NegevNerds

Collaborative Exam Preparation Platform

Test Ducoment

Ben-Gurion University

Software Engineering Department

Project Advisor:

Dr. Elior Sulem

Team Members:

Shachar Cohen

Nadav Ktalav

Noa Aboody

David Volodarsky

NegevNerds Test Document

This document outlines the testing methodology, test scenarios, and validation strategies used to evaluate the functionality, reliability, and performance of the **NegevNerds** system – a collaborative web-based platform aimed at enhancing the exam preparation experience for university students. The platform enables students to share, discuss, and answer exam-related questions, fostering a community-driven approach to studying.

The main objective of this document is to ensure that all system components – including the backend frontend interface, database , and user workflows – operate according to the design specifications and provide a seamless and error-free user experience.

Purpose of the Tests

The purpose of the testing process is to verify that the system behaves as expected under various scenarios and usage conditions. Since the platform is designed to support key interactions such as posting questions, answering them, managing user accounts, and browsing shared content, each of these functionalities must be tested rigorously to identify and resolve any bugs or inconsistencies.

Testing also aims to validate the following aspects of the system:

- Functional correctness: ensuring that each feature performs its intended task.
- Usability: verifying that users can navigate and use the system easily and intuitively.
- Performance: assessing how the system performs under load and with large amounts of data.
- Security: confirming that user data is protected and that access controls are properly enforced.
- Compatibility: checking the platform's behavior across different browsers and devices.

By documenting the testing process in a structured and organized manner, we aim to provide a clear reference for current and future development, maintenance, and quality assurance efforts.

Table of Contents

Functional Requirements:

1.1 UC 1 – register

Test Case 1: Verify that a guest can register using valid BGU email and matching passwords.

- **Steps:**
    1. Create a NegevNerds instance.
    2. Call the registration method with: first name, last name, BGU email, password, confirm password, and terms accepted.
    3. Simulate email verification by providing the correct verification code.
- **Expected Result:**
  The guest is registered, stored in the database, and marked as a verified user.

Test Case 2: Verify error is raised for non-BGU email during registration.

- **Steps:**

1. Create a NegevNerds instance.
2. Call the registration method with a non-BGU email.

- **Expected Result:**
  Registration fails with an error: "Only BGU emails are allowed."

Test Case 3: Verify error is raised for already-registered email.

- **Steps:**

1. Register a user with a specific BGU email.
2. Attempt to register another user with the same email.

- **Expected Result:**
  Registration fails with an error: "Email already registered."

Test Case 4: Verify error is raised for mismatching passwords.

- **Steps:**

1. Call the registration method with mismatched password and confirm password.

- **Expected Result:**
  Registration fails with an error: "Passwords do not match."

Test Case 5: Verify error is raised for incorrect verification code.

- **Steps:**

1. Complete the registration step (valid info).
2. Provide an incorrect verification code.

- **Expected Result:**
  Verification fails with an error: "Invalid verification code."

1.2 UC 2 – Login

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

Test Case 1: Verify user can successfully log in with valid credentials.

- **Steps:**
  1. Create a NegevNerds instance.
  2. Register a user with a valid email and password.
  3. Call the login method with the same email and password.
- **Expected Result:**
  The system authenticates the user, creates an active session, and sets user status as logged in.

Test Case 2: Verify login fails with invalid credentials.

- **Steps:**
  1. Create a NegevNerds instance.
  2. Attempt to call the login method with an incorrect password or unregistered email.
- **Expected Result:**
  The system rejects the login and returns: "Invalid email or password. Please try again."

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.

Test Case 1: Verify user can successfully register for an existing course.

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create and log in a user.
  3. Add a course to the system with a valid courseID.
  4. Call the course registration method with the user's ID and the courseID.
- **Expected Result:**
  The user is added to the course's participant list, and the course appears on the user's homepage.

Test Case 2: Verify error is returned if the user is already registered for the course.

- **Steps:**

1. Register a user to a course.
2. Attempt to register the same user again to the same course.

- **Expected Result:**
  Registration fails with the error: "User is already enrolled in the course."

Test Case 3: Verify error is returned if the course does not exist.

- **Steps:**
  1. Create and log in a user.
  2. Call the course registration method with a non-existent courseID.
- **Expected Result:**
  Registration fails with the error: "Course not found. Would you like to create it?"
- **Test Case 4:** Verify that registered course appears in the user instance.
- **Steps:**
  1. Register a user to a valid course.
  2. Retrieve the user courses.
  3. Check if the current course apper in the courses
- **Expected Result:**
  The registered course is listed in the user's course list.

1.4 UC 4 – Open a Course

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

Test Case 1: Verify user can successfully open a new course with valid details.

- **Steps:**
  1. Create a new NegevNerds instance.
  2. Create a new user with valid credentials.
  3. Log in the user.
  4. Call the open_course() method wit good parameters
- **Expected Result:**
  The course is added to the system.
  The user becomes the course manager.
  The user is automatically registered to the course.
  The course exist.

Test Case 2: Verify error is returned if the course already exists.

- **Steps:**
    1. Create a new NegevNerds instance.
    2. Create and log in a user.
    3. Open a course with a specific course_id.
    4. Attempt to open another course with the same course_id.
- **Expected Result:**
  The system returns the error: "Course already exists in the platform."

Test Case 3: Verify user becomes course manager after opening a course.

- **Steps:**
    1. Create a new NegevNerds instance.
    2. Create and log in a user.
    3. Open a valid course using the user's ID.
    4. Retrieve the course manager for that course.
- **Expected Result:**
  The user who opened the course is listed as the course manager.

1.5 UC 5 – Post an Exam

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

Test Case 1: Verify user can successfully upload a valid exam file

- **Steps:**
    1. Create a new NegevNerds instance.
    2. Create a course and assign a course manager.
    3. Create and log in a user who is registered to that course.
    4. Call the upload_exam() method with valid parameters
- **Expected Result:**

  The system extracts the exam details.
  The exam is stored in the system database.
  The exam is listed on the course page.

Test Case 2: Verify error is returned when uploading a file with invalid format

- **Steps:**
    1. Create a new NegevNerds instance.
    2. Create and log in a user.
    3. Call the upload_exam() method with an invalid exam file (e.g., unsupported format or missing metadata).

- **Expected Result:**
The system returns an error: "Invalid file format. Please upload a supported exam file."

Test Case 3: Verify error is returned when the course is not on the website

- **Steps:**
    1. Create a new NegevNerds instance.
    2. Create and log in a user.
    3. Call the upload_exam() method with an exam file that refers to a non-existing course.

- **Expected Result:**

    The system returns an error: "Course not found on the platform."
    The system suggests the user to open the course.

Test Case 4: Verify error is returned if the exam already exists

- **Steps:**
    1. Create a new NegevNerds instance.
    2. Create and log in a user.
    3. Upload a valid exam file.
    4. Attempt to upload the same exam file again.
- **Expected Result:**
The system returns an error: "Exam already exists on the platform."

1.6 UC 6 – Post an Exam's Question

Test Case 1: Verify user can successfully post a valid exam question

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create a user and log them in.
  3. Create a course and an exam for that course.
  4. Callt the add_question() with valid details (question number, is American, file).
- **Expected Result:**
  The question is successfully posted.

Test Case 2: Verify error is returned if the course doesn't exist

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create and log in a user.
  3. Call add_question() with a non-existing course_id.
- **Expected Result:**
  Error: "Course not found on the platform."

Test Case 3: Verify error is returned if the question already exists

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create and log in a user.
  3. Add a valid question.
  4. Attempt to add the same question again.
- **Expected Result:**
  Error: "This question already exists in the system."

Test Case 4: Verify successful question post without one necessary parameter

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create and log in a user.

3. Create a course and an exam.
4. Post a valid question without one necessary parameter.

- **Expected Result:**
  The question is not posted and raise an error.

1.7 UC 7 – Search Questions by Text

Test Case 1: Verify user can search and retrieve matching questions

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create and log in a user.
  3. Add some questions to the system with text content.
  4. Call the search_questions() method with text input related to one of the questions.
- **Expected Result:**
  The system processes the input text and returns a list of matching questions.

Test Case 3: Verify multiple matching questions are returned for a common search term

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create and log in a user.
  3. Add multiple questions with similar text content.
  4. Call search_questions() with a keyword that matches several questions.
- **Expected Result:**
  The system returns a list of all matching questions.

Test Case 4: Verify search functionality with partial text match

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create and log in a user.
  3. Add questions with varying degrees of similarity to the search term.
  4. Call search_questions() with a partial text match (e.g., a substring).
- **Expected Result:**
  The system returns questions that partially match the input text.

Test Case 5: Verify system handles case-insensitive search

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create and log in a user.
    3. Add a question with a specific case-sensitive text.
    4. Call search_questions() with the same text but in a different case (e.g., uppercase instead of lowercase)
- **Expected Result:**
  The system returns the matching question, showing that the search is case-insensitive.

1.8 UC 8 – Search Questions by Topic

Test Case 1: Verify user can search and retrieve matching questions by topic

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create and log in a user.
    3. Add some questions with specific topics to the system.
    4. Call search_questions_by_topic() method with a topic related to one of the questions.
- **Expected Result:**
  The system processes the input topic and returns a list of matching questions.

Test Case 2: Verify error message is returned if no questions match the topic

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create and log in a user.
    3. Call search_questions_by_topic() with a topic that doesn’t match any questions in the system.
- **Expected Result:**
  The system returns the message: "No results found."

Test Case 3: Verify multiple matching questions are returned for a common topic

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create and log in a user.
    3. Open new course.
    4. Add multiple questions with the same topic.
    5. Call search_questions_by_topic() with the topic.
- **Expected Result:**
  The system returns a list of all questions that share the given topic.

Test Case 4: Verify search functionality with partial topic match

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create and log in a user.
    3. Open new course.
    4. Add questions with topics related to the search term, some with partial matches.
    5. Call search_questions_by_topic() with a partial topic match.
- **Expected Result:**
  The system returns questions that partially match the input topic.

1.9 UC 9 – Search a Question by Specifics

Test Case 1: Verify user can search and retrieve a specific question using details

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create and log in a user.
    3. Open new course
    4. Add multiple questions with specific details (course name, year, semester, moed, question number).
    5. Call search_question_by_specifics() with specific details (course name, year, semester, moed, question number).
- **Expected Result:**
  The system processes the input details and returns the specific question homepage.

Test Case 2: Verify error message is returned if no question matches the specified details

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create and log in a user.
    3. Call search_question_by_specifics() with details that don't match any questions.
- **Expected Result:**
  The system returns the message: "No results found."

Test Case 3: Verify search functionality with partial details

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create and log in a user.
    3. Open new course.
    4. Add multiple questions with varying specific details.
    5. Call search_question_by_specifics() with partial details (e.g., only course or course + semester).
- **Expected Result:**
  The system returns a list of questions that partially match the provided details.

1.10 UC 10 – Comment on a Question

Test Case 1: Verify user can successfully add a comment to a question

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create a new user and log in.
    3. Open a course by calling the open_course() method.
    4. Add a question to the course by calling the add_question() method.
    5. Call add_comment() with a valid comment text.
- **Expected Result:**
  The system confirms that the comment is successfully added under the question.

Test Case 2: Verify error is returned if the comment is empty

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create a new user and log in.
    3. Open a course by calling the open_course() method.
    4. Add a question to the course by calling the add_question() method.
    5. Call add_comment() with an empty comment.
- **Expected Result:**
  The system returns an error message prompting the user to add content to the comment.

1.11 UC 11 – Comment on a Comment

Test Case 1: Verify user can successfully comment on another user's comment

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create a new user and log in.
    3. Open a course by calling the open_course() method.
    4. Add a question to the course by calling the add_question() method.
    5. Add a comment to the question by calling add_comment().
    6. Call add_comment () with the pred comment id to add a comment on the previous comment.
- **Expected Result:**
  The system confirms that the comment is successfully added.

Test Case 2: Verify error is returned if the comment is empty

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create a new user and log in.
    3. Open a course by calling the open_course() method.
    4. Add a question to the course by calling the add_question() method.
    5. Add a comment to the question by calling add_comment().
    6. Call add_comment() with the previous comment id an empty comment.
- **Expected Result:**
  The system returns an error message prompting the user to add content to the comment.

1.12 UC 12 – Edit a Comment

Test Case 1: Verify user can successfully edit their comment

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create a new user and log in.
    3. Open a course by calling the open_course() method.
    4. Add a question to the course by calling the add_question() method.
    5. Add a comment to the question by calling add_comment().
    6. Call edit_comment() with updated content .
- **Expected Result:**
  The system confirms that the comment was successfully updated and is under the question with the new content.

Test Case 2: Verify error is returned if the edited comment is empty

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create a new user and log in.
    3. Open a course by calling the open_course() method.
    4. Add a question to the course by calling the add_question() method.
    5. Add a comment to the question by calling add_comment().
    6. Call edit_comment() with an empty comment.
- **Expected Result:**
  The system returns an error message prompting the user to add content to the comment,
  The comment content doesn't change.

1.13 UC 13 – Download Exam Full PDF File

Test Case 1: Verify user can successfully download the exam PDF

- **Steps:**
    1. Create a NegevNerds instance.
    2. Create a new user and log in.
    3. Open a course by calling the open_course() method.
    4. Add an exam to the course by calling the add_exam() method with a valid PDF file.

5. Call the download_exam_pdf() method for the exam.

- **Expected Result:**
  The system successfully retrieves the PDF file and downloads it to the user's device.

Test Case 2: Verify error is returned if the exam PDF file is missing

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create a new user and log in.
  3. Open a course by calling the open_course() method.
  4. Add an exam to the course but do not provide a PDF file.
  5. Call the download_exam_pdf() method for the exam.
- **Expected Result:**
  The system returns an error message stating "File Not Found" and the PDF is not downloaded.

1.14 UC 14 – Upload Exam Full PDF File

Test Case 1: Verify user can successfully upload a valid PDF exam file

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create a new user and log in.
  3. Open a course by calling the open_course() method.
  4. Call the add_exam() method with a valid PDF file for the exam.
  5. Verify that the file is uploaded and associated with the exam.
  6. Call the download_exam_pdf() method to verify the file is available for users to download.
- **Expected Result:**
  The PDF exam file is successfully uploaded, associated with the exam, and available for users to download.

Test Case 2: Verify error is returned if an unsupported file format is uploaded

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create a new user and log in.
  3. Open a course by calling the open_course() method.

4. Attempt to upload a non-PDF file (e.g., a Word document or image file) using the add_exam() method.

- **Expected Result:**
  The system displays an error message stating "Unsupported file format" and the file is not uploaded.

Test Case 3: Verify user cannot upload a file if the exam already exists

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create a new user and log in.
  3. Open a course by calling the open_course() method.
  4. Add an exam to the course with a valid PDF file.
  5. Attempt to upload a new PDF file for the same exam.
- **Expected Result:**
  The system returns an error message stating "Exam already exists" and does not upload the new file.

1.15 UC 15 – React to a Comment with an emoji

Test Case 1: Verify user can successfully react to a comment with an emoji

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create a new user and log in.
  3. Open a course by calling the open_course() method.
  4. Add a question and post a comment on it.
  5. Call the react_to_comment() method, selecting an emoji (e.g., 👍).
  6. Verify that the emoji reaction is exist under the comment.
- **Expected Result:**
  The selected emoji is displayed as a reaction under the comment.

Test Case 2: Verify user can change their reaction to a comment

- **Steps:**
  1. Create a NegevNerds instance.
  2. Create a new user and log in.
  3. Open a course by calling the open_course() method.
  4. Add a question and post a comment on it.
  5. React to the comment with an initial emoji (e.g., 💔).
  6. Change the emoji reaction to another emoji (e.g., 😂).
  7. Verify new emoji is the user reaction in the comment.
- **Expected Result:**
  The emoji reaction is updated, and the new emoji is exist under the comment.

1.16 UC 16 – Delete a Question

Test Case 1: Verify Course Manager can successfully delete a question

- **Steps:**
  1. Create a NegevNerds instance.
  2. Open a course by calling the open_course() method.
  3. Create a new Course Manager user and log in.
  4. Add a question to the course.
  5. Call the delete_question() method, passing the question ID.
  6. Confirm that the question and its associated comments and reactions are deleted.
- **Expected Result:**
  The question is removed from the system along with all its associated comments and reactions.

Test Case 2: Verify error if the question does not exist when trying to delete

- **Steps:**
  1. Create a NegevNerds instance.
  2. Open a course
  3. Create a new Course Manager user and log in.
  4. Attempt to delete a non-existing question by passing a non-valid question ID to the delete_question() method.
- **Expected Result:**
  The system returns an error message stating "Question not found."

1.17 UC 17 – Edit a Question

Test Case 1: Verify Course Manager can successfully edit a question

- **Steps:**
    1. Create a NegevNerds instance
    2. Open a course by calling the open_course() method.
    3. Create a new Course Manager user and log in.
    4. Add a question to the course.
    5. Call the edit_question() method, passing updated parameters (e.g., year, semester, topics, etc.).
    6. Verify that the question details are updated correctly in the system.
- **Expected Result:**
  The question details are updated, and the changes saved.

1.18 UC 18 – Delete a Comment

Test Case 1: Verify Course Manager can successfully delete a comment

• **Steps:**

1. Create a NegevNerds instance.
2. Open a course by calling the open_course() method.
3. Create a new Course Manager user and log in.
4. Add a question to the course.
5. Add a comment to the question.
6. Call the delete_comment() method on the comment.
7. Verify that the comment deleted value is true.
8. Verify that all reactions to the comment are removed.
9. Verify that replies to the deleted comment remain exist.

- **Expected Result:**
  The comment is deleted, all reactions are removed.

Test Case 2: Verify comment writer user can successfully delete a comment

• **Steps:**

1. Create a NegevNerds instance.
2. Add a new user to the system.
3. Open a course by calling the open_course() method.
4. Add a question to the course.
5. Add a comment to the question.
6. Call the delete_comment() with the user id of the user who added the comment.
7. Verify that the comment deleted value is true.
8. Verify that all reactions to the comment are removed.
9. Verify that replies to the deleted comment remain exist.

- **Expected Result:**
  The comment is deleted, all reactions are removed.

Test Case 3: Verify regular user cannot delete a comment.

• **Steps:**

1. Create a NegevNerds instance.
2. Add a new user to the system.
3. Open a course by calling the open_course() method.
4. Add a question to the course.
5. Add a comment to the question.
6. Call the delete_comment() with user id other than the user who added the comment.
7. Verify that the comment deleted value is false.

- **Expected Result:**
  The comment isn't deleted, all reactions arent removed.

1.19 UC 19 – Appoint a User as a Course Manager

Test Case 1: Verify Course Manager can successfully appoint a user

• **Steps**:

1. Create a NegevNerds instance.
2. Add two users to the system: one as the current Course Manager and one as the user to be appointed.
3. Open a course using the open_course() method.
4. Assign the first user as a Course Manager of the course.
5. Call the appoint_course_manager() method with the email of the second user.
6. Simulate the second user accepting the appointment.
7. Verify that the user was added as a Course Manager to the course.
8. Verify that the user has Course Manager permissions in the course.

• **Expected Result**:
The second user is successfully appointed and now has Course Manager permissions for the course.

Test Case 2: Verify user can decline Course Manager role

• **Steps**:

1. Create a NegevNerds instance.
2. Add two users to the system: one as the current Course Manager and one as the user to be appointed.
3. Open a course using the open_course() method.
4. Assign the first user as a Course Manager of the course.
5. Call the appoint_course_manager() method with the email of the second user.
6. Simulate the second user **declining** the appointment.
7. Verify that the user's role did not change.
8. Verify that the user was **not** added as a Course Manager to the course.

• **Expected Result**:
The user is not appointed, and no changes are made to the user’s role.

Test Case 3: Verify system handles appointment of user who is already a Course Manager

• **Steps**:

1. Create a NegevNerds instance.
2. Add a user to the system.
3. Open a course using the open_course() method.
4. Assign the user as a Course Manager of the course.
5. Call the appoint_course_manager() method with the same user's email.
6. Verify that the system returns a message indicating the user is already a Course Manager.

• **Expected Result**:
The system prevents duplicate appointment and shows a relevant message.

Test Case 4: Verify system handles appointment with invalid email

• **Steps**:

1. Create a NegevNerds instance.
2. Add one user (Course Manager) to the system.
3. Open a course using the open_course() method.
4. Assign the user as a Course Manager of the course.
5. Call the appoint_course_manager() method with an email that does **not** exist in the system.
6. Verify that the system throws an error or returns a message indicating the email is invalid.

• **Expected Result**:
The system does not initiate the appointment and shows an error for the invalid email.

1.20 UC 20 – Remove a Course Manager

Test Case 1: Verify System Manager can successfully remove a Course Manager

• **Steps**:

1. Create a NegevNerds instance.
2. Add two users: one System Manager, one Course Manager.
3. Open a course using the open_course() method.
4. Assign the second user as a Course Manager for the course.
5. Call remove_course_manager() with the course ID and the Course Manager's email.
6. Verify that the user no longer has Course Manager permissions for the course.

• **Expected Result**:
The user is removed from the Course Manager role, permissions are revoked.

Test Case 2: Verify system prevents removal if user is not a Course Manager

• **Steps**:

1. Create a NegevNerds instance.
2. Add a System Manager and a regular user to the system.
3. Open a course using the open_course() method.
4. Ensure the user is **not** assigned as a Course Manager.
5. Call remove_course_manager() with the course ID and the user's email.
6. Verify that the system returns an appropriate error or message indicating the user is not a Course Manager.
7. Verify that no changes are made to the user’s roles.

• **Expected Result**:
The system prevents the removal and shows an appropriate message. No role changes occur.

Test Case 3: Verify system handles invalid email during removal

• **Steps**:

1. Create a NegevNerds instance.
2. Add a System Manager to the system.
3. Open a course using the open_course() method.
4. Call remove_course_manager() with a course ID and an email that does **not** belong to any user.
5. Verify that the system returns an error or failure message.
6. Verify that no roles are modified.

• **Expected Result**:
The system does not perform the removal and shows an error due to invalid email.

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

Test Case 1: Verify user can successfully download a ZIP of all exams

• **Steps**:

1. Create a NegevNerds instance.
2. Add a user and log in.
3. Open a course using the open_course() method.
4. Upload at least one exam with associated files to the course.
5. Call download_all_exams() on the course.
6. Verify that a ZIP file is generated.
7. Verify the ZIP file contains all uploaded exams.
8. Verify the file is available to the user for download.

• **Expected Result**:
The ZIP file is successfully generated and downloaded with all exam files included.

Test Case 3: Verify error message is shown when no exams exist

• **Steps**:

1. Create a NegevNerds instance.
2. Add a user and log in.
3. Open a course using the open_course() method.
4. Ensure no exams are uploaded for the course.
5. Call download_all_exams() for the course.
6. Verify that no ZIP file is generated.
7. Verify that the system displays a relevant message (e.g., "No exams available for download").

• **Expected Result**:
The system does not generate a ZIP file and informs the user that there are no exams to download.

Nonfunctional Requirements:

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

Test Case 1: Verify system processes registration requests within 30 seconds

• **Steps**:

1. Create a NegevNerds instance.
2. Start the registration process for a new user.
3. Measure the time it takes for the system to complete the registration.

• **Expected Result**:
The registration process should complete within 30 seconds, assuming the user enters the authentication code immediately.

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

Test Case 1: Verify system processes log-in requests within 5 seconds

• **Steps**:

1. Create a NegevNerds instance.
2. Add a user instance.
3. Start the log-in process for the user.
4. Measure the time it takes for the system to complete the log-in.

• **Expected Result**:
The system should complete the log-in process within 5 seconds.

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

Test Case 1: Verify system displays search results by date within 5 seconds

• **Steps**:

1. Create a NegevNerds instance.
2. Add course instance
3. Add question with specific date
4. Start a search by the question specific date.
5. Measure the time it takes for the system to return the search results.

• **Expected Result:**
The search results should be displayed within 5 seconds.

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

Test Case 1: Verify system uploads a question within 30 seconds

• **Steps**:

1. Create a NegevNerds instance.
2. Open new course
3. Add new exam to the course
4. Start the upload process for a question.
5. Measure the time it takes for the system to complete the upload.

• **Expected Result:**
Uploading a question should be completed within 30 seconds.

Test Case 2: Verify system uploads a full exam within 1 minute

• **Steps:**

1. Create a NegevNerds instance.
2. Start the upload process for a full exam.
3. Measure the time it takes for the system to complete the upload.

• **Expected Result:**
Uploading a full exam should be completed within 1 minute.

Post a Comment – Posting a comment to a discussion should take no more than 5 seconds.

Test Case 1: Verify system posts a comment within 5 seconds

• **Steps:**

1. Create a NegevNerds instance.
2. Add a new user.
3. Add a new course.
4. Add a question to the course.
5. Start the comment posting process on a discussion.
6. Measure the time it takes for the system to post the comment.

• **Expected Result:**
The system should complete the posting of the comment within 5 seconds.

Push Notifications – The system will send notifications within 3 seconds of a triggered event.

Test Case 1: Verify system sends push notifications within 3 seconds of an event

• **Steps:**

1. Create a NegevNerds instance.
2. Trigger an event that requires a push notification (e.g., user registration or comment post).
3. Measure the time it takes for the notification to be sent.

• **Expected Result:**
The notification should be sent within 3 seconds of the triggered event.

Capacity Requirements Testing

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

Test Case 1: Verify system supports 1,000 concurrent users without performance degradation

• **Steps**:

1. Simulate up to 1,000 concurrent users using a load testing tool (e.g., JMeter, Locust).
2. Send requests to the system from 1,000 simulated users (e.g., login, search, comment).
3. Monitor the system's performance (e.g., response times, error rates) during the simulation.

• **Expected Result**:
The system should handle 1,000 concurrent users without significant performance degradation (e.g., response times remain within acceptable limits, no crashes, no excessive errors).

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

Test Case 1: Verify system accepts files up to 10 MB

• **Steps**:

1. Upload a file that is exactly 10 MB in size.
2. Monitor the system's response and performance during the upload.

• **Expected Result**:
The system should accept files up to 10 MB without issues.

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

Test Case 1: Verify the system allows 3 file uploads per minute

• **Steps**:

1. Upload 3 files in 1 minute (e.g., every 20 seconds).
2. Monitor the system's behavior.

• **Expected Result**:
The system should allow exactly 3 file uploads per minute without any issues.

Safety and Security Testing

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

Test Case 1: Verify passwords are stored using AES-256 encryption

• **Steps**:

1. Register a new user with a password.
2. Verify that the password is stored in an encrypted format.

• **Expected Result**:
Passwords should be encrypted using the AES-256 standard and should not be retrievable in plain text from the database.

Restrict Password – The system will reject passwords that do not meet the defined strength criteria.

Test Case 1: Verify the system rejects weak passwords

• **Steps**:

1. Attempt to register a new user with a weak password (e.g., less than 8 characters, no uppercase or special characters).
2. Monitor the system's response.

• **Expected Result**:
The system should reject passwords that do not meet the defined strength criteria and prompt the user to provide a stronger password.

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

Test Case 1: Verify the system generates valid tokens

• **Steps**:

1. Log in as a user and obtain an authentication token.
2. Verify that the token is valid (e.g., can access protected resources).

• **Expected Result**:
The system should generate a valid token that allows access to protected resources.

Test Case 2: Verify the system expires tokens after a set time

• **Steps**:

1. Log in and obtain an authentication token.
2. Wait for the token expiration time to pass.
3. Attempt to use the expired token to access protected resources.

• **Expected Result**:
The token should expire after the specified time, and the system should deny access using the expired token.

Test Case 3: Verify the system prevents the reuse of expired tokens

• **Steps**:

1. Use a previously expired token to access protected resources.
2. Monitor the system's response.

• **Expected Result**:
The system should prevent the reuse of expired tokens and return an unauthorized error.

BGU Users – The system should only allow registrations of BGU students.

Test Case 1: Verify the system rejects registrations from non-BGU email addresses

• **Steps**:

1. Attempt to register a new user with an email address that is not affiliated with Ben-Gurion University (e.g., not "bgu.ac.il" or "post.bgu.ac.il").
2. Monitor the system's response.

• **Expected Result**:
The system should reject registrations from email addresses outside the "bgu.ac.il" and "post.bgu.ac.il" domains and display an appropriate error message.

Portability Testing

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

Test Case 1: Verify system functionality across supported browsers

- **Steps:**

1. Test the application on different supported browsers (e.g., Chrome, Firefox, Safari, Edge, etc.).
2. Validate that all major use cases (e.g., registration, login, file uploads, search) work as expected on each browser.
3. Monitor for UI issues, broken functionality, or differences in performance.

- **Expected Result:**
The system should function seamlessly across all supported browsers, with no significant issues in performance, UI, or functionality.

Mobile Compatibility – The system should be fully functional and provide an optimal user experience across all supported mobile platforms and devices.

Test Case 1: Verify system functionality on mobile devices

- **Steps:**

1. Test the application on various mobile platforms (e.g., iOS, Android) and devices (e.g., smartphones, tablets).
2. Validate that all critical use cases (e.g., registration, login, uploading exams) work as expected.
3. Test responsiveness, ensuring that UI components adjust well to different screen sizes.

- **Expected Result:**
The system should be fully functional and optimized for all supported mobile platforms and devices, providing an optimal user experience.

Reliability Testing

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

Test Case 1: Verify system recovery after server failure

- **Steps:**

1. Simulate a server failure by stopping the server or disconnecting the database.
2. Measure the time it takes for the system to recover and return to full functionality.
3. Validate that all services are operational after recovery.

• **Expected Result:**
The system should recover and return to full functionality within 10 minutes, with no loss of data or services.

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

Test Case 1: Verify data integrity after server crash

• **Steps:**

1. Simulate a server crash during a critical operation (e.g., user registration, exam upload).
2. Restart the server and check the system's data (e.g., database records, file uploads).
3. Verify that data is intact and no corruption has occurred.

• **Expected Result:**
Data integrity should be maintained, and no data corruption should occur during server crashes or shutdowns.

Usability Testing

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

Test Case 1: Verify usability of the user interface

• **Steps:**

1. Conduct usability testing with various participants (e.g., new users, experienced users, People with disabilities).
2. Ask participants to complete key tasks (e.g., logging in, posting a comment, downloading an exam) with minimal guidance.
3. Collect feedback regarding ease of navigation and any points of confusion.

• **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**.**

Availability Testing

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

Test Case 1: Verify system uptime

- **Steps:**

1. Monitor the system's up-time over an extended period (e.g., 5 days).
2. Track any downtime and determine if it was planned or unexpected.
3. Ensure that the system is always accessible.

- **Expected Result:**

The system should be operational and accessible 24/7, with no unexpected downtime outside of scheduled maintenance.

UI Testing

Test Case 1: Login Functionality

- **Steps:**
  1. Navigate to the NegevNerds login page.
  2. Enter valid credentials (username and password).
  3. Click the login button.

- **Expected Result**: User is successfully logged in and redirected to the dashboard.

Test Case 2: Course Search Functionality

- **Steps**:
  1. Navigate to the NegevNerds homepage.
  2. Enter a course name or keyword in the search bar.
  3. Click the search button.
- **Expected Result**: Search results should show a list of courses matching the search keyword, and the user can click on a course to open it.

Test Case 3: View Course Content

- **Steps:**
  1. Log into the NegevNerds platform.
  2. Navigate to a course from the course list.
  3. Click on the course to view its details, including questions and comments.
- **Expected Result:** The course page displays the list of questions and comments for that course.

Test Case 4: Post a Comment to a Question

- **Steps:**
  1. Log into the NegevNerds platform.
  2. Navigate to a course with questions.
  3. Click on a question.
  4. Add a comment in the comment section.
  5. Click the "Post Comment" button.
- **Expected Result**: The comment should appear below the question, and the comment should be visible to other users.

Test Case 5: Add a New Question to a Course

- **Steps:**
  1. Log into the NegevNerds platform as an admin or course manager.
  2. Navigate to the course where the question will be added.
  3. Click the "Add New Question" button.
  4. Enter the question details.
  5. Submit the question.
- **Expected Result**: The new question is added to the course, and it appears in the course's question list.

Test Case 6: Navigate Between Pages (Multi-View Navigation)

- **Steps:**
  1. Log into the NegevNerds platform.
  2. Navigate to the course list page.
  3. Click on a course to open it.
  4. From the course page, click on the "Home" link to return to the homepage.
- **Expected Result:** The user should be able to navigate back and forth between the homepage and the course page without issues.

Test Case 7: Search Questions by Subject

- **Steps:**
  1. Log into the NegevNerds platform.
  2. Navigate to the course list page.
  3. Apply a search to display questions by subject.
  4. View the filtered questions.
- **Expected Result:** Only courses belonging to the selected category should be displayed in the course list.

Test Case 8: Test Input Validation on Comment Submission

- **Steps:**
  1. Navigate to question page.
  2. Try to post an empty comment or one with only spaces.
- **Expected Result:** The system should block the submission and show an error message.

Test Case 9: Test Posting a New Question via exam

- **Steps:**
  1. Navigate to a Exam
  2. click “Add new Question"
  3. fill out the form
  4. submit.
- **Expected Result:** Question appears under the exam with correct details and timestamp.

Test Case 10: Test Posting a New Question via course

- **Steps:**
    1. Navigate to a course
    2. click “Add new Question"
    3. fill out the form
    4. submit.
- **Expected Result:** Question appears under the course with correct details and timestamp.

Test Case 11: Test Posting a New Question via home page

- **Steps:**
    1. Navigate to a home page
    2. click “Add new Question"
    3. fill out the form
    4. submit.
- **Expected Result:** Question appears under the course with correct details and timestamp.

Test Case 12: Test Uploading a Full Exam

- **Steps:**
    1. Go to the course page
    2. click “Upload Exam”
    3. upload a valid file.
    4. Click submit.
- **Expected Result:** File is uploaded, and success message appears.

Testing Run, Build & Deployment

To ensure that the NegevNerds system builds correctly and operates as intended across development and production environments, a structured approach was taken covering the **build process**, **component integration**, and **deployment pipeline**.

Build Process

- **Python 3.9 Interpreter**
  The backend system is built using **Python 3.9**, selected for its stability and compatibility with required libraries. This version ensures consistency across development, testing, and production environments.
- **Virtual Environment**
  A Python virtual environment (venv) is used to **isolate dependencies**, preventing conflicts with globally installed packages and allowing independent development and deployment environments.
- **Requirements File**
  A requirements.txt file is maintained and updated with the **exact package versions** required by the system. This guarantees reproducibility and consistency in both local development and production servers.

Example of requirements.txt:

```
Flask==1.1.2
SQLAlchemy==1.3.23
Pydantic==1.8.2
pytest==6.2.2
Flask-Cors==3.0.10
requests==2.25.1
```

This file ensures any environment can be quickly set up with:

- pip install -r requirements.txt

Run Process

Running all test by terminal-

1. Install pytest using "pip install pytest".
2. Navigate teminal to the NegevNerds directory.
3. Running all project test using "pytest".

Database for Tests

- the tests run will create a new DB called **test_NegevNerds.db**

Deployment

- **Smooth Deployment Process**
  Deployment is streamlined using:
  The requirements.txt file for dependency resolution.
  Environment variables for configuration.
- **Version Control & CI/CD Support- All project code, configurations, and requirements are managed in Git. This allows:**
  Easy rollback to previous stable states.
  Automatic environment setup for new developers.