1. **Test descriptions:**

Number	Description	type
System Administrator		
1.1.1 Installing	**Given:** A clean system with no prior installation. **Action:** Install the system using the provided installation package. **After:** The system is successfully installed without any errors.	Installation Test
1.1.2 Updating	**Given:** A system with a previous version installed. **Action:** Update the system using the provided update package. **After:** The system is successfully updated to the new version without any errors.	Regression Test
1.2 Install WMS	**Given:** A clean university system. **Action:** Install the WMS on the university system. **After:** The WMS is successfully installed and operational on the university system.	Installation Test
1.3 Clone WMS	**Given:** An operational WMS. **Action:** Use the cloning feature to create a duplicate WMS system. **After:** A cloned WMS system that operates identically to the original.	Functional Test
1.4 Update WMS	**Given:** An operational WMS. **Action:** Apply updates and patches to the WMS. **After:** The WMS is updated without any disruptions to its functionality.	Regression Test
1.5 Create new courses	**Given:** A functioning WMS. **Action:** Add new courses to the system. **After:** New courses are successfully added and available in the WMS.	Functional Test
1.6.1 Remove courses	**Given:** Existing courses in the WMS. **Action:** Remove the courses. **After:** Courses are successfully removed without affecting other data.	Functional Test
1.6.2 Backup, remove, and restore courses	**Given:** Existing courses in the WMS. **Action:** Backup the courses, remove them, then restore them from the backup. **After:** Courses are successfully backed up, removed, and restored without data loss.	Functional Test
1.7 Assign initial roles	**Given:** A new course in the WMS. **Action:** Assign initial roles for the course. **After:** Initial roles are successfully assigned to the course.	Functional Test
1.8.1 Remove roles	**Given:** Assigned roles in a course. **Action:** Remove the roles. **After:** Roles are successfully removed without affecting other data.	Functional Test
1.8.2 Change roles	**Given:** Assigned roles in a course. **Action:** Change the roles. **After:** Roles are successfully changed as specified.	Functional Test
Course Administrator		
2.1 Define course staff	**Given:** A new course in the WMS. **Action:** Define course staff. **After:** Course staff is defined and listed in the WMS.	Functional Test
2.2 Assign roles to staff	**Given:** Defined course staff. **Action:** Assign roles to the course staff. **After:** Roles are successfully assigned to the course staff.	Functional Test

2.3 Define exams	**Given:** A course in the WMS. **Action:** Define exams for the course. **After:** Exams are successfully defined and listed in the course.	Functional Test
2.4 Define exam type	**Given:** Defined exams in the WMS. **Action:** Define the type of each exam (e.g., test, quiz). **After:** Exam types are successfully defined and recorded.	Functional Test
2.5 Define exam direction	**Given:** Defined exams in the WMS. **Action:** Define the exam direction (RTL, LTR). **After:** Exam directions are successfully defined and recorded.	Functional Test
2.6 Define exam length	**Given:** Defined exams in the WMS. **Action:** Define the length of each exam. **After:** Exam lengths are successfully defined and recorded.	Functional Test
2.7 Define exam date	**Given:** Defined exams in the WMS. **Action:** Set the date for each exam. **After:** Exam dates are successfully set and recorded.	Functional Test
2.8 Define stylistic elements	**Given:** Defined exams in the WMS. **Action:** Define stylistic elements such as fonts and sizes for the exams. **After:** Stylistic elements are successfully defined and applied to the exams.	Functional Test
2.9 Define frontal matter	**Given:** Defined exams in the WMS. **Action:** Define the frontal matter (e.g., title pages) for the exams. **After:** Frontal matter is successfully defined and included in the exams.	Functional Test
2.10 Define headers	**Given:** Defined exams in the WMS. **Action:** Define headers for the exams. **After:** Headers are successfully defined and included in the exams.	Functional Test
2.11 Define instructions	**Given:** Defined exams in the WMS. **Action:** Define instructions for the exams. **After:** Instructions are successfully defined and included in the exams.	Functional Test
2.12 Define layout	**Given:** Defined exams in the WMS. **Action:** Define the basic layout (number of columns, number of items). **After:** Layout is successfully defined and applied to the exams.	Functional Test
2.13 Define number of versions	**Given:** Defined exams in the WMS. **Action:** Define the number of versions for each exam. **After:** The number of versions is successfully defined and ready for generation.	Functional Test
2.14 Select questions	**Given:** Defined exams in the WMS. **Action:** Select questions for each exam. **After:** Questions are successfully selected and included in the exams.	Functional Test
2.15 Select appendices	**Given:** Defined exams in the WMS. **Action:** Select appendices for each exam. **After:** Appendices are successfully selected and included in the exams.	Functional Test
2.16 Generate exam documents	**Given:** Defined exams in the WMS with selected questions and appendices. **Action:** Generate the exam documents. **After:** Exam documents are successfully generated and ready for use.	Functional/Integration Test

2.17 Generate exam versions	**Given:** Defined exams in the WMS. **Action:** Generate multiple versions of each exam. **After:** Multiple versions of each exam are successfully generated.	Functional Test
2.18 Generate special versions	**Given:** Defined exams in the WMS. **Action:** Generate special versions of exams for reading-impaired students. **After:** Special versions of exams are successfully generated.	Functional Test
2.19 Generate exam keys	**Given:** Generated exams in the WMS. **Action:** Generate exam keys (PDF, CSV). **After:** Exam keys are successfully generated and available in the specified formats.	Functional Test
2.20 Generate exam catalog	**Given:** Generated exams in the WMS. **Action:** Generate exam catalog documents. **After:** Exam catalog documents are successfully generated and available.	Functional Test
2.21 Generate solved exams	**Given:** Generated exams in the WMS. **Action:** Generate solved exams for student distribution. **After:** Solved exams are successfully generated and available.	Functional Test
2.22 Inspect changes	**Given:** Changes made by course staff in the WMS. **Action:** Inspect the changes and validate them. **After:** Changes are successfully inspected and validated as appropriate.	Functional Test
Course Staff		
3.1.1 Add questions	**Given:** A course with editable questions in the WMS. **Action:** Add new questions to the course. **After:** Questions are successfully added to the course.	Functional Test
3.1.2 Delete questions	**Given:** A course with editable questions in the WMS. **Action:** Delete questions from the course. **After:** Questions are successfully deleted from the course.	Functional Test
3.1.3 Edit questions	**Given:** A course with editable questions in the WMS. **Action:** Edit existing questions in the course. **After:** Questions are successfully edited as required.	Functional Test
3.1.4 Validate questions	**Given:** A course with editable questions in the WMS. **Action:** Validate questions to ensure they meet the necessary criteria. **After:** Questions are successfully validated and ready for use.	Functional Test
3.2.1 Add stems	**Given:** A course with editable stems in the WMS. **Action:** Add new stems to the course. **After:** Stems are successfully added to the course.	Functional Test
3.2.2 Delete stems	**Given:** A course with editable stems in the WMS. **Action:** Delete stems from the course. **After:** Stems are successfully deleted from the course.	Functional Test
3.2.3 Edit stems	**Given:** A course with editable stems in the WMS. **Action:** Edit existing stems in the course. **After:** Stems are successfully edited as required.	Functional Test
3.2.4 Validate stems	**Given:** A course with editable stems in the WMS. **Action:** Validate stems to ensure they meet the necessary criteria. **After:** Stems are successfully validated and ready for use.	Functional Test
3.3.1 Add keys	**Given:** A course with editable keys in the WMS. **Action:** Add new keys to the course. **After:** Keys are successfully added to the course.	Functional Test

3.3.2 Delete keys	**Given:** A course with editable keys in the WMS. **Action:** Delete keys from the course. **After:** Keys are successfully deleted from the course.	Functional Test
3.3.3 Edit keys	**Given:** A course with editable keys in the WMS. **Action:** Edit existing keys in the course. **After:** Keys are successfully edited as required.	Functional Test
3.3.4 Validate keys	**Given:** A course with editable keys in the WMS. **Action:** Validate keys to ensure they meet the necessary criteria. **After:** Keys are successfully validated and ready for use.	Functional Test
3.4.1 Add distractors	**Given:** A course with editable distractors in the WMS. **Action:** Add new distractors to the course. **After:** Distractors are successfully added to the course.	Functional Test
3.4.2 Delete distractors	**Given:** A course with editable distractors in the WMS. **Action:** Delete distractors from the course. **After:** Distractors are successfully deleted from the course.	Functional Test
3.4.3 Edit distractors	**Given:** A course with editable distractors in the WMS. **Action:** Edit existing distractors in the course. **After:** Distractors are successfully edited as required.	Functional Test
3.4.4 Validate distractors	**Given:** A course with editable distractors in the WMS. **Action:** Validate distractors to ensure they meet the necessary criteria. **After:** Distractors are successfully validated and ready for use.	Functional Test
3.5.1 Add solutions	**Given:** A course with editable solutions in the WMS. **Action:** Add new solutions to the course. **After:** Solutions are successfully added to the course.	Functional Test
3.5.2 Delete solutions	**Given:** A course with editable solutions in the WMS. **Action:** Delete solutions from the course. **After:** Solutions are successfully deleted from the course.	Functional Test
3.5.3 Edit solutions	**Given:** A course with editable solutions in the WMS. **Action:** Edit existing solutions in the course. **After:** Solutions are successfully edited as required.	Functional Test
3.5.4 Validate solutions	**Given:** A course with editable solutions in the WMS. **Action:** Validate solutions to ensure they meet the necessary criteria. **After:** Solutions are successfully validated and ready for use.	Functional Test

WMS Activities

4.1 User login	**Given:** The WMS login page. **Action:** Enter user ID and password to log in. **After:** The user is successfully logged into the system.	Security/Functional Test
4.2 Search for meta-question	**Given:** The WMS search functionality. **Action:** Search for a specific meta-question. **After:** The meta-question is successfully found and displayed.	Functional Test
4.3 Edit current exam	**Given:** An existing exam in the WMS. **Action:** Edit the exam settings as required. **After:** The exam settings are successfully edited and saved.	Functional Test
4.4 Suggest new questions and submit new questions to the Course Administrator.	**Given:** The interface for suggesting questions. **Action:** Write **After:**	Functional Test

	The new questions are successfully submitted and awaiting approval.	
4.5 Work on existing questions	**Given:** Editable questions assigned to the user. **Action:** Edit or improve the questions based on user role. **After:** Changes to the questions are successfully saved.	Functional Test
4.6 Request task	**Given:** The task request interface. **Action:** Request tasks sorted by urgency and category. **After:** Tasks are successfully requested and assigned.	Functional/Performance Test
4.7 Select course	**Given:** A user active in multiple courses. **Action:** Select the appropriate course from a list. **After:** The correct course is selected and active.	Functional Test

2. Testing non- functional requirements:

Implementation constraints:

1. Performance (Speed, Capacity, Throughput, etc.)

   The system should be able to support multiple users* (of the same or different types) using the system at the same time.

   Tested: We ran a remote server, connected and used the system with multiple computers. Each computer with different user.

   *The system is expected to serve only a handful of users with a maximum of ~10 users. Our system does is not expected to deal with heavy loads and with no stress during its whole lifetime.

2. Reliability & Stability
   1. the distractors and answers as well as past exams will be persisted in a database and will support data recovery.
      - Will be tested by deleting the require information from the system than recover it.
   2. In case of errors in internet connection, crash or hardware failure, the system will roll back all related updates until reaching the last stable version.
      - Will be tested by crashing the system and checking that we run a stable version when recover

3. Safety & Security
    1. The system will save any sensitive data such as passwords be encrypted.
        - The system hashes every password before sending it over the internet.
        - In production use, the system should be installed on university's servers and be accessed only through local network. No connection from outside should be available.
    2. The system will not allow any access to its sensitive data to users without - permission.
        - Test by trying to access data from a user without the right permission/
4. Portability
    1. The system is web based and can be accessed only while connected to the university's network.
        - We will work with the BGU VPN
    2. The system should be accessible from different browsers.
        - Using React which supports different browsers.
5. Usability
    1. The system's users do not have any special expertise in computers or programming; therefore, the system's interface should be as simple and clear as possible.
        - Continuously ask the customers of the design of the UI and how to make it simpler
6. Availability
    1. Unless the system is undergoing updates, the system should be available 24/7
        - Make sure that we have a stable version that will run non-stop on a server.

3. TDD development:

We have not used TDD development strategy.

TDD is a great way to develop, but due to time limitations for every milestone in the project we could not use TDD because of its time-consuming nature.

4. Random & automatically generated tests:

We have not used any random & automatically - generated test in our project.

5. Testing the user interface:

Since the system we are developing is not currently a system that is intended to serve 'the masses' but instead is tailored to the requests and requirements of a single person (the Client) we plan on performing a close-up user-study.

We will meet with the user on Zoom and will be asked to perform the following tasks:

1. Access the system through the browser and perform a first-time-login into a 'lecturer' type user.
2. Browse the Meta-Questions database and add a new Meta-Question to the system.
3. Complete day-to-day tasks such as 'Add a key to given Meta-Question' and 'Tag an existing answer as key/distractor'
4. Build an exam and publish it.

The user will be monitored during each task with the time it took him and if was able to complete the task with/without help.
After each task the user will rank his own feelings about the system by the intuitiveness, ease-of-use parameters by ranking each of these parameters between 1 and 5.

6. Testing Build, Integration & Deployment

Build Verification:

To ensure that the application builds cleanly, we performed manual testing at each stage of development. Our process included:

- Running the build commands to compile the application and checking for any errors or warnings.
- Ensuring that all dependencies are correctly installed and configured by executing `npm install` to install Node.js packages.

Integration Testing:

Our application is built using Node.js and React, and a PostgreSQL DB server.
To verify that the components integrate well:

- We manually tested the integration between the front-end (React) and back-end (Node.js) components.
- Ensured that the API endpoints are correctly called and data is properly exchanged between the client and server.
- Verified that the application functions as expected when all modules are combined.
- Built tests ensuring correct communication between the system and the DB server.

Deployment Testing:

During the deployment process, we conducted manual tests to ensure the application is installed properly:

- Installation Verification**:** After deploying the application, we manually accessed it through a web browser to ensure it runs correctly. This involved checking the UI, verifying that all features work as intended, and ensuring that the server is responsive.
- Uninstallation**:** We did not test the uninstallation process due to time constraints.
- Resource and Prerequisite Verification: To verify that the application has all the necessary resources and prerequisite software, we ensured that the server has Node.js installed. The following steps were taken:
    - Confirming Node.js installation by running `node -v` and `npm -v` to check the versions.
    - Running `npm install` to install all required packages listed in the `package.json` file.
    - Starting the application using `npm start` to ensure it launches without issues.

By following these steps, we aimed to ensure that the build, integration, and deployment processes were as smooth and error-free as possible, given our manual testing approach.