Faculty of Engineering and Technology

Computer Science Department

COMP438 – Software QA

Course Main Project

End-to-End Testing and Quality Assurance for a Web Application

Prepared By

Name: Amro Al Deek	ID: 1221642
Name: Saleh Al Deek	ID: 1172324
Name: Nagham Nazeeh	ID: 1220813
Name: Julia Daibes	ID: 1222428

June 16, 2025

Contents

1 Project Overview

1.1 Application Description

This project aims to design an electronic system for managing projects and their related tasks in a team setting. The system enables users to create projects and tasks, allocate team members to individual tasks, track the work progress, change task statuses, and either accept or decline tasks.

To facilitate organized workflows and efficient accomplishment of project objectives, the system is aimed at project managers, team leaders, and team members.

Project Components and Functions

The system includes a set of essential pages and functionalities, such as:

- **Login and Registration Pages:** Logging into the system through `signUp.php` and `signIn.php` enables users to register and log in with their credentials.
- **Management of Projects:** Pages like `addProjectForm.php`, `saveProject.php`, and `saveAllocation.php` allow users to add new projects, view their details, and assign team leaders and members.
- **Creation and Assignment of Tasks:** Using `taskCreationForm.php`, `task_assignment.php`, `update_task.php`, and `taskSearch.php`, users can create tasks, assign them, and update their status.
- **Accepting and Rejecting Tasks:** Through `acceptOrRejectTask.php`, team members can choose to accept or reject assigned tasks.
- **User Dashboard:** The dashboard displays assigned tasks, enables tracking of progress, and provides update tools.

1.2 Scope of Testing

This section defines the boundaries of the testing effort, outlining what was tested and what was not, as well as the testing techniques employed.

1.2.1 In-Scope

The following areas were included in our testing:

- **Static Testing:** Each team member reviewed one main PHP file from the project. We evaluated the code structure, readability, and checked for missing validations or security/session issues.

- **Black-Box Testing:** The system was tested through form inputs without looking at the source code.

  - Equivalence Partitioning was used to test valid and invalid input ranges (e.g., empty fields, valid names).
  - Boundary Value Analysis was applied to fields such as contribution (e.g., 0% and 100%) and date ranges.
  - Decision Tables were employed to verify logic correctness, such as preventing duplicate team members or handling missing roles.

- **White-Box Testing:** Internal logic and code paths were examined:

  - Statement coverage ensured every line of code was executed at least once.
  - Branch coverage verified all decision outcomes (e.g., task existence, assignment duplication).
  - Path coverage followed complete execution flows, such as task creation and database saving.

- **Regression Testing:** After bug fixes or code changes, we re-executed tests on login, task creation, and user assignments to confirm previous functionalities still worked.

1.2.2 Out-of-Scope

The following areas were not covered in this testing phase:

- Cross-browser or mobile responsiveness testing (only Google Chrome was used).
- External systems (e.g., email services or third-party APIs).
- In-depth security assessments such as penetration testing.
- Accessibility checks (e.g., screen reader compatibility).
- Performance or database optimization evaluations.

1.3 Team Responsibilities

1.3.1 Amro Al Deek

Led the QA process and took responsibility for the core backend testing. Specifically:

- Performed static code inspection on `saveProject.php`, identifying security issues (file uploads, CSRF, permissions, etc.).
- Refactored legacy code to support testability (e.g., extracted validation and insertion functions).
- Developed PHPUnit test cases for backend logic such as `insertProject` and `isValidDateRange`.
- Designed and executed 3 Katalon automated tests for login, logout, and dashboard access(add project,assign project to team leader,create task and search functionalities).
- Conducted black-box and white-box testing techniques and documented them.
- Helped configure the environment (Composer, PHPUnit, XAMPP) and led the report documentation in LaTeX.

1.3.2 Saleh Al Deek

- Created a complete inspection report for the `ProjectLeader` class.
- Designed black-box and white-box test cases, developed four scenarios, and integrated them with the team on Jira.
- Created four automated test cases using the Katalon environment.
- Participated in analyzing the system's functional and non-functional requirements.
- Contributed to performance testing using JMeter and participated in regression testing efforts.
- Developed unit tests, restructured the codebase, and separated core logic to enhance testability. Refactored classes and functions for better modularity and maintainability.
- Played a dual role by completing all individual tasks and actively contributing to collaborative group work.

1.3.3 Nagham Nazeeh

Led the functional and logical testing of the `update_task.php` module, ensuring the accuracy of task progress and status updates.

- Conducted static code inspection to verify:
  - Proper session handling
  - Input sanitization
  - Logical flow of progress calculation, including automatic transition to `"Completed"` when progress reaches 100%
- Designed and executed both black-box and white-box test cases using Katalon Recorder, covering:
  - Invalid input values
  - Progress exceeding the allowed limit (100%)
  - Non-existent task IDs
- Validated that updates are correctly reflected in both:
  - `tasks` table
  - `tasks_team_members` table
- Provided feedback that led to:
  - Enhanced validation logic
  - Clearer user error messages
  - Robust prevention of invalid progress submissions

1.3.4 Julia Daibes

- Led the functional and logical testing of the `assignTeamMembers.php` module, which is responsible for assigning users to project tasks.
- Reviewed database interactions to ensure proper validation of assignments, including prevention of duplicate member-task associations.
- Validated the logic behind contribution percentage calculations, ensuring that total assigned contributions do not exceed 100%.

- Designed and executed both manual and automated test cases using Katalon Recorder, covering:
  - Missing team member selection
  - Duplicate assignments
  - Missing role values
  - Valid and invalid input combinations
- Identified logic flaws related to contribution validation and ensured that proper error messages are rendered when inputs are invalid.
- Coordinated closely with the backend developer to improve form security and enhance the overall usability of the assignment process.

1.4 Requirement Analysis

1.4.1 Functional Requirements

- User registration with personal information.
- User login using username and password.
- Validation of login credentials.
- Project creation with details like title, description, client, budget, start and end dates.
- Assignment of a team leader to a project.
- Creation of new tasks linked to a specific project.
- Assignment of tasks to team members with defined roles and expected effort.
- Display of assigned task list for each user.
- Acceptance or rejection of assigned tasks.
- Updating task progress and completion percentage.
- Searching tasks by ID, name, or project name.
- Display of success or error messages during operations.
- Viewing and updating user profile information.

1.4.2 Non-Functional Requirements

Security:

- Session management for secure access.
- Use of prepared statements to prevent SQL injection.
- Client-side and server-side data validation.

Usability:

- Clear and simple user interfaces with disabled input fields for non-editable data.
- Clear messages for success or failure of operations.

Performance:

- Efficient database queries to retrieve only necessary data.
- Fast page loading with minimized data transfer.

Maintainability:

- Organized code into separate pages and modules.
- Use of PDO standard for database interactions.

Compatibility:

- Support for modern browsers through HTML5 and CSS.
- Responsive design (if suitable CSS is provided).

Scalability:

- Ability to add new features in the future such as reports, notifications, or additional permission levels.

This task and project management system provides a comprehensive solution to control the project life cycle, from user registration, project and task creation, team allocation, task acceptance or rejection, to progress tracking and status updates. It also ensures user data security and maintains ease of use while meeting performance and maintainability requirements.

2 Static Testing

2.1 Code Inspections

2.1.1 By Amro Al Deek

This inspection targeted the core logic behind the project upload feature, specifically the `saveProject.php` script in the PHP-based system. The analysis revealed both security and code quality concerns that need to be addressed to meet professional standards in software quality assurance.

1. Security – SQL Injection Prevention

The system correctly uses `PDO` with prepared statements and `bindValue()`, which helps prevent SQL injection attacks. This is a well-implemented practice and requires no change.

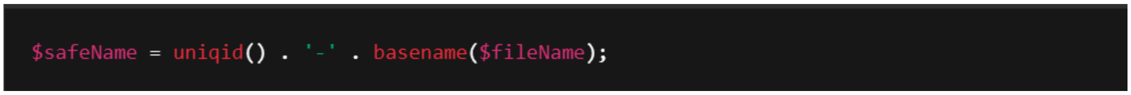
2. File Upload – Security Risks

Issues Identified:

- The file is saved using `basename($fileName)` without any renaming or sanitization. This can lead to:
  - **Filename Collisions:** If two users upload a file with the same name (e.g., `report.pdf`), the latter will overwrite the former, resulting in data loss or inconsistent system behavior.
  - **Disguised Malicious Files:** Attackers can rename a PHP script (e.g., `hack.php`) to `image.jpg` and upload it. If your system doesn't verify MIME types, the server might allow execution of malicious scripts.

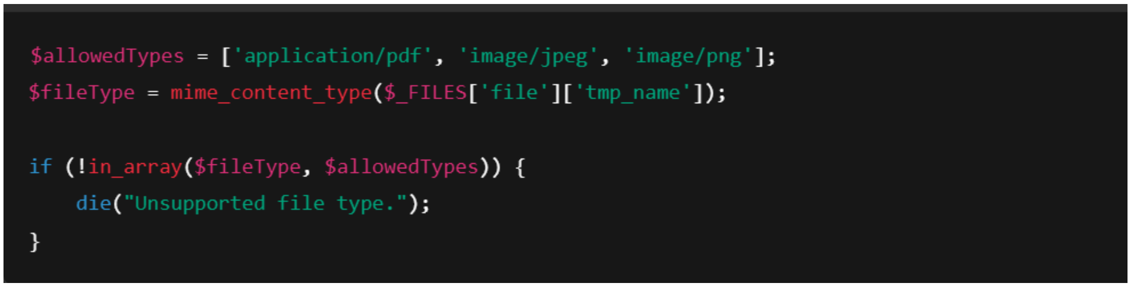
Recommended Fix:

- Always sanitize and rename uploaded files using unique identifiers to avoid collisions. Example approach:



```
$safeName = uniqid() . '-' . basename($fileName);
```

- Validate file MIME types and restrict allowed formats (e.g., PDF, JPG, PNG):



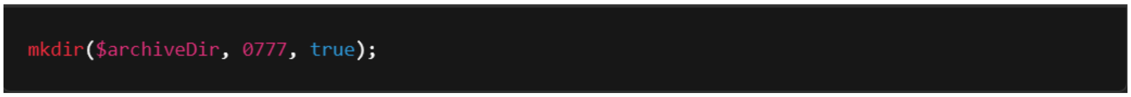
```php
$allowedTypes = ['application/pdf', 'image/jpeg', 'image/png'];
$fileType = mime_content_type($_FILES['file']['tmp_name']);

if (!in_array($fileType, $allowedTypes)) {
    die("Unsupported file type.");
}
```

- Reject unsupported file types immediately to reduce the attack surface and ensure server safety.

3. File Permissions

The use of :

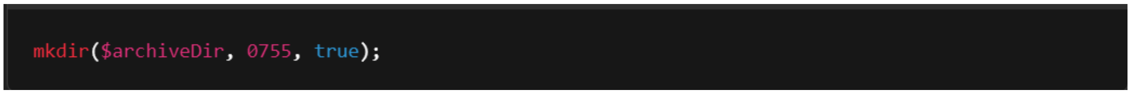


```php
mkdir($archiveDir, 0777, true);
```

grants unsafe full permissions to all users (owner, group, and others), which can lead to serious security vulnerabilities, including unauthorized file modifications or deletions.

Recommended Fix:

Use:



```php
mkdir($archiveDir, 0755, true);
```

This sets safer directory permissions by allowing:

- **Owner:** Read, write, execute
- **Group & Others:** Read, execute only (no write access)

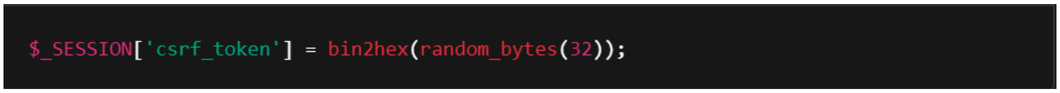
This prevents external users from altering or injecting malicious content into the directory.

4. Missing CSRF Protection

Forms are submitted without any CSRF tokens, making them vulnerable to Cross-Site Request Forgery (CSRF) attacks. This allows unauthorized commands to be submitted on behalf of logged-in users.

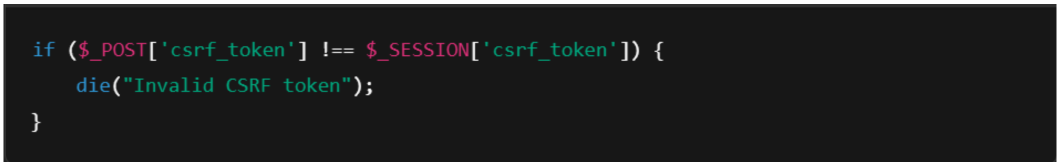
Recommended Fix:

- **Token Generation:** Store a secure CSRF token in the session before rendering the form.



```php
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));
```

- **Token Validation:** Verify that the token submitted matches the session token before processing POST requests.



```php
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("Invalid CSRF token");
}
```

CSRF tokens validate the authenticity of form submissions and ensure that requests are intentionally made by users through your application only.

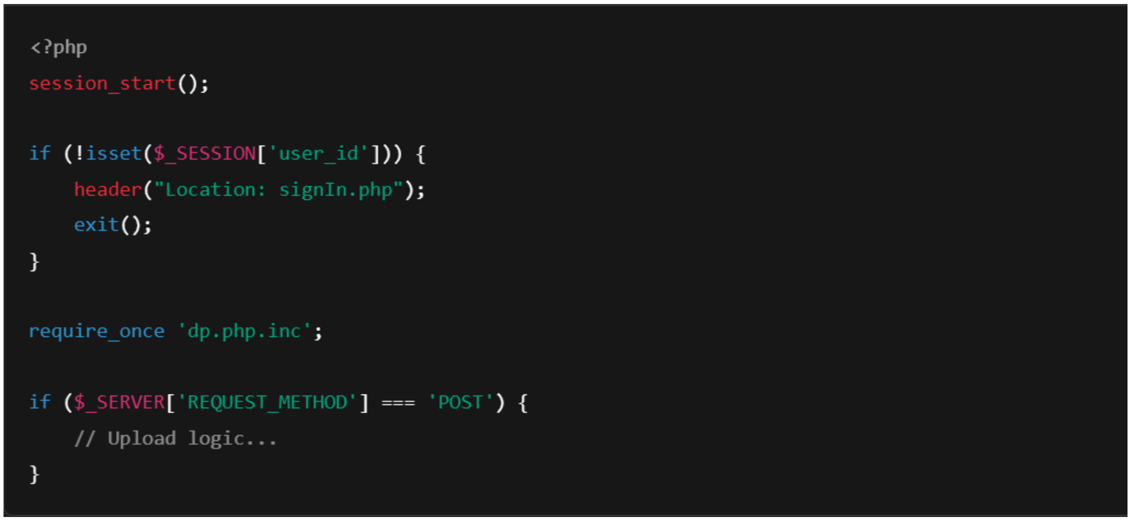
5. Validation – Good Practices Found

The following validation checks are present and effective:

- Ensures end date is after start date.
- Limits on file count and size.
- Requires titles for each uploaded file.

6. Authentication Check Missing

Sensitive upload logic runs without checking authentication:



```php
<?php
session_start();

if (!isset($_SESSION['user_id'])) {
    header("Location: signIn.php");
    exit();
}

require_once 'dp.php.inc';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Upload logic...
}
```

Recommended Fix: Always wrap protected features inside an authentication check like the one shown above.

2.1.2 By Saleh Al Deek

This static code inspection and review presents three modules within the project management system:

- Create Task
- Search Task
- Assign Team Members to Tasks

The objective is to assess code quality, logic correctness, security, performance, and maintainability. Recommendations are provided to improve the overall software quality.

1. Create Task Module

This module manages task creation with validation and persistence. Key observations:

- **Code Quality:** Variables are well-named; code is structured but lacks client-side validation.
- **Logic:** Proper checks for project existence and task name uniqueness are implemented.
- **Security:** Uses prepared statements which protect against SQL injection.
- **Performance:** Queries are optimized but could benefit from caching project data.
- **Maintainability:** Separation of concerns exists via `TaskService`, which is good practice.

Example snippet:

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Input reading and validation
    $projectDates = $taskService->getProjectDates((int)$project_id);
    if (!$projectDates) {
        $errorMessage = "Project not found.";
    }
}
```

2. Search Task Module

This module facilitates searching tasks by filters. Key points:

- **Code Quality:** Clear structure; filtering parameters are read securely.
- **Logic:** Filtering logic handles empty and specified filters well.
- **Security:** Prepared statements used, but input validation for date ranges is missing.

- **Performance:** Potential optimization by limiting fetched data when no filters are set.
- **Maintainability:** Simple and modular with `TaskService` abstraction.

Example snippet:

```
$filters = [
    ’priority’ => $priority,
    ’status’ => $status,
    // ...
];
$tasks = $taskService->getTasksByFilters($filters,
$_SESSION[’role’], $_SESSION[’user_id’]);
```

3. Assign Team Members Module

This module assigns team members to tasks ensuring unique assignments and contribution limits.

- **Code Quality:** Straightforward procedural code; variable names descriptive.
- **Logic:** Prevents duplicate assignments and checks cumulative contribution does not exceed 100%.
- **Security:** Uses prepared statements but lacks input sanitization and role-based access control.
- **Performance:** Executes multiple sequential queries; could be optimized.
- **Maintainability:** Mixing logic and redirection makes testing harder; recommend refactoring.

Example snippet:

```
if (isset($_POST[’team_member_id’]) && isset($_POST[’role’])) {
    // Check for duplicate assignment
    // Calculate total contribution
    // Insert assignment
}
```

4. Summary and Recommendations

Overall, the modules are functionally sound and utilize prepared statements for SQL security. However, improvements are recommended:

- Add client-side and server-side input validation (e.g., date ranges, numeric fields).

- Implement role-based access control to restrict sensitive operations.
- Refactor procedural code into modular classes/services to improve maintainability.
- Optimize database queries and consider caching frequently used data.
- Enhance error handling and user feedback.

Implementing these recommendations will increase code robustness, security, and maintainability.

2.1.3 By Nagham Nazeeh

This inspection reviews the logic behind the `update_task.php` script, which is responsible for updating task details through POST requests. Several issues were identified related to security, data validation, and missing checks, which could affect both data integrity and system safety.

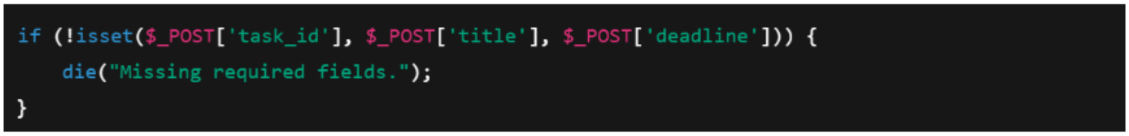
1. Security – SQL Injection Prevention

Positive: If the code uses `PDO` with `prepare()` and `bindValue()`, it is protected from SQL injection.

2. Required Field Validation

Issue: Fields such as `task_id`, `title`, and `deadline` are not checked before performing the update.

Recommended Fix: Add conditional checks to ensure all required POST variables are present.

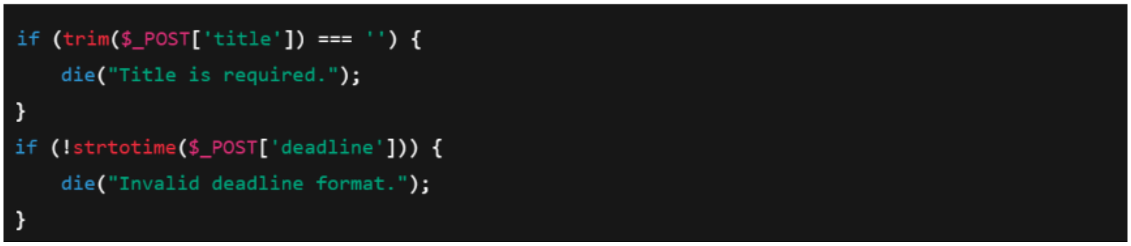


```
if (!isset($_POST['task_id'], $_POST['title'], $_POST['deadline'])) {
    die("Missing required fields.");
}
```

3. Empty Field / Data Validation

Issue: Submitted fields (e.g., `title` or `deadline`) may be empty or incorrectly formatted.

Recommended Fix: Check that fields are not empty and use regex or date parsing to ensure validity.

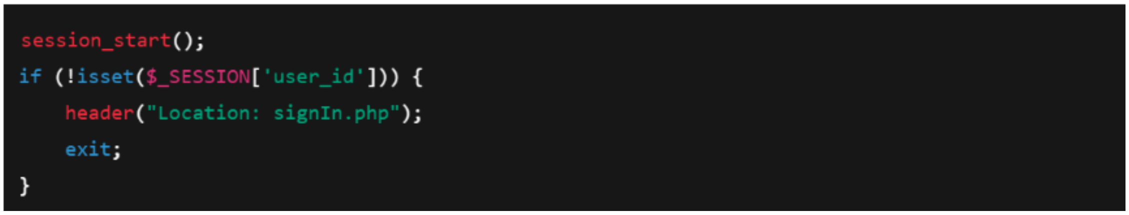


```
if (trim($_POST['title']) === '') {
    die("Title is required.");
}
if (!strtotime($_POST['deadline'])) {
    die("Invalid deadline format.");
}
```

4. Authentication Check Missing

Risk: If the script runs without checking whether the user is logged in, unauthorized users may update task data.

Recommended Fix: Wrap the script in an authentication check.



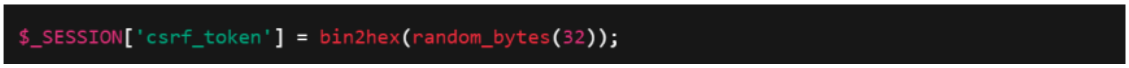
```
session_start();
if (!isset($_SESSION['user_id'])) {
    header("Location: signIn.php");
    exit;
}
```

5. CSRF Token Validation Missing

Issue: The script does not use CSRF protection, exposing it to Cross-Site Request Forgery attacks.

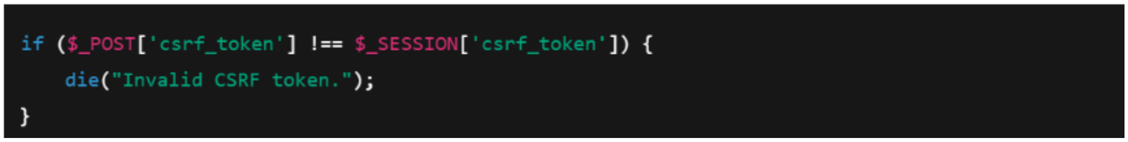
Recommended Fix:

- **Token Generation:** Store a CSRF token in session and embed it in the form.



```
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));
```

- **Token Validation:** Compare form token with session token before processing.

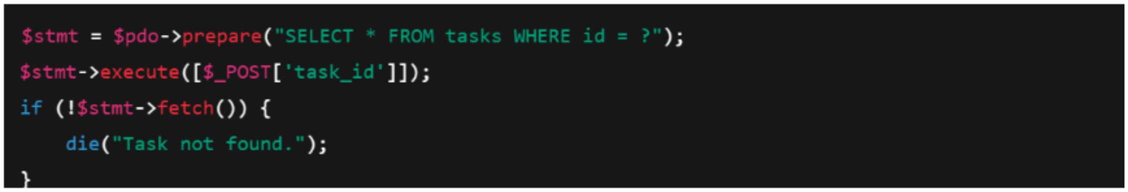


```
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("Invalid CSRF token.");
}
```

6. Task Existence Check Missing

Issue: The script may attempt to update a task that doesn't exist in the database.

Recommended Fix: Verify the existence of the task before updating by querying it by `task_id`.



```
$stmt = $pdo->prepare("SELECT * FROM tasks WHERE id = ?");
$stmt->execute([$_POST['task_id']]);
if (!$stmt->fetch()) {
    die("Task not found.");
}
```

2.1.4 By Julia Daibes

This inspection targeted the PHP script responsible for assigning team members to specific tasks — `assignTeamMembers.php`. The script receives `team_member_id`, `task_id`, `role`, and `contribution` via a POST request and inserts a new assignment after checking for duplication.

1. Security – SQL Injection Prevention

Positive:

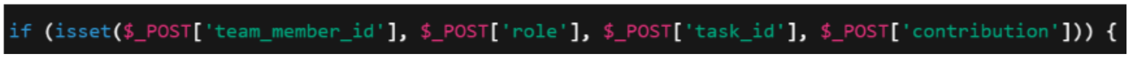
The script uses `PDO` with `bindValue()`, which protects against SQL injection. This is a good secure practice.

2. Input Validation – Missing Field Checks

Issue:

Only `team_member_id` and `role` are checked. `task_id` and `contribution` are missing.

Recommended Fix:



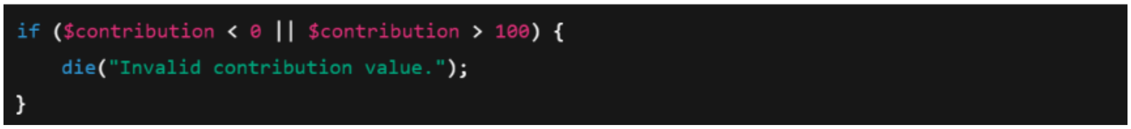
```
if (isset($_POST['team_member_id'], $_POST['role'], $_POST['task_id'], $_POST['contribution'])) {
```

3. Missing Range Validation on Contribution

Issue:

No check ensures `contribution` is between 0–100%.

Recommended Fix:



```
if ($contribution < 0 || $contribution > 100) {
    die("Invalid contribution value.");
}
```

4. Duplicate Assignment Handling

Positive:

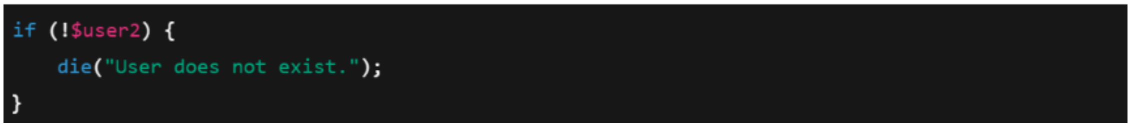
The script correctly checks for existing assignments to prevent duplicates.

5. User Existence Validation

Issue:

No check whether the fetched user actually exists.

Recommended Fix:



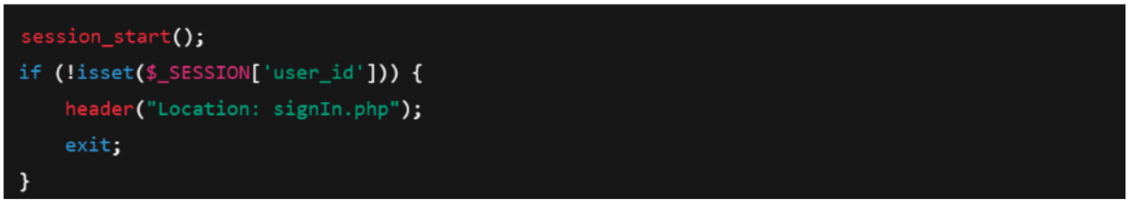
```
if (!$user2) {
    die("User does not exist.");
}
```

6. Authentication Check Missing

Issue:

Script runs without verifying user authentication.

Recommended Fix:



```
session_start();
if (!isset($_SESSION['user_id'])) {
    header("Location: signIn.php");
    exit;
}
```

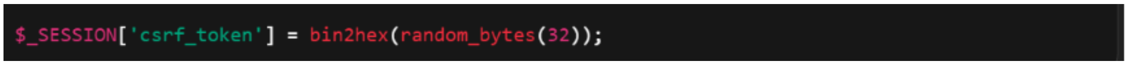
7. CSRF Token Protection Missing

Issue:

No CSRF protection is implemented for the POST request.

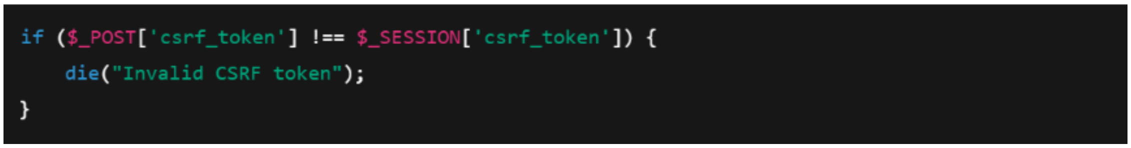
Fix (Frontend & Backend):

- **Token Generation:**



```
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));
```

- **Token Validation:**



```
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("Invalid CSRF token");
}
```

3 Test Design

3.1 Black-Box Testing

3.1.1 By Amro Al Deek

black-box test cases (EP, BVA, etc.) applied.

3.1.2 By Saleh Al Deek

black-box test cases (EP, BVA, etc.) applied.

3.1.3 By Nagham Nazeeh

black-box test cases (EP, BVA, etc.) applied.

3.1.4 By Julia Daibes

black-box test cases (EP, BVA, etc.) applied.

3.2 White-Box Testing

3.2.1 By Amro Al Deek

White-box tests with path/statement/branch coverage .

3.2.2 By Saleh Al Deek

White-box tests with path/statement/branch coverage .

3.2.3 By Nagham Nazeeh

White-box tests with path/statement/branch coverage .

3.2.4 By Julia Daibes

White-box tests with path/statement/branch coverage .

4 Automated Testing with Selenium

4.0.1 By Amro Al Deek

3 automated test cases with Katalon Extension.

4.0.2 By Saleh Al Deek

3 automated test cases with Katalon Extension.

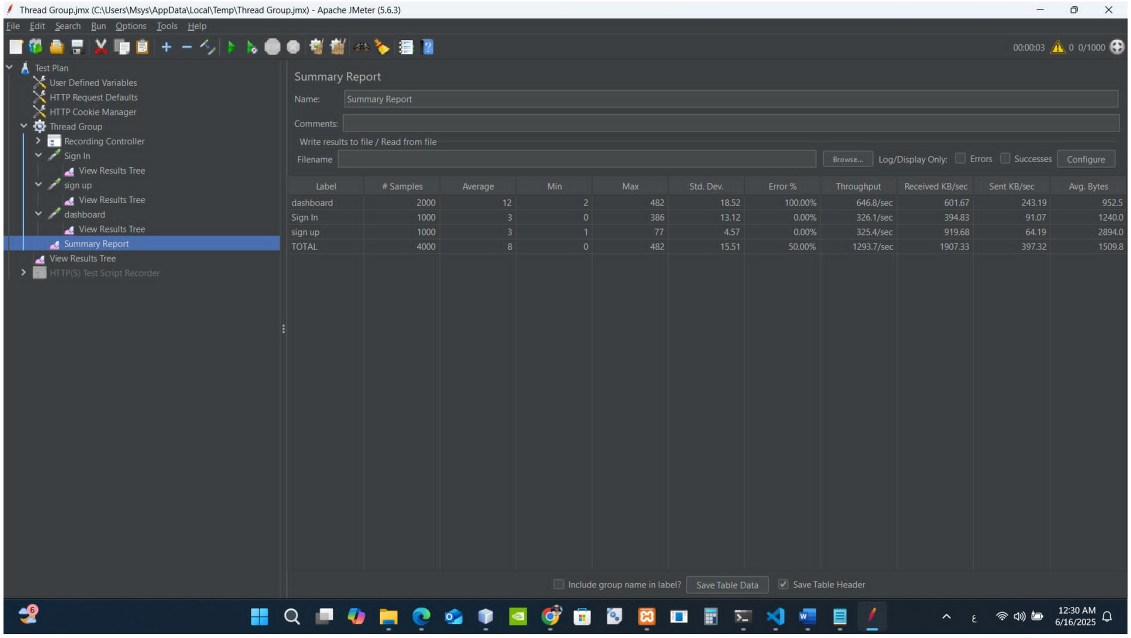
4.0.3 By Nagham Nazeeh

3 automated test cases with Katalon Extension.

4.0.4 By Julia Daibes

3 automated test cases with Katalon Extension.

5 Performance and Load Testing (JMeter)



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
dashboard	2000	12	2	482	18.52	100.00%	646.8/sec	601.67	243.19	952.5
Sign In	1000	3	0	386	13.12	0.00%	326.1/sec	394.83	91.07	1240.0
sign up	1000	3	1	77	4.57	0.00%	325.4/sec	919.68	64.19	2894.0
TOTAL	4000	8	0	482	15.51	50.00%	1293.7/sec	1907.33	397.32	1509.8

Figure: JMeter Load Test Summary Report

We ran a performance test using Apache JMeter to simulate a load of 1,000 users within a short period of 3 seconds, testing three main functions of the system: **Sign In**, **Sign Up**, and **Dashboard**. The performance summary report shows the following results:

Request Distribution:

- 2,000 requests for Dashboard access
- 1,000 requests for Sign In
- 1,000 requests for Sign Up

Key Findings from the Report:

- **Average Response Time:**
  - Dashboard: 12 ms
  - Sign In and Sign Up: only 3 ms

This indicates good speed in executing requests, with a relative delay in the dashboard.

- **Min/Max Response Time:**
  - Minimum: 0 ms
  - Maximum: 482 ms (for the dashboard)

  This variation indicates some fluctuation in performance under high load.

- **Standard Deviation:**
  - Dashboard: 18.52 ms
  - Sign In and Sign Up: very low (indicating consistent performance)

- **Error Rate:**
  - Dashboard: 100% (all requests failed)
  - Sign In and Sign Up: 0% (no failures)

  This suggests a critical issue with the dashboard functionality under load—possibly a misconfiguration or bug.

- **Throughput (requests/sec):**
  - Dashboard: 646.8 requests/sec
  - Sign In and Sign Up: approximately 326 requests/sec each

  Indicates that dashboard received the majority of traffic and executed at high frequency.

- **Data Transfer Rate (KB/sec):**
  - Highest observed on Sign Up requests

Conclusion:
Sign In and Sign Up endpoints performed exceptionally well in terms of speed, reliability, and stability under high load. However, the Dashboard endpoint demonstrated a 100% error rate, despite having the highest throughput, suggesting a failure under stress. This may be due to a server crash, session handling error, or incorrect request configuration in JMeter. Further debugging is necessary. It is recommended to:

- Optimize the dashboard page and its backend logic.
- Improve error handling.
- Add stress-specific monitoring/logging for deeper diagnostics.

6 Test Management (Jira)

6.1 Traceability Matrix

See the Traceability Matrix here

7 Regression Testing

1. Understanding Regression Testing

Regression testing is the quick examination we perform after any change to the software—bug fix, new feature, or cleanup—to ensure that anything that previously worked still works.

Basically, the goal here is to find broken pieces before the users notice, which means checking that old features still behave as they always did, even after changes to the code.

2. Summary of Code Changes

While building the PHP tool `Task Allocator Pro`, we peeled the business rules away from form handling in the `Create Task` ,`add Project`, `check Sign In` features so unit tests could run independently.

Before that tidy-up, the forms worked fine, yet after the logic split, a few controls stopped reacting as expected. Separating the code unintentionally broke some links between the rules and the interface, leaving fields that once passed input checks out of sync.

3. Re-running Previous Test Cases

Once the separation errors were patched, we reran the old test cases for the task creation form. The checks were:

- Does the form load and show all elements correctly?
- Are inputs validated as intended?
- Can a new task be saved when every field is filled in?

Result:

- A handful of test cases failed as a result of the refactoring.

- Specifically, some of the fields were no longer rendering, or data was not being submitted, due to the linkage being broken between the front end and the now independently separated back-end logic.

4. Impact Analysis and Lessons Learned

Impact Analysis:

- The refactoring caused an undesired side effect by breaking the connection of the frontend (HTML/PHP form) to the backend, resulting in undefined or unlinked methods and variables from separation of code.
- We did not re-verify the integration of components after the changes were made, which resulted in regression.

Lessons Learned:

- Regression testing is required after every code change, no matter how trivial or ”safe” it may appear.
- Code refactoring must be followed up with integration testing to know that the components communicate correctly with each other.
- Unit testing is insufficient when incorporating separation of concerns; the whole system behavior must be re-evaluated.
- Maintain simultaneous coverage of both legacy and new functionality when testing in order to maintain confidence in system stability.

5. Unit Testing Evidence and Coverage

The following PHPUnit test cases were created to confirm the core functionalities after refactoring:

1. **SignInTest.php**
    - Tests user authentication against a mock database.
    - Confirms successful login with valid credentials.
    - Confirms failure with invalid credentials.
2. **ProjectDateTest.php**
    - Tests the logic of date validation for project timelines.

- Confirms that same-day and reversed date inputs are rejected.

3. **AddProjectTest.php**

   - Tests inserting a new project into an SQLite memory database.
   - Confirms insertion success and detects duplicate project IDs.

Each test case passed individually in isolation. However, only when retesting the full application did integration bugs emerge, confirming that unit tests alone are not sufficient to prevent regressions in the full stack.

6. Testing Strategy Adjustment

The experience highlighted a need to adopt a layered testing approach. In future iterations, the following methodology will be followed:

- **Unit Tests:** First ensure individual components (like input validation, date parsing, etc.) work after refactoring.
- **Integration Tests:** Next, verify that the frontend elements correctly link with backend logic.
- **Regression Suite:** Re-run a regression suite containing all previously working features to confirm nothing regresses.
- **Smoke Tests:** Perform a final high-level pass to ensure overall application integrity.

This layered strategy would prevent the same issue from recurring and improve confidence in every release.

8 Test Plan Document Summary

8.1 Risk Analysis

- **Late Deliverables:** Due to lack of prior testable functions, some code had to be refactored during testing.
- **Tool Setup Issues:** Initial problems installing PHPUnit and Selenium slowed early automation tasks.
- **Incomplete Test Data:** Some features required realistic user input or uploaded files, making automation complex.
- **Security Gaps:** Static testing revealed unescaped outputs and missing CSRF protection which posed potential security risks.

8.2 Strategy Overview

Testing Types Applied:

- **Static Testing:** Manual code reviews to catch logic and security flaws early (e.g., upload handler).
- **Unit Testing:** PHPUnit tests for functions like `insertProject()` and `authenticateUser()`.
- **Black-Box Testing:** Equivalence Partitioning, Boundary Value Analysis for form fields and logic.
- **White-Box Testing:** Coverage-based tests on internal logic and conditionals.
- **Automated Testing:** Selenium scripts for login, logout, dashboard access, and invalid inputs.
- **Performance Testing:** Planned using JMeter for form submission and login stress testing.
- **Regression Testing:** Manual and unit tests rerun after security changes to verify nothing broke.

Tools Used:

- **PHPUnit** — for automated unit tests.
- **Katalon Recorder -Selenium tests generator** — for functional UI tests.
- **Jira (AIO Test)** — for test planning, execution tracking, and issue reporting.
- **JMeter** — for load and performance simulation.
- **XAMPP** — Used as the local server environment for running PHP and MySQL, enabling backend logic and database operations.
- **LaTeX** — for writing the documentation and formal report.

Test Schedule:

- **Week 1–2:** Requirement analysis and team task distribution.
- **Week 3:** Static testing and first round of unit tests.
- **Week 4:** Black-box and white-box test case design.
- **Week 5:** Automation setup (PHPUnit and Selenium).
- **Week 6:** Execute automated tests + performance testing with JMeter.
- **Week 7:** Regression testing + report finalization + Jira linking.

8.3 Traceability Summary

The following table outlines the mapping between requirements and test coverage:

Requirement	**Test Type**	**Artifact/Tool**
User Authentication	Unit Testing, Katalon	`authenticateUser()` test+login Katalon script
Project Upload Validation	Static + Unit Testing	`saveProject.php` code Inspection, PHPUnit
Form Field Validations	Black-Box Testing	EP , BVA and Decision tables test cases (Jira)
Upload Limits + File Types	White-Box Testing	Test coverage screenshots + assertions
Session Management	Regression Testing	Manual re-check after security fixes
CSRF Protection	Static Review + Strategy	Added verification + image proof
Performance Handling	JMeter Testing	JMeter test plan + response time chart

9 Conclusion

This report showcases an in-depth analysis of the project requirements and the traceability of those requirements to make sure the system operates as intended, is within the set standards, and is up to quality. The addition of user stories emphasizes the user's viewpoint and attempts to fulfill their needs as well as address the technical aspects of the project. Useful traceability enhances overall measure of efficiency when managing the project, testing, and maintenance in the future.

This report provides a comprehensive documentation of the Task Management System, covering its functional and non-functional requirements, user stories, and traceability matrix. The goal is to ensure that all project objectives are met, and the system performs efficiently and securely. The inclusion of detailed requirements and traceability aids in future development, testing, and maintenance.

Special thanks to Dr. Samer Zain for his great effort and guidance throughout the course.