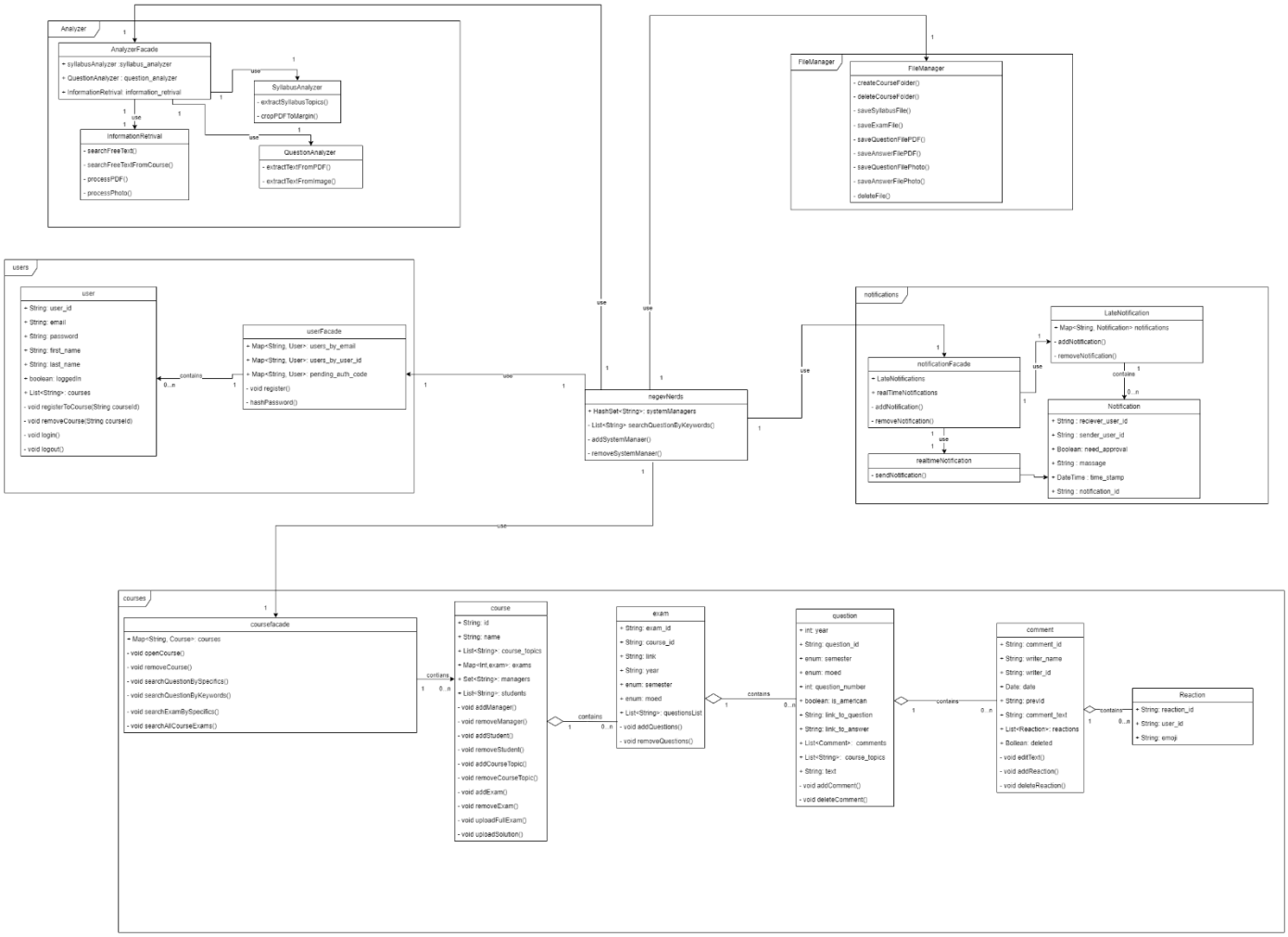



*Figure 28: Class Diagram*

## b. Class Description

### 1. Course Component-

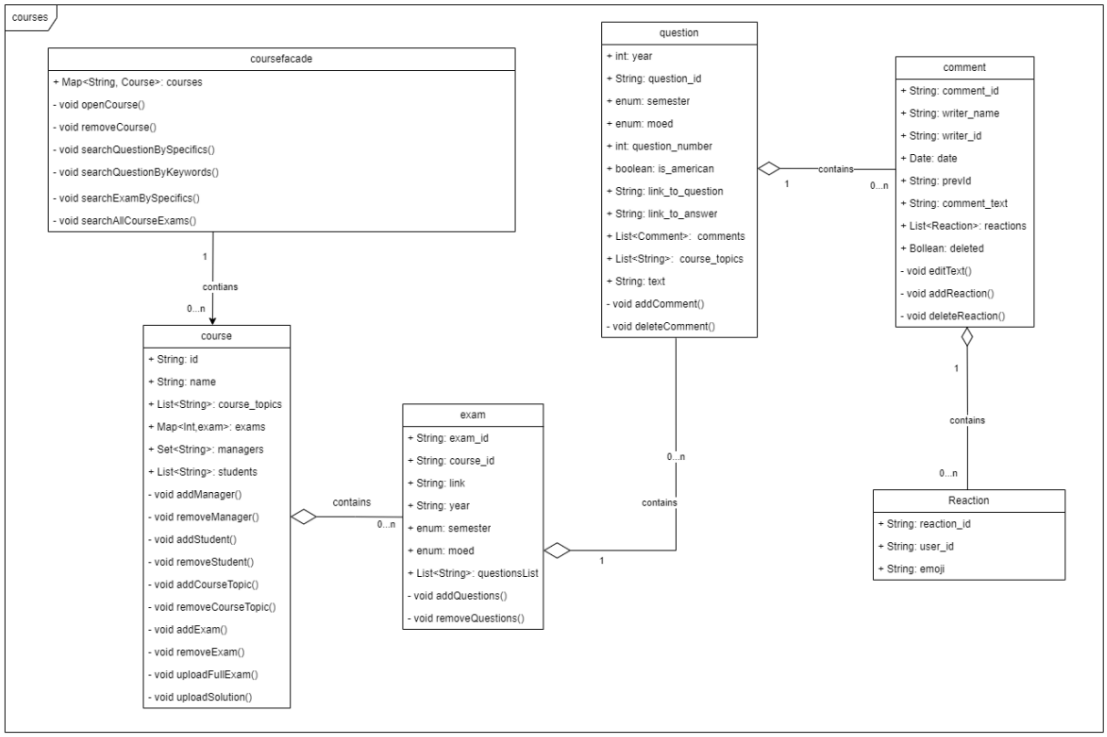



*Figure 29: Class Diagram for Course Component*

#### 1.1. CourseFacade Class

**Responsibilities:** The CourseFacade class serves as a central interface for managing courses, exams, and related operations within the system. This class provides methods to open courses, remove courses, and search for questions or exams by different criteria (specifics, topic or keywords).

**Attributes and invariant**

- **self.courses:** A dictionary where the key is the course_id and the value is a Course object. This represents the available courses in the system.

**Main Methods:**

1. **openCourse()**
    - **Responsibilities**: This method is responsible for opening or activating a course in the system.

- **Pre-conditions**: The course does not exist in the system.
- **Post-conditions**: The course created and available for use by users.
- **Implementation hints:** appoint the user who open the course, as the course manager.

2. **removeCourse()**
    - **Responsibilities**: This method allows to remove a course from the system.
    - **Pre-conditions**: The course must be in the system.
    - **Post-conditions**: The course is deleted from the system and is no longer available for users.
    - **Implementation hints:** remove course from user courses.
3. **searchQuestionBySpecifics()**
    - **Responsibilities**: This method allows users to search for questions based on specific parameters (e.g. semester, year, moed , question number).
    - **Pre-conditions**: None
    - **Post-conditions**: The list of questions matching the paramters is returned.
    - **Implementation hints:** None.
4. **searchQuestionByKeywords()**
    - **Responsibilities**: This method performs a search for questions based on provided keywords.

      **Pre-conditions**: None.
    - **Post-conditions**: The system returns questions that match the given keywords.
    - **Implementation hints:** validate the word is lower-case.
5. **searchExamBySpecifics()**
    - **Responsibilities**: This method allows searching for exams based on specific details, such as the year, semester, moed.
    - **Pre-conditions**: None.

- **Post-conditions**: The system returns exams that meet the provided specifics.
- **Implementation hints:** None.

### 1.2. Course Class

**Responsibilities:**

- Represents a course in the system.
- Manages the list of students currently enrolled in the course.

**Attributes and invariant**

- **self.course_id**: This stores the unique String identifier for the course.
  invariant: self.course_id.matches('\d{3}\.\d{1}\.\d{4}') and Course.allInstances()->isUnique(c | c.course_id)
  implementation hint:
- **self.name**: This stores the name of the course, likely as a string.
  invariant: self.courseName <> ""
- **self.course_topics:** A set that holds all topics related to the course.
- **self.exams:** A dictionary that stores exams, where the key is the year and the value is the corresponding exam.
- **self.managers:** A set that stores the managers of the course, where the manager is identified by a unique manager_id.
- **self.users:** A list to store users_id, which represent students or registerd to the course.

**Methods:**

1. **addStudent()**

   - **Responsibilities**: This method allows adding a student to the course.
   - **Pre-condition:** The student is not already enrolled in the course.
   - **Post-condition**: The student is added to the list of enrolled students.
   - **Implementation hints:** synchronize method with data structures.

2. **removeStudent()**

- **Responsibilities**: This method allows removing a student from the course.
- **Pre-condition:** The student must already be enrolled in the course.
- **Post-condition:** The student is removed from the course's list of enrolled students.
- **Implementation hints:** synchronized method with data structures.

3. **void addCourseTopic()**

- **Responsibilities:** This method allows adding a new topic to the course.
- **Pre-condition:** The topic must not already exist in the course.
- **Post-condition**: The topic is added to the course's list of topics.
- **Implementation hints:** synchronized method with data structures.

4**. void removeCourseTopic()**

- **Responsibilities:** This method allows removing a topic from the course.
- **Pre-condition:** The topic must be part of the course's list of topics.
- **Post-condition:** The topic is removed from the course's list of topics.
- **Implementation hints:** synchronized method with data structures.

**5. void removeExam()**

- **Responsibilities:** This method allows removing an exam from the course.
- **Pre-condition:** The exam must exist in the course's list of exams.
- **Post-condition:** The exam is removed from the course's list of exams.
- **Implementation hints:** synchronized method with data structures.

**6. void addExam()**

- **Responsibilities:** This method allows adding an exam to the course.
- **Pre-condition:** The exam must not already exist in the course.
- **Post-condition:** The exam is added to the course's list of exams.
- **Implementation hints:** synchronized method with data structures

7. **void addManager()**
    - **Responsibilities:** This method allows adding a user as a manager for the course.
    - **Pre-condition:** The user id not in the managers list
    - **Post-condition:** The user is added to the list of course managers.
    - **Implementation hints:** synchronized method with data structures, ensure user enrolled the course.
8. **void removeManager()**
    - **Responsibilities:** This method allows removing a user from the course's manager list.
    - **Pre-condition:** The user must already be a manager for the course.
    - **Post-condition:** The user is removed from the course's list of managers.
    - **Implementation hints:** synchronized method with data structures.

### 1.3. Exam Class

**Responsibilities:**

- Represents an exam within a course.
- Manages a collection of questions related to the exam.

**Attributes and Invariants:**

- **self.course_id:** Stores the unique identifier for the exam.
  *Invariant:* Course.allInstances()->isUnique(c | c.course_id).
- **self.course_id:** Stores the ID of the course to which the exam belongs.
  *Invariant:* self. course_id->forAll(cid|Course.allInstances()->exists(c|c.course_id= cid))
- **self.link:** Stores the link to the exam document.
  *Invariant:* Must be a valid file path.
- **self.year:** Stores the year in which the exam took place.
  *Invariant:* self.year<= Date.Now.year

- **self.semester :** Represents the semester during which the exam was held.
  *Invariant:* Must be a valid value from the Semester enumeration.
- **self.moed :**Represents the exam session.
  *Invariant:* Must be a valid value from the Moed enumeration.
- **self.questions_list**: A dictionary where keys are question numbers, and values are Question objects.

**1. add_question()**

- **Responsibilities:** This method adds a question to the exam's list of questions.
- **Pre-condition:** The question_number must not already exist in the exam's questions list.
- **Post-condition:** The question is added to the exam's list of questions, and the questions_list is updated.
- **Implementation hints:** synchronized method with data structures, be sure it not already exist in different question number.

**2. remove_question()**

- **Responsibilities:** This method removes a question from the exam's list of questions.
- **Pre-condition:** The question_number must exist in the exam's list of questions.
- **Post-condition:** The question is removed from the questions_list
- **Implementation hints:** synchronized method with data structures.

**3. upload_full_exam_pdf()**

- **Responsibilities:** This method uploads the full exam PDF to the course, updating the link to the exam.
- **Pre-condition:** full exam does not exist in the system.
- **Post-condition:** The exam's link is updated**.**
- **Implementation hints:** None.

### 1.4. Comment Class

**Responsibilities:**

- Represents a user comment in the system.
- Handles reactions associated with the comment.

**Attributes and Invariants:**

- **self.comment_id** Unique identifier for the comment.
  *Invariant:* Comment.allInstances()->isUnique(c | c.comment_id)
- **self.writer_name :** Stores the name of the comment's author.
  *Invariant:* User.allInstances()->exists (u | u.name = self.writer_name)
- **self.writer_id :**Unique identifier for the comment's author.
  *Invariant:* User.allInstances()->exists (u | u.user_id = self.writer_id)
- **self.date :** Stores the date and time when the comment was created.
  *Invariant:* self.date<= Date.now
- **self.prev_id :** ID of the previous comment if this comment is a reply.
  *Invariant:* self.prev_id = " " or comment.allInstances()->exists(c | c.comment_id = self.prev_id and c!=self)
- **self.comment_text :** The actual content of the comment.
  *Invariant:* self.comment_text <> " "
- **self.reactions :** Stores all reactions made on the comment.
  Invariant: self.reaction.allInstances()->isUnique(r | r.writer_id)
- **self.deleted:** Indicates whether the comment has been deleted.
- **self.edited:** ndicates whether the comment has been edited.

**Methoods:**

**1. addReaction()**

- **Responsibilities:** This method adds a reaction to a specific comment.

- **Pre-condition:** the comment exist in the system
- **Post-condition:** The reaction is successfully added to the post, if the user already add reaction to this post the previous reaction will be removed.
- **Implementation hints:** synchronized method with data structures, verify if user already have reaction to this post.

**2. deleteReaction():**

- **Responsibilities:** This method deletes a specific reaction from a comment.
- **Pre-condition:** The reaction must already exist for the given comment by the specific user who try to remove it.
- **Post-condition:** The reaction is removed from the comment reactions list.
- **Implementation hints:** synchronized method with data structures.

### 1.5. Reaction Class

**Responsibilities:**

- Represents a user's reaction to a comment in the system.

**Attributes and Invariants:**

- **self.reaction_id**: Unique identifier for the reaction.
  **Invariant**: Reaction.allInstances()->isUnique(r | r.reaction_id)
- **self.user_id**: Unique identifier for the user who added the reaction.
  **Invariant**: User.allInstances()->exists (u | u.user_id = self.user_id)
- **self.emoji**: The String representing the emoji of the reaction.
  **Invariant**: self.emoji <> ""

## 2. User Component -

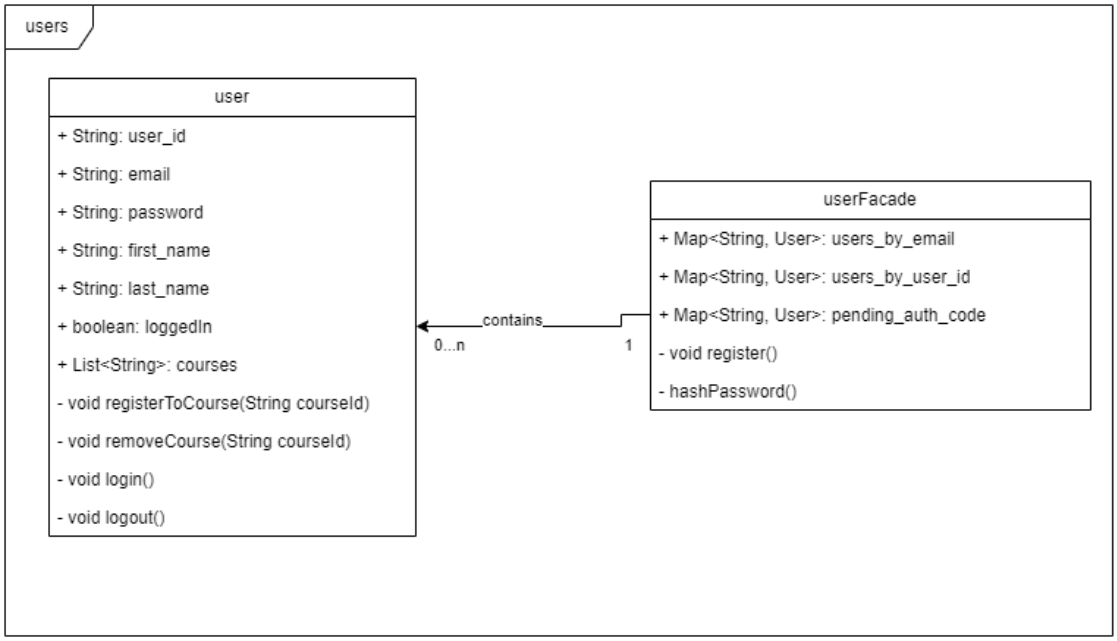



*Figure 30: Class Diagram for User Component*

### 2.1. User Class

**Responsibilities:**

- Represents a user in the system.
- Manages the user's personal information and login status.
- Tracks the courses the user is enrolled in.

**Attributes and Invariants:**

- **self.user_id**: Unique identifier for the user.

  **Invariant**: User.allInstances()->isUnique(u | u.user_id)
- **self.email**: The user's email address.

  **Invariant**: User.allInstances()->isUnique(u | u.email)
- **self.password**: The user's password.

  **Invariant**: self.password.length() >= 8 and self.password.length() <= 20 and self.password.matches('^[a-zA-Z0-9{[!@$%^&*()+=]}]+$') and self.password.matches('.*[A-Z].*') and self.password.matches('.*[a-z].*') and


self.password.matches('.*[0-9].*') and self.password.matches('.*[{[!@$%^&*()+=}].*')

- **self.first_name**: The user's first name.
  **Invariant**: self.first_name.matches('^[\u0590-\u05FF]+([\s\t-][\u0590-\u05FF]+)*$')
- **self.last_name**: The user's last name.
  **Invariant**: self.last_name.matches('^[\u0590-\u05FF]+([\s\t-][\u0590-\u05FF]+)*$')
- **self.loggedIn**: A flag indicating whether the user is logged in.
- **self.courses**: A list of courses the user is enrolled in.
  **Invariant**: self.courses->isUnique(c | c.course_id)

**Methods**

1. **registerToCourse()**
   - **Responsibilities:** This method allows a user to enroll in a specific course.
   - **Pre-condition:** The course must exist in the system, and the user should not already be enrolled in the course.
   - **Post-condition:** the course list for the user is updated to include the newly enrolled course.
   - **Implementation hints:** synchronized method with data structures, verify same registration in course class.
2. **removeCourse()**
   - **Responsibilities:** This method allows a user to remove themselves from a specific course.
   - **Pre-condition:** The user must be enrolled in the course.
   - **Post-condition:** the course list for the user is updated to reflect the removal.
   - **Implementation hints:** synchronized method with data structures, verify same removal in course class.
3. **login ()**
   - **Responsibilities:** This method allows the user to log into the system.
   - **Pre-condition:** The user must be register to the system.

- **Post-condition:** The user's loggedIn status is set to true.
- **Implementation hints:** None.

4. **logout()**
   - **Responsibilities:** This method allows the user to log out of the system.
   - **Pre-condition:** The user must be logged in.
   - **Post-condition:** The user's loggedIn status is set to false, and they are logged out of the system.
   - **Implementation hints:** None.

### 2.2. UserFacade Class

**Responsibilities:**

- Manages user registration, authentication, and related processes.
- Manages pending authentication codes for user verification.

**Attributes and Invariants:**

- **self.users_byEmail**: A dictionary mapping user email addresses to their corresponding User objects.
- **self.users_byId**: A dictionary mapping user IDs to their corresponding User objects.
- **self.pending_auth_codes**: A dictionary mapping authentication codes to user emails for verification purposes.

**Methods**

1. **hashPassword()**
   - **Responsibilities:** This method hashes the user's password to securely store it in the system.
   - **Pre-condition:** The password should be a valid string.

- **Post-condition:** The password is securely hashed, and the hashed password is returned.
- **Implementation hints:** None.

2. **register()**
   - **Responsibilities:** This method registers a new user by creating a User object and storing it in the system.
   - **Pre-condition:** The email must not already be in use.
   - **Post-condition:** A new User object is created with the provided details, and the user is added to both users_byEmail and users_byId dictionaries.
   - **Implementation hints:** verify there are 3 frontend levels for registration.

## 3. Analyzer Component -

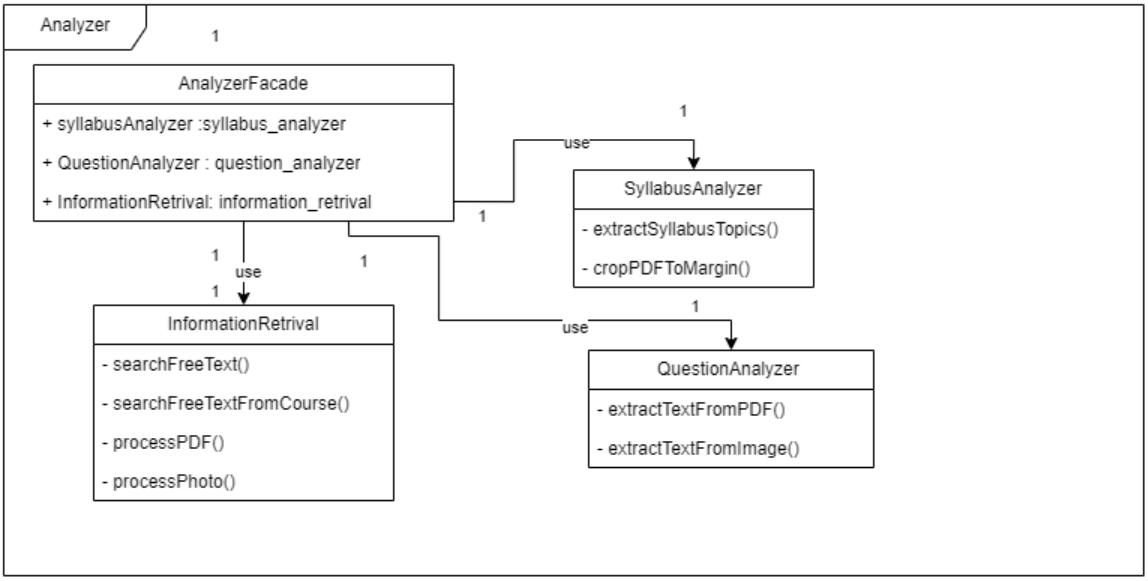



*Figure 31: Class Diagram for Analyzer Component*

### 3.1. AnalyzerFacade Class

**Responsibilities:**

- Provides a unified interface to analyze various aspects of exam-related content.

**Attributes and Invariants:**

- **self.question_analyzer**: An instance of the QuestionAnalyzer
- **self.information_retrival**: An instance of the InformationRetrieval
- **self.syllabus_analyzer**: An instance of the SyllabusAnalyzer

### 3.2. InformationRetrieval Class

**Responsibilities:**

- Handles the retrieval of information from PDFs and images.
- Provides search capabilities for free-text queries within specific courses or the entire system.

**Methods:**

1. **processPDF()**
   - **Responsibilities:** This method processes a given PDF file to extract textual content.
   - **Pre-condition:** the file is PDF file.
   - **Post-condition:** The text content of the PDF is extracted, processed, and stored in the system.
   - **Implementation hints:** use of OCR if needed.
2. **processPhoto()**
   - **Responsibilities:** This method processes an image file to extract text using Optical Character Recognition (OCR).
   - **Pre-condition:** The file is image file.
   - **Post-condition:** The textual content within the image is extracted ,saved and returned for further use.
   - **Implementation hints:** None.
3. **searchFreeTextFromCourse()**
   - **Responsibilities:** This method searches for the given words within the question of the specific course in the system .
   - **Pre-condition:** None.
   - **Post-condition:** Relevant questions list from the specific course returned.
   - **Implementation hints:** move text to lower.

4. **searchFreeText()**
   - **Responsibilities:** This method searches for the given words within the question in the system.
   - **Pre-condition:** None.
   - **Post-condition:** Relevant questions list returned.
   - **Implementation hints:** move text to lower.

### 3.3. QuestionAnalyzer Class

**Responsibilities:**

- Analyzes questions documents to extract textual content.
- Handles text extraction from PDFs and images.

**Methods:**

1. **extractTextFromPDF()**
   - **Responsibilities:** This method extracts text from a given PDF file containing exam question.
   - **Pre-condition:** the file is PDF type.
   - **Post-condition:** The textual content from the PDF is extracted and returned.
   - **Implementation hints:** use of OCR if needed.
2. **extractTextFromImage()**
   - **Responsibilities:** This method extracts text from an image file (e.g., scanned exam question) using Optical Character Recognition (OCR).
   - **Pre-condition:** the file is image File
   - **Post-condition:** The textual content from the image is extracted and returned for further processing.
   - **Implementation hints:** None.

### 3.4. SyllabusAnalyzer Class

**Responsibilities:**

- Handles operations related to analyzing and processing syllabus PDF files, such as extracting topics and adjusting document margins.

**Methods:**

1. **extractSyllabusTopics()**
   - **Responsibilities:** Extracts the syllabus topics from the provided PDF content.

- **Pre-condition**: The input must be a valid PDF file containing syllabus content.
- **Post-condition:** A list of extracted topics is returned.
- **Implementation hints:** None.

2. **cropPDFToMargin()**
   - **Responsibilities:** Crops the provided PDF document to few cropes PDF each file for each question.
   - **Pre-condition:** The input must be a valid PDF file.
   - **Post-condition:** A cropped PDF files is generated and ready for further use.
   - **Implementation hints:** None.

## 4. FileManager Component-

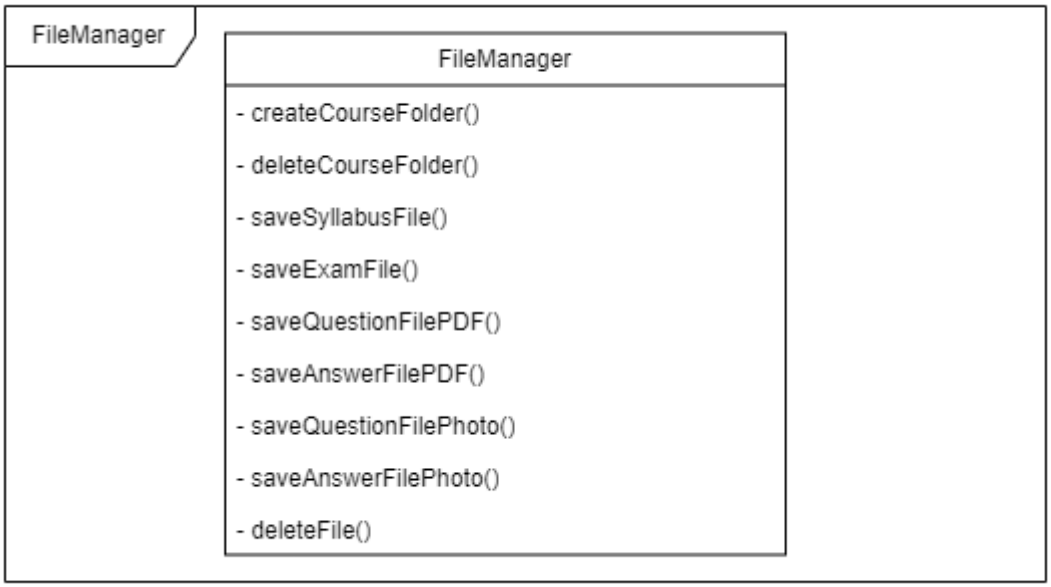



*Figure 32: Class Diagram for FileManager Component*

### 4.1. FileManager Class

**Responsibilities:**

- Manages system file operations including creating, saving, and deleting files and directories.

**Methods:**

1. **deleteCourseFolder()**
   - **Responsibilities:** Deletes the folder associated with the specified course.
   - **Pre-condition:** The course folder must exist.
   - **Post-condition:** The course folder and all its contents are permanently removed.
   - **Implementation hints:** None.
2. **createCourseFolder()**
   - **Responsibilities:** Creates a new folder for storing files related to the specified course.
   - **Pre-condition:** A folder for the given courseId must not already exist.


- **Post-condition:** A new course folder is created and ready for storing course files.
- **Implementation hints:** None.

3. **saveSyllabusFile()**
   - **Responsibilities:** Saves the syllabus file for the specified course.
   - **Pre-condition:** The file must exist and be in an PDF format.
   - **Post-condition:** The syllabus file is stored in the corresponding course folder.
   - **Implementation hints:** None.
4. **saveExamFile()**
   - **Responsibilities:** Saves an exam file for the specified course.
   - **Pre-condition:** The file must exist and contain valid exam content.
   - **Post-condition:** The exam file is stored in the corresponding course folder.
   - **Implementation hints:** None.
5. **saveQuestionFilePDF()**
   - **Responsibilities:** Saves a question file in PDF format for the specified course.
   - **Pre-condition:** The file must be a valid PDF document.
   - **Post-condition:** The question file is stored in the course's designated folder.
   - **Implementation hints:** None.
6. **saveAnswerFilePDF()**
   - **Responsibilities:** Saves an answer file in PDF format for the specified course.
   - **Pre-condition:** The file must be a valid PDF document.
   - **Post-condition:** The answer file is stored in the course's designated folder.
   - **Implementation hints:** None.
7. **saveAnswerFilePhoto()**
   - **Responsibilities:** Saves an answer file in image format for the specified course.
   - **Pre-condition:** The image file must be in a supported format (JPEG, JPG, PNG).

- **Post-condition:** The answer image is stored in the course's designated folder.
- **Implementation hints:** None.

8. **saveQuestionFilePhoto(String courseId, String imagePath)**
   - **Responsibilities:** Saves a question file in image format for the specified course.
   - **Pre-condition:** The image file must in a supported format (JPEG, JPG, PNG).
   - **Post-condition:** The question image is stored in the course's designated folder.
   - **Implementation hints:** None.
9. **deleteFile()**
   - **Responsibilities:** Deletes a specific file from the system.
   - **Pre-condition:** The file must exist in the system.
   - **Post-condition:** The specified file is permanently removed.
   - **Implementation hints:** None

## 5. Notification Component-

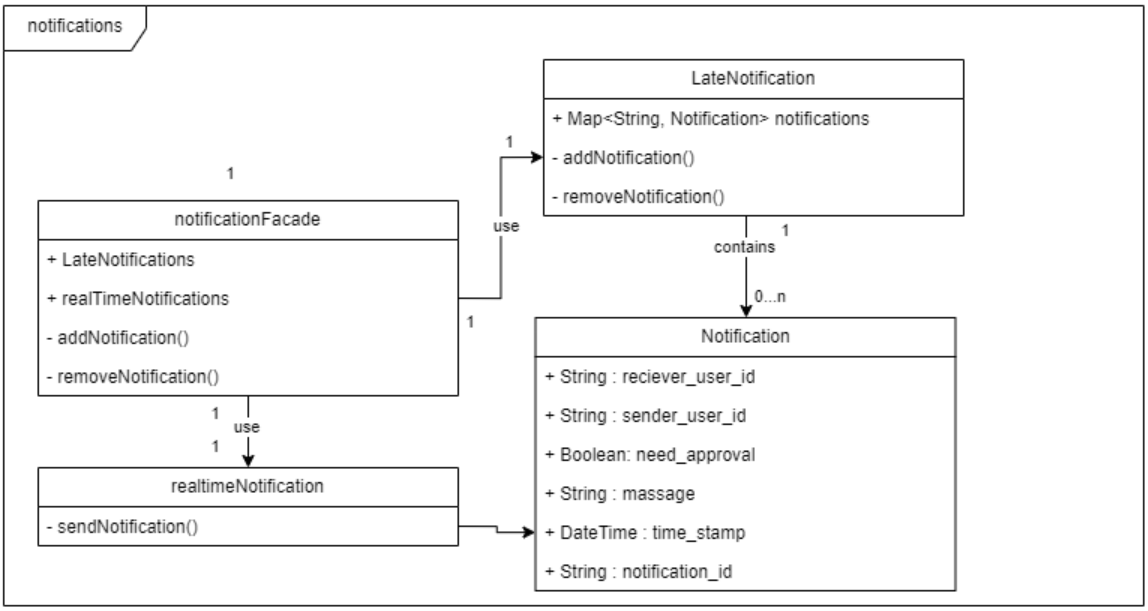



*Figure 33: Class Diagram for Notification Component*

## 5.1. NotificationFacade Class

**Responsibilities:**

- Manages the sending and handling of notifications in the system.

**Attributes and Invariants:**

- **self.late_notifications**: Handles notifications that are sent not in real time.
- **self.real_time_notifications**: Handles instant notifications delivered in real time.

**Methods:**

1. **addNotification()**
   - **Responsibilities:** Adds a new notification to the system, either for immediate delivery or for late.
   - **Pre-condition:** the user_id who is the target of the notification is related to exist user.
   - **Post-condition:** The notification is stored in the appropriate notification system (real-time or late).
   - **Implementation hints:** synchronize with data structures.
2. **removeNotification()**
   - **Responsibilities:** Removes a previously pending notification from the system.
   - **Pre-condition:** The notification must exist in the system.
   - **Post-condition:** The notification is deleted, and the user will no longer receive it.
   - **Implementation hints:** synchronize with data structures.

## 5.2. RealTimeNotification Class

**Responsibilities:**

- Handles the immediate delivery of notifications to users.

**Methods:**

1. **sendNotification()**
    - **Responsibilities:** Sends an instant notification to the specified user.
    - **Pre-condition:** the user_id who is the target of the notification is related to exist user.
    - **Post-condition:** The notification is send to the user in real time.
    - **Implementation hints:** None.

**5.3. LateNotification Class**

**Responsibilities:**

- Manages notifications that cannot be deliver in real time.

**Attributes and Invariants:**

- **self.notifications**: A dictionary of pending notifications by user_id.

**Methods:**

1. **addNotification()**
    - **Responsibilities:** Adds a new notification to the system for late delivery.
    - **Pre-condition:** the user_id who is the target of the notification is related to exist user.
    - **Post-condition:** The notification is stored in the appropriate notification system (real-time or late).
    - **Implementation hints:** synchronize with data structures.
2. **removeNotification()**
    - **Responsibilities:** Removes a previously pending notification from the system.
    - **Pre-condition:** The notification must exist in the system.
    - **Post-condition:** The notification is deleted, and the user will no longer receive it.
    - **Implementation hints:** synchronize with data structures.

### 5.3. Notification Class

**Responsibilities:**

- Represents a notification in the system that can be sent from one user to another.

**Attributes and Invariants:**

- **self.receiver_user_id**: The unique identifier of the user receiving the notification.
  **Invariant:** User.allInstances()->exists(u | u.user_id = self.receiver_user_id)
- **self.sender_user_id**: The unique identifier of the user sending the notification.
  **Invariant:** User.allInstances()->exists(u | u.user_id = self.sender_user_id)
- **self.message**: The content of the notification message.
  **Invariant:** self.message <> ""
- **self.timestamp**: The date and time when the notification was created.
  **Invariant:** self.timestamp <= Date.now
- **self.need_approval**: A boolean flag indicating if the notification requires approval before delivery.
- **self.notification_id**: A unique identifier for the notification.
  **Invariant:** Notification.allInstances()->isUnique(n | n.notification_id)

**Main Component – Negev Nerds**

## 6. NegevNerds Class

**Responsibilities:**

- Manages the core functionality of the NegevNerds system, connects all the system components.

**Attributes and Invariants:**

- **self.systemManagers**: A set of unique identifiers for the system managers.
  **Invariant:** self.systemManagers->isUnique(user_id)

**Methods:**

1. **searchQuestionByKeywords()**
   - **Responsibilities:** Searches for questions within the system that match the provided keywords.
   - **Pre-condition:** None
   - **Post-condition:** A list of questions that match the given keywords is returned.
   - **Implementation hints:** None.
2. **addSystemManager()**
   - **Responsibilities:** Adds a user as a system manager, granting them administrative privileges.
   - **Pre-condition:** The manager user_id must correspond to an existing user in the system, the user id don't exist in the system.
   - **Post-condition:** The user with the given managerId is added to the systemManagers set.
   - **Implementation hints:** synchronize with data structures.
3. **removeSystemManager()**
   - **Responsibilities:** Removes a system manager from the list of authorized system managers.
   - **Pre-condition:** The manager user_Id must exist in the systemManagers set.
   - **Post-condition:** The user with the given managerId is removed from the systemManagers set.
   - **Implementation hints:** synchronize with data structures.

## c. Packages

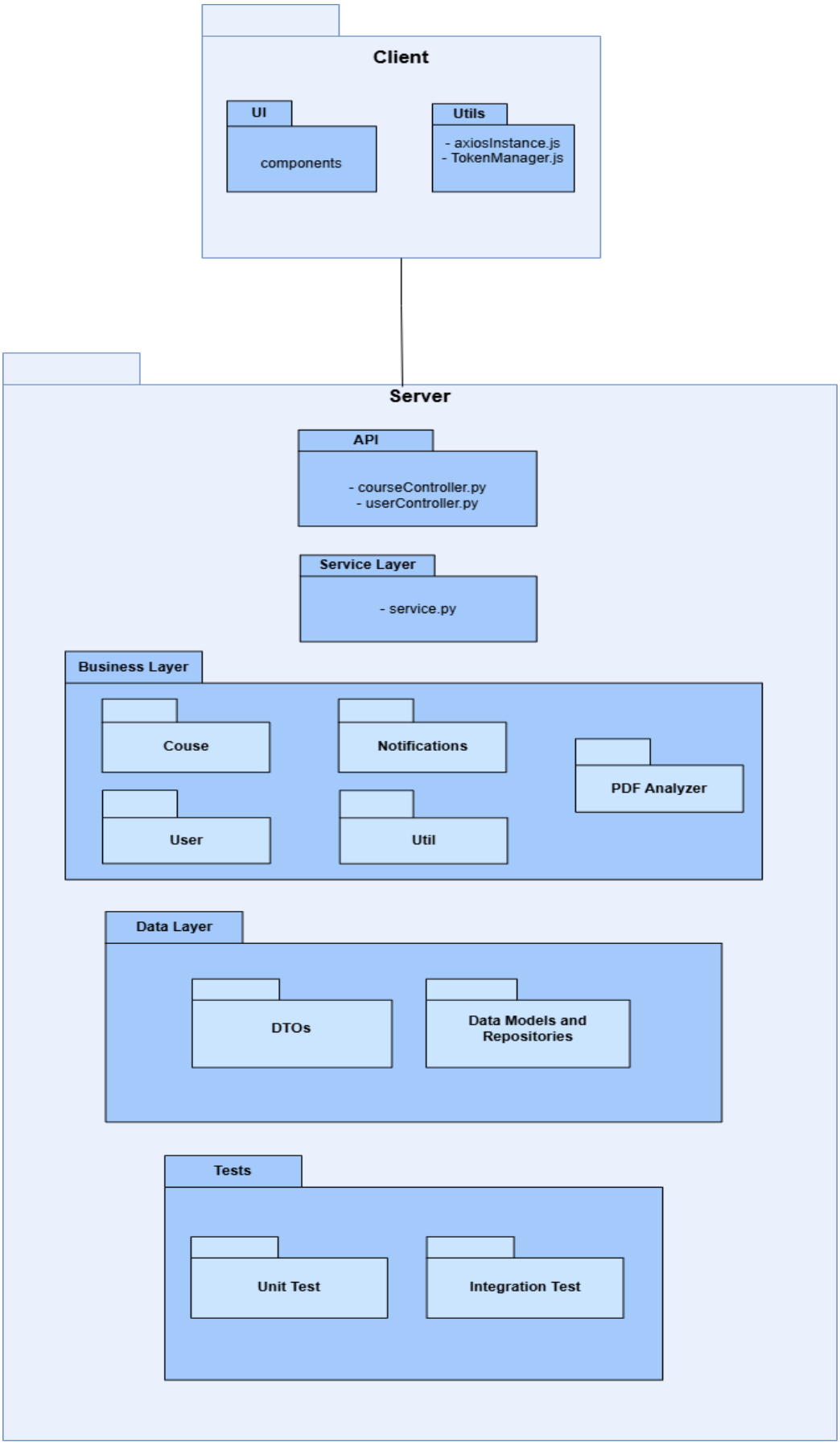



*Figure 34: Packages*

## i. Client

1. **UI Package:**

   **Purpose:** This package serves as the presentation layer of the application. It contains the logic responsible for rendering the user interface and displaying the various components of the app.

   **Content:** React components that represent all screens on the website.

2. **Utils Package:**

   **Purpose:** This package contains utility functions that are used across the entire application.

   **Content:**

   - axiosInstance.js – A configuration file for the Axios HTTP client, setting up parameters and desired behaviours for API requests.
   - tokenManager.js – Handles the generation, storage, and retrieval of authentication tokens used for securing API calls.

## ii. Server

1. **API Layer:**

   **Purpose:** The API layer is responsible on communication between the frontend and backend. It handles operations that allow data to be created, retrieved, and updated, ensuring smooth interaction between the client and the server.

   **Content:** The courseController.py and userController.py files are responsible for managing the application's requests.

2. **Service Layer:**

   **Purpose:** The service layer acts as a bridge between the application’s business logic and the API functions.

   **Content:** The service.py file serves as the main service component, facilitating communication between the business layer and the API. It processes incoming


requests, coordinates with the business logic, and returns the relevant responses to the client.

3. **Business Layer:**

   **Purpose:** The business layer encompasses the application's domain logic. It is responsible for managing essential functionality, such as handling user interactions, course management, notifications, and user management.

   **Content:**

   - **Course/** – Manages business logic related to courses, such as course creation and overall course management.
   - **Notifications/** – Handles logic for sending notifications to users, for example, when a new course or exam is available.
   - **PDF Analyzer/** – Processes PDF files to extract relevant question data or analyze document content, such as extracting course topics from syllabus or splitting exams into individual answers.
   - **User/** – Manages user-related logic, such as user registration, authentication, and profile management.
   - **Util/** – General utility functions used across the business layer and includes system exceptions.

4. **Data Layer:**

   **Purpose:** The data layer handles the interaction with the database. It performs operations like storing, retrieving and updating data.

   **Content:** This layer contains models and repositories for each database table. The model defines the structure of the data, while the repository manages the operations needed to retrieve and store the data.

5. **Tests:**

   **Purpose:** The tests package is responsible for ensuring that all components of the backend work as expected. It includes unit and integration tests to validate the functionality and behavior of the system.

   **Content:**

- **Unit Test/** – Contains unit tests for individual components.
- **Acceptance Test/** – Includes tests that validate the interaction between different components of the system.

## d. Unit Testing

### i. Course Package

| ID | Function | Test Description | Expected Result |
|---|---|---|---|
| 1 | register_to_course (course_id, user_id) | Valid parameters | The user is successfully added to the course's student list. |
| 2 | | Invalid course_id | An error message is displayed. |
| 3 | | Student already registered | An error message is displayed. |
| 4 | remove_student_from_course (course_id, user_id) | Valid parameters | The user is successfully removed from the course's student list. |
| 5 | | User not enrolled in the course | An error message is displayed. |
| 6 | open_course (course_id, name, course_topics) | Valid parameters | A new course has been successfully added to the courses list. |

| | | | |
|---|---|---|---|
| 7 | | Course ID already exists | An error message is displayed. |
| 8 | | invalid course ID | An error message is displayed. |
| 9 | | invalid course name | An error message is displayed. |
| 10 | Remove_course (course_id) | Valid course_id | The course is successfully removed from the courses list. |
| 11 | | Course ID does not exist | An error message is displayed. |
| 12 | get_course (course_id) | Valid course ID | Returns the course instance. |
| 13 | | Invalid course ID | Returns None. |
| 14 | add_course_topic (course_id, course_topic) | Valid parameters | The course topic is successfully added to the course's topics list. |
| 15 | | Topic already exists | An error message is displayed. |
| 16 | remove_course_topic (course_id, course_topic) | Valid parameters | The course topic is removed successfully to the topics set. |
| 17 | | Topic not found | An error message is displayed. |
| 18 | set_syllabus_of_course (course_id, syllabus) | Valid parameters | The course topic is successfully removed from the course's topics list. |
| 19 | | Invalid syllabus | An error message is displayed. |
| 20 | | Invalid course_id | An error message is displayed. |

| | | | |
|---|---|---|---|
| 21 | add_manager_to_course (course_id, manager_id) | Valid parameters | The manager is successfully added to the course's manager list. |
| 22 | | Manager already exists | An error message is displayed. |
| 23 | | Invalid course ID | An error message is displayed. |
| 24 | remove_manager_from_course (course_id, manager_id) | Valid parameters | The manager is successfully removed from the course's manager list. |
| 25 | | Invalid course ID | An error message is displayed. |
| 26 | | Manager not found | An error message is displayed. |
| 27 | is_course_manager (course_id, user_id) | Valid course ID and manager user ID | Returns true. |
| 28 | | Valid course ID and non-manager user ID | Returns false. |
| 29 | add_exam (course_id, course_name, link, year, semester, moed) | Valid parameters | The exam is added to the course. |
| 30 | | Duplicate exam details (same year, semester, moed) | An error message is displayed. |
| 31 | remove_exam (course_id, year, semester, moed) | Valid parameters | The exam is removed from the course. |

| | | | |
|---|---|---|---|
| 32 | | Exam not found | An error message is displayed. |
| 33 | add_question (question_number, is_american, question_topics, pdf_question_path, pdf_answer_path, question_text) | Valid question details | The question is added successfully to the exam. |
| 34 | | Invalid one or more parameters | An error message is displayed. |
| 35 | remove_question (question_number) | Valid question number | The question is removed from the exam. |
| 36 | | Question number does not exist | An error message is displayed. |
| 37 | uploadSolution (course_id, year, semester, moed, question_number, answer_path_path) | Valid parameters | The solution is successfully uploaded, and the answer path is updated in the system. |
| 38 | | Invalid one or more parameters | An error message is displayed. |
| 39 | add_reaction (user_id, emoji) | Valid parameters | A reaction is successfully added to the comment. |
| 40 | | Invalid emoji | An error message. |
| 41 | | User already reacted with the same emoji | An error message is displayed. |
| 42 | remove_reaction (reaction_id) | Valid reaction_id | The reaction is successfully removed from the comment. |
| 43 | | Invalid reaction_id | An error message is displayed. |
| 44 | edit_text (new_text) | Valid text | The comment's text is |

| | | | |
|---|---|---|---|
| | | | updated. |
| 45 | | Empty text | An error message is displayed. |
| 46 | add_comment (writer_name, writer_id, prev_id, comment_text, deleted) | Valid comment details | The comment is successfully added to the question. |
| 47 | | Invalid comment data | An error message is displayed. |
| 48 | delete_comment (comment_id) | Valid comment ID | The comment is successfully deleted. |
| 49 | | Comment not found | An error message is displayed. |

*Table 2: Unit Testing for Course Package*

## ii. Notification Package

| ID | Function | Test Description | Expected Result |
|---|---|---|---|
| 1 | delete_user_notifications (user_id) | Valid user_id | The user's notifications are removed. |
| 2 | | Invalid user_id | An error message is displayed. |
| 3 | get_user_notifications(user_id) | Valid user_id | Returns the list of notifications of the user |
| 4 | | Invalid user_id | An error message is displayed. |
| 5 | send_notification (sender_id, receiver_id, message, need_approval) | Valid parameters | The notification added for the receiver |
| 6 | | Invalid sender ID or receiver ID | An error message is displayed. |
| 7 | | Missing or invalid message | An error message is displayed. |

*Table 3: Unit Testing for Notification Package*

### iii. Analyzer Package

| ID | Function | Test Description | Expected Result |
|---|---|---|---|
| 1 | extract_syllabus_topic_total (pdf_path) | Valid PDF path | The syllabus topics are successfully extracted from the PDF. |
| 2 | | Invalid PDF path | An error message is displayed. |
| 3 | perform_information_retrival_question_pdf (pdf_question_path, question_id, course_id) | Valid parameters | The PDF is processed successfully, and the information retrieval is performed for the question. |
| 4 | | Invalid PDF question path | An error message is displayed. |
| 5 | perform_information_retrival_question_photo (text, question_id, course_id) | Valid parameters | The photo is processed successfully, and information retrieval is performed. |
| 6 | | Invalid text or parameters | An error message is displayed. |
| 7 | search_free_text (text) | Valid text | The free text search is successfully performed, and relevant results are returned. |
| 8 | | Invalid text | An error message is displayed. |
| 9 | search_free_text_from_course (text, course_id) | Valid parameters | The free text search is performed within the specified course, and relevant results are returned. |
| 10 | | Invalid one or more parameters | An error message is displayed. |

*Table 4: Unit Testing for Analyzer Package*

## iv. User Package

| ID | Function | Test Description | Expected Result |
|---|---|---|---|
| 1 | register (email, password, password_confirm, first_name, last_name) | Valid parameters | A user is successfully registered, and an authentication code is sent. |
| 2 | | Invalid one or more parameters | An error message is displayed. |
| 3 | register_authentication_part (email, auth_code) | Correct auth code | Proceeds to the next part of the registration process. |
| 4 | | Expired auth code or incorrect auth code | An error message is displayed. |
| 5 | register_termOfUse_part(email , password, first_name, last_name) | Valid parameters | Registers the user and returns a success message. |
| 6 | | Invalid one or more parameters | An error message is displayed. |
| 7 | login (email, password) | Valid parameters | Successfully logs in and returns user info with a success message. |
| 8 | | Invalid one or more parameters | An error message is displayed. |
| 9 | Logout (user_id) | Valid user_id | Successfully logs the user out and returns a success message. |
| 10 | | Invalid user ID | An error message is displayed. |

| | | | |
|---|---|---|---|
| 11 | | User not logged in | An error message is displayed. |
| 12 | registerToCourse (courseId, userId) | Valid course ID and user ID | Successfully registers the user for the course. |
| 13 | | Invalid course ID | An error message is displayed. |
| 14 | | Invalid user_id | An error message is displayed. |

*Table 5: Unit Testing for User Package*

# 6. User Interface Draft

Sign up:

Step 1 – This screen enables users to create a new account for the website by providing their personal details and choosing a password.

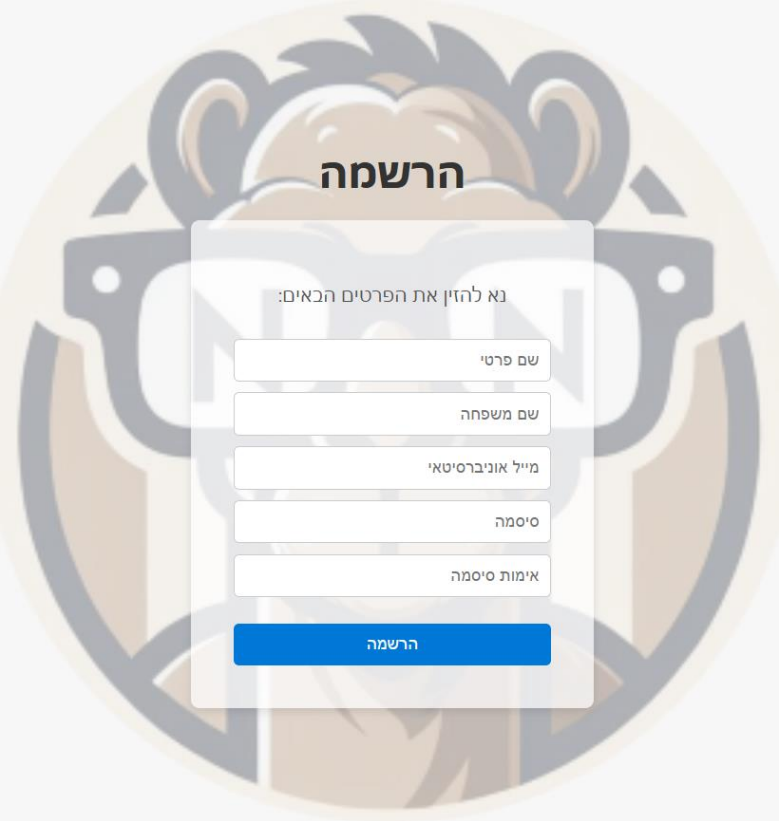



*Figure 35: SignUp – Step1 View*

Step 2 – In this screen, the user is required to enter the one-time code sent to their email to complete the authentication process.

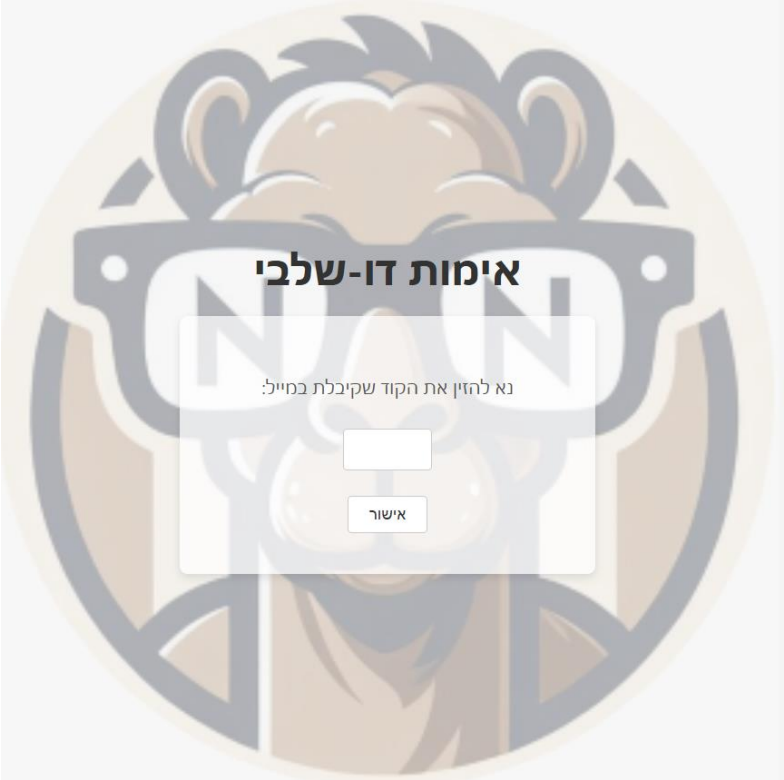



*Figure 36: SignUp - Step 2 View*

Step 3 – In this screen, the user must agree to the website's terms and conditions. Once approved, the registration process is completed, and the system redirects the user to the home page.

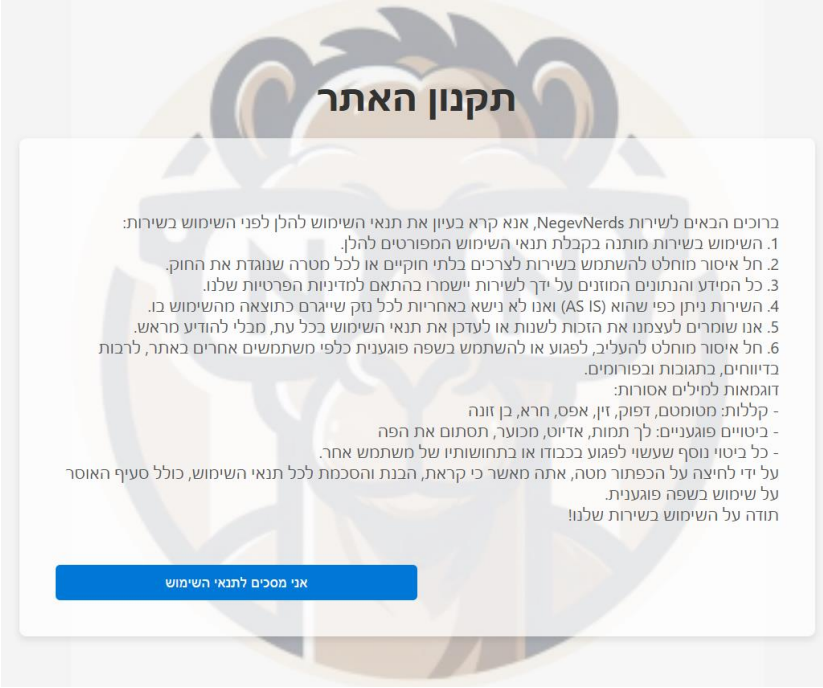



*Figure 37: SignUp - Step 3 View*

Log in:

The login screen allows registered users to access the website by entering their university email and password.

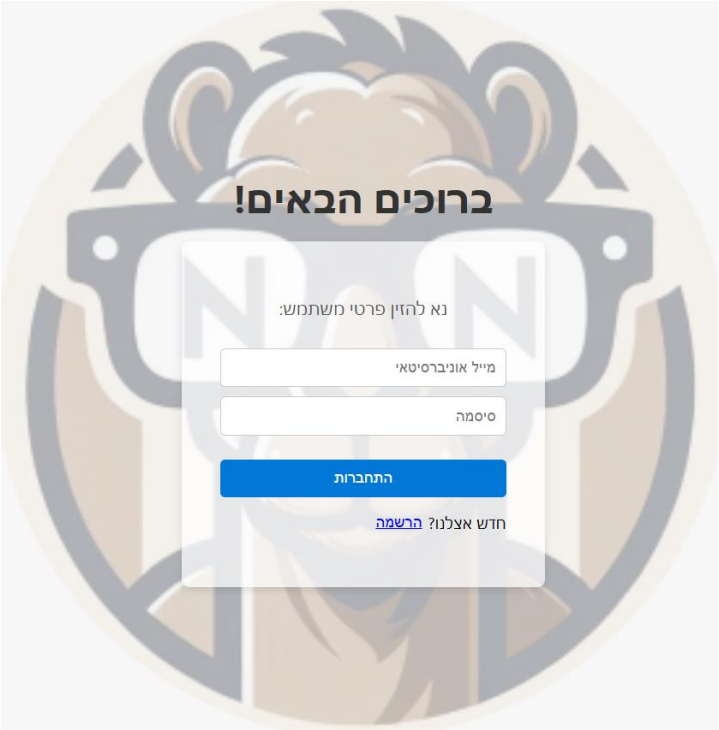



*Figure 38: Login View*

Home:

The home screen is the starting point for logged in users, where they can search for questions based on various criteria, upload exams or questions, access their saved courses, search for courses, and create a new course.

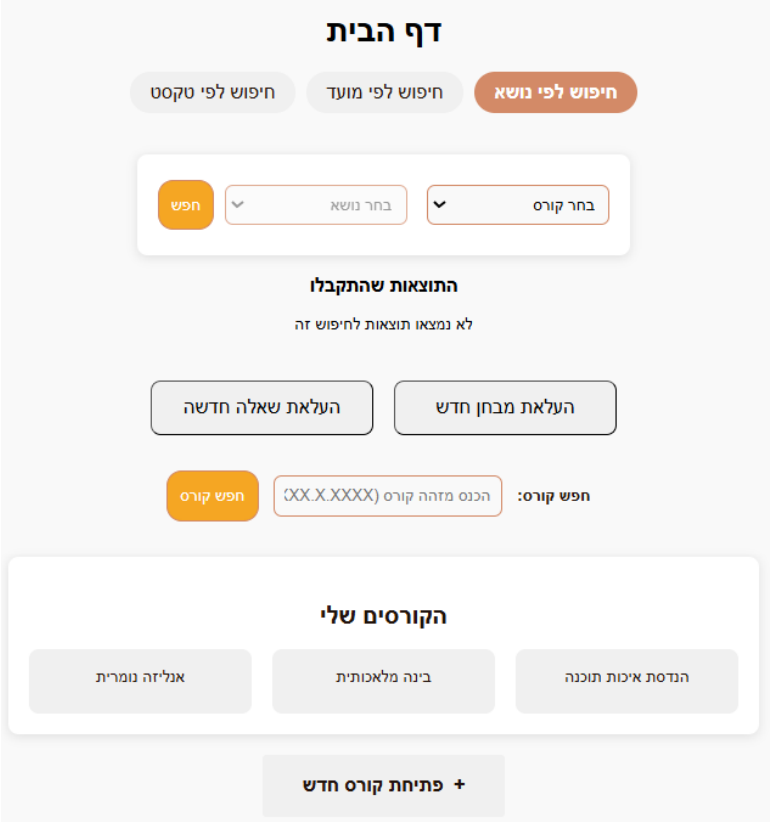



*Figure 39: HomePage View*

Course:

The course page allows users to search for questions within the course based on various criteria, access the course's exams available on the website, upload an exam or a question to this course, and add or remove the course from their list of saved courses.

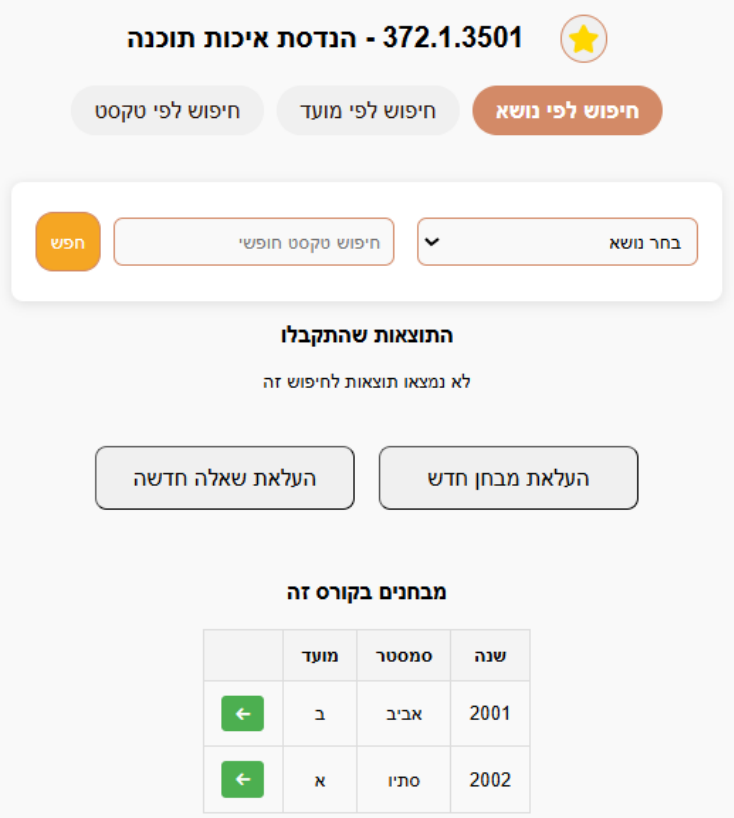



*Figure 40: Course Page View*

Exam:

The exam page displays the exam details and all the questions for which information is available on the website. Users can also upload or download the complete exam (if available) and add a new question to this exam.

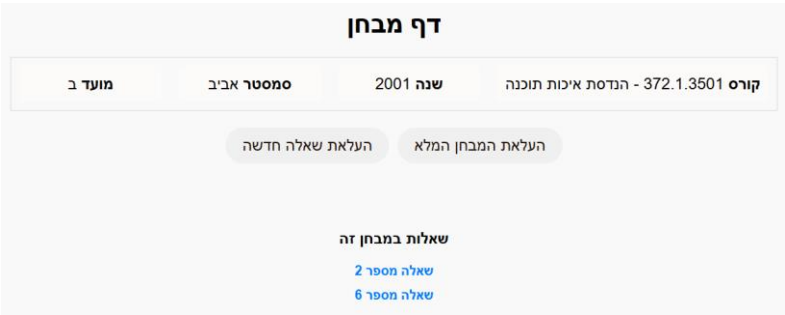



*Figure 41: Exam page View*

Question:

The question page displays the question's details, along with the question and solution as they appear in the exam. Additionally, it features a chat where users can communicate with each other, including support for emojis, nested replies, and the ability to delete and edit comments.

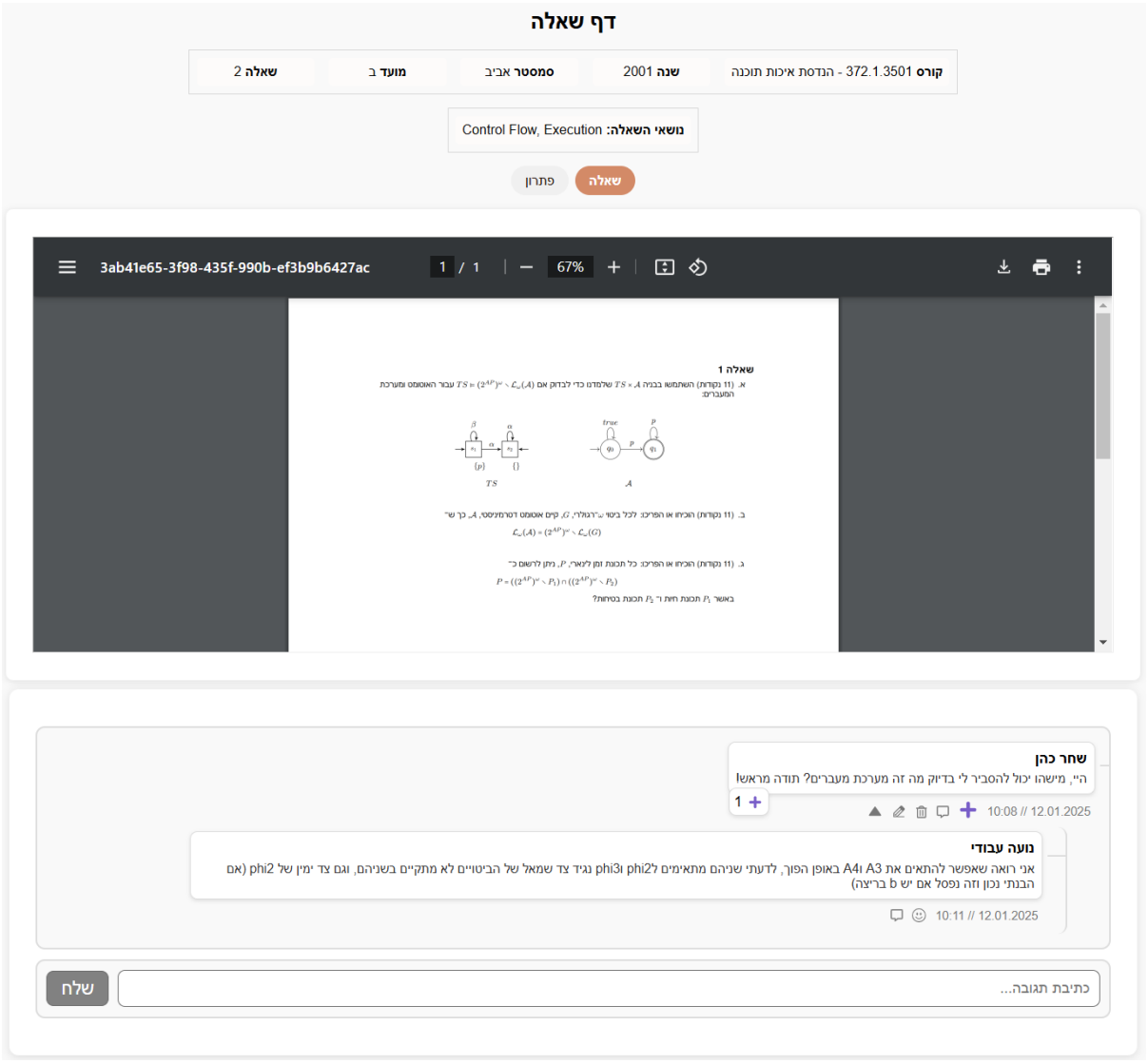



*Figure 42: Question Page View*

Open new course:

In this page, users can create a new course by entering the course number, course name, and its official syllabus.

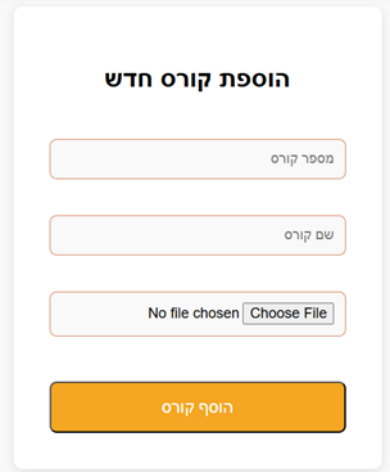



*Figure 43: Open new course view*

Upload question:

Step 1 – In this screen, users enter the exact date of the question, allowing the system to first verify whether the question already exists. If it does, the user will be redirected to the existing question.

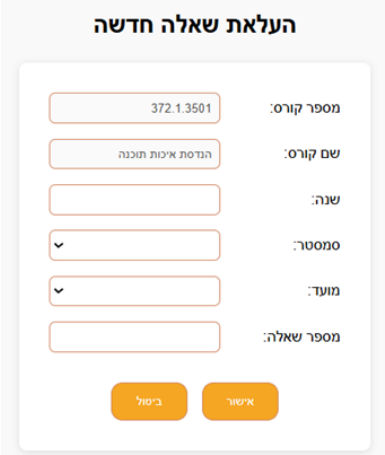



*Figure 44: Upload new Question – Step1 View*

Step 2 – In this screen, users upload question details as they appear in the exam. Fields for solution and topics are optional.

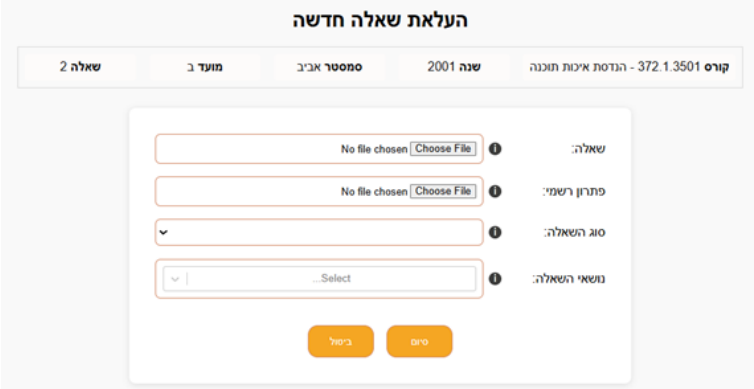



*Figure 45: Upload new Question - Step 2 View*

# 7. Testing

## a. Functional Requirements

- **Unit Testing**: We design and execute unit tests for all public methods, interfaces, and abstract classes. Every new method or class is thoroughly tested to ensure it operates as intended and delivers the required functionality. These tests focus on verifying the internal logic of each system component, including actions such as adding comments, searching for questions, and registering users.
- **Acceptance Testing**: Acceptance testing will be conducted to verify that the system satisfies the defined functional requirements and meets user expectations. This testing ensures that key use cases function as expected from the end user's perspective. Unlike unit tests, acceptance tests span multiple modules and classes, providing a broader validation of the system's behavior across different components.
- **Test Execution Before Commit:** As a best practice, we will run all tests prior to each commit to confirm that no part of the system has been unintentionally broken.

## b. Non-functional Requirements

1. **<u>Speed Requirements Testing</u>**

- **Registration –** The system should process registration requests within no longer than 30 seconds.

| | |
|---|---|
| **Test Method** | Measure the time it takes to process the registration for a new user. |
| **Expected Result** | The registration process should not exceed 30 seconds, (assuming the user enters the auth code immediately). |

- **Login -** The system should process log-In to the system within no longer than 5 seconds.

| | |
|---|---|
| **Test Method** | Measure the time it takes for users to log in to the system. |
| **Expected Result** | Log-in time should not exceed 5 seconds. |

- **Search by Date/Keywords -** The system should display the search results within 5(Date)/15(Keyword) seconds.

| | |
|---|---|
| **Test Method** | Measure the time it takes for search results to appear when searching by date or keywords. |
| **Expected Result** | Search results should be displayed within 5-15 seconds, depending on the query. |

- **Uploads (Exam/Question) -** The system should finish the upload of specific question within 30 seconds and 1 minute for full exam.

| | |
|---|---|
| **Test Method** | Measure the time taken for uploading a question and a full exam. |
| **Expected Result** | Uploading a question should be completed within 30 seconds, and uploading an exam should finish within 1 minute. |

- **Post a Comment -** Post a comment to discussion should take no more than 5 seconds.

| | |
|---|---|
| **Test Method** | Measure the time it takes to post a comment. |
| **Expected Result** | Posting a comment should take no more than 5 seconds. |

- **Push Notifications -** The system will be sent within 3 second of a triggered event.

| | |
|---|---|
| **Test Method** | Measure the time it takes to send a notification. |
| **Expected Result** | notifications should be sent within 3 seconds. |

2. **Capacity Requirements Testing**

- **Concurrent Users -** The system will support up to 1,000 concurrent users during normal operation.

| | |
|---|---|
| **Test Method** | Simulate up to 1,000 concurrent.<br>We will simulate a large volume of concurrent users by simulating multiple requests. |
| **Expected Result** | The system should handle 1,000 concurrent users without significant performance degradation. |

- **File Upload Limit -** The system will limit individual file uploads to 10 MB per file.

| | |
|---|---|
| **Test Method** | Upload files of various sizes, including exactly 10 MB, and monitor the system's response and performance. |
| **Expected Result** | The system should accept files up to the 10 MB size limit without issues. Files larger than 10 MB should be rejected with an appropriate error message. |

- **File Upload Rate Limiting -** User can upload up to 3 files per minute.

| | |
|---|---|
| **Test Method** | Upload 3 files per minute, ensuring that additional uploads are queued properly when the rate limit is exceeded. Monitor system behavior for potential issues. |
| **Expected Result** | The system should successfully queue excess uploads without failure, ensuring that the rate limit of 3 files per minute per user is adhered to. |

3. **Safety and Security Testing**

- **Password Encryption -** User passwords and personal data should be stored using encryption standards: AES-256.

| | |
|---|---|
| **Test Method** | Test password encryption by verifying that passwords are stored using the AES-256 encryption standard and that passwords are not retrievable in plain text from the database. |
| **Expected Result** | Passwords should be encrypted and stored securely using AES-256. They should not be accessible in plain text under any circumstance. |

- **Restrict Password -** The system will reject passwords that do not meet the defined strength criteria, such as being shorter than 8 characters or lacking uppercase or special characters.

| | |
|---|---|
| **Test Method** | Validate password strength requirements by attempting to register users with weak passwords (e.g., less than 8 characters, lacking uppercase or special characters) and ensuring the system rejects them. |
| **Expected Result** | The system should reject any password that does not meet the strength criteria. |

- **Token -** The system should generate valid tokens, and expire them after a specified time, and prevent the reuse of expired tokens.

| | |
|---|---|
| **Test Method** | Test the token-based authentication system by verifying that tokens are generated correctly on login, expire after a set time, and cannot be reused. |
| **Expected Result** | Tokens should expire after a set time and should not allow unauthorized access beyond their expiration. |

- **BGU users -** The system should only allow registrations of BGU students.

| | |
|---|---|
| **Test Method** | Simulate the registration of a user from an email domain that is not affiliated with Ben-Gurion University (e.g., a non-"bgu.ac.il" email address) and confirm the system rejects these registrations. |
| **Expected Result** | The system should only allow registrations from email addresses with the domains “bgu.ac.il” and “post.bgu.ac.il” and reject others. |

4. **Portability Testing**

- **Cross-Browser Compatibility –** The system should function seamlessly across all supported browsers, ensuring consistent performance and user experience.

| | |
|---|---|
| **Test Method** | We will validate use cases across all supported browsers to ensure consistent functionality. |
| **Expected Result** | The system should function seamlessly across all supported browsers without any issues. |

- **Mobile Compatibility – (Future Scope) -** The system should be fully functional and provide an optimal user experience across all supported mobile platforms and devices.

| | |
|---|---|
| **Test Method** | We will test use cases on all supported mobile platforms as part of future development. |
| **Expected Result** | The system should function flawlessly across all supported mobile devices and operating systems. |

5. **Reliability Testing**

- **Server Recovery -** In case of a server failure, the system should recover and return to full functionality within 10 minutes.

| | |
|---|---|
| **Test Method** | Simulate a server failure and measure recovery time. |
| **Expected Result** | The system should recover and return to full functionality within 10 minutes. |

- **Data Integrity -** Data integrity must be preserved, ensuring no corruption occurs during server crashes or shutdowns.

| | |
|---|---|
| **Test Method** | Simulate server crashes during critical operations and verify that data is intact after restart. |
| **Expected Result** | Data integrity should be maintained, and No data corruption should occur during shutdowns or crashes. |

6. **Usability Testing**

- **User Interface Testing -** The user interface should be intuitive and easy to navigate.

| | |
|---|---|
| **Test Method** | Conduct user testing with various participants to evaluate ease of use. |
| **Expected Result** | The interface should be intuitive and easy to navigate. At least 90% of users should be able to complete key tasks with minimal guidance. |

7. **Availability Testing**

- **System Uptime -** The system is operational and accessible 24/7, except during scheduled maintenance.

| | |
|---|---|
| **Test Method** | Monitor the system's uptime over an extended period. |
| **Expected Result** | The system should be operational and accessible 24/7, except during scheduled maintenance. |