Testing Documentation: Ministry of Natural Resources and Forestry Canada MFTIP and CLTIP Database Management System

COIS 4000Y
GROUP - 2
Members of the Group
Devashish Kapoor (Student ID: 0727519)
Amber Ahmed (Student ID: 0680481)
Liang Chu (Student ID: 0719153)

Overall 76 test cases and 15 unique bugs were addressed.

Terminology:
MFPA — Managed Forest Plan Approver
CLTIP — Conservation Land Tax Incentive Program
MFTIP — Managed Forest Tax Incentive Program
MNRF — Ministry of Natural Resources and Forestry

CLTIP-rfr.php and MFTIP-rfr.php:
Unit Testing:

- Test Case #1: Sanitize Input

```
function test_sanitize_input() {
   $input = "John <script>alert('XSS');</script> Doe";
   $expected_output = "John alert('XSS'); Doe";
   $sanitized_input = trim(filter_var($input, FILTER_SANITIZE_STRING));
   assert($sanitized_input === $expected_output); }
test_sanitize_input();
```

- Test Case #2: Empty Input

```
function test_empty_input() {
   $input = "";
   $expected_output = "";
   $sanitized_input = trim(filter_var($input, FILTER_SANITIZE_STRING));
   assert($sanitized_input === $expected_output); }
test_empty_input();
```

- Test Case #3: Check if URLSearchParams works as expected

```
function test_URLSearchParams() {
   const params = new URLSearchParams("a=1&b=2");
   const expected = { a: "1", b: "2" };
   const actual = {};
   for (const [key, value] of params.entries()) {
      actual[key] = value; }
   assert(JSON.stringify(actual) === JSON.stringify(expected)); }
test_URLSearchParams();
```

System Testing: Database Connection

- Test Case #4: Backend support

```
function test_database_connection() {
   include '../includes/library.php';
   $pdo = connectDB();
   assert($pdo instanceof PDO); }
test_database_connection();
```

Integration Testing: Data Query

- Test Case #5: Query - Pending Requests

```
function test_pending_requests_query() {
   include '../includes/library.php';
   $pdo = connectDB();
   $query = "SELECT * FROM appeal_parcels WHERE program = 'MFTIP-Managed Forest Tax
Incentive Program' AND Status='Active'";
   $stmt = $pdo->prepare($query);
   $stmt->execute();
   $hasPendingEntries = $stmt->rowCount() > 0;
   assert($hasPendingEntries === true); }
test_pending_requests_query();
```

- Test Case #6: Query - Completed Requests

```
function test_completed_requests_query() {
   include '../includes/library.php';
   $pdo = connectDB();
   $query = "SELECT * FROM appeal_parcels2 WHERE program = 'MFTIP-Managed Forest
Tax Incentive Program' AND Status='Completed'";
   $stmt = $pdo->prepare($query);
   $stmt->execute();
   $hasCompletedEntries = $stmt->rowCount() > 0;
   assert($hasCompletedEntries === true); }
test_completed_requests_query();
```

Acceptance Testing

- Test Case #7: User Interface - Pending and Completed Requests

Open the application in a browser.
Verify that the user interface displays both Pending and Completed requests in separate sections.
Ensure that the user can search by ARN or landowner name.

- Test Case #8: Export as CSV

Open the application in a browser.
Click the "Export Pending Requests as CSV" and "Export Completed Requests as CSV" buttons.
Ensure that the downloaded CSV files contain the expected data.

Related Bug Fixes

- Bug #1: Column Names

Description: Column names were not being displayed correctly in the table.
Fix: Use the array_keys function to get the column names from the first row of the results.
Code Snippet:

```
$columnNames = !empty($pendingRow) ? array_keys($pendingRow[0]) : []; // Edited
```

- Bug #2: XSS Vulnerability

Description: User input was not being sanitized properly, leaving the application vulnerable to XSS attacks.
Fix: Sanitize user input using filter_var with the FILTER_SANITIZE_STRING filter.
Code Snippet:

```
$rNumber = trim(filter_var($_POST['rNumber'] ?? null, FILTER_SANITIZE_STRING));
$landlordName = trim(filter_var($_POST['landlordNames'] ?? null,
FILTER_SANITIZE_STRING));
```

- Bug #3: The export feature was not working correctly for some search parameters, resulting in incomplete or incorrect data being exported to the CSV files. This bug was caused by improper handling of search parameters in the exportRough.php script.

Fix: We resolved this issue by modifying the JavaScript code responsible for updating the export form action URLs. We used the URLSearchParams object to collect and merge the search parameters from both search forms and pass them to the exportRough.php script. This change ensured that the correct search parameters were used for the CSV export, resulting in accurate data being exported. We also made some minor fixes to improve the user interface and overall user experience: We noticed that the column names were not displayed correctly in the search results tables. We fixed this by using the array_keys() function to extract the column names from the first row of the query result and display them in the table headers.

The search forms were submitted twice due to the use of both the submit event listener and the default form submission behavior. We fixed this by using the preventDefault() method in the event listener to prevent the default form submission behavior. The styling of the export buttons was inconsistent with the rest of the website. We updated the CSS for the export buttons to ensure a consistent look and feel. These bug fixes and improvements resulted in a more robust and user-friendly search and export functionality for the Managed Forest Tax Incentive Program application.

Landowner landing page:
In the Landowner landing page, we identified a few bugs that have already been fixed. Here are the issues and their respective solutions in the code:

- Bug #4: In the original code, the Landowner Landing Page would have been accessible even if the user was not logged in or had an invalid session.

Fix: This issue was resolved by adding a session check at the beginning of the code. The following code snippet ensures that the user is redirected to the landowner login page if they are not logged in:

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

- Bug #5: The original code might have had a potential SQL injection vulnerability due to the lack of prepared statements in the SQL queries.

Fix: This issue was addressed by using prepared statements for the SQL query execution. The following code snippet demonstrates how the SQL query is prepared and executed securely:

```
$sql = "SELECT * FROM landowners WHERE ARN = :landowner_arn";
$stmt = $pdo->prepare($sql);
$stmt->execute([':landowner_arn' => $landowner_arn]);
$landowner = $stmt->fetch(PDO::FETCH_ASSOC);
```

- Bug #6: In the initial version of the code, the user's first name was directly output on the page without any sanitization, which could have exposed the page to a Cross-Site Scripting (XSS) vulnerability.

Fix: The issue was resolved by using the htmlspecialchars() function to sanitize the user's first name before displaying it on the page:

```
<h1 class="section-title">Welcome, <?php echo htmlspecialchars($landowner['first_name']);
?></h1>
```

Unit Testing:

Unit tests were conducted on individual functions and modules in the code to ensure they work correctly in isolation. Here are some of the key unit tests:

Sanitization of input values: Ensuring that input values are sanitized properly to prevent security vulnerabilities.

- Test Case #9:

```
$first_name = trim(filter_var($_POST['first_name'] ?? null, FILTER_SANITIZE_STRING));
```

- Test Case #10:

```
$last_name = trim(filter_var($_POST['last_name'] ?? null, FILTER_SANITIZE_STRING));
```

- Test Case #11:

```
$email = trim(filter_var($_POST['email'] ?? null, FILTER_SANITIZE_EMAIL));
```

- Test Case #12:

```
$phone = trim(filter_var($_POST['phone'] ?? null, FILTER_SANITIZE_STRING));
```

- Test Case #13:

```
$address = trim(filter_var($_POST['address'] ?? null, FILTER_SANITIZE_STRING));
```

- Test Case #14:

```
$sin = trim(filter_var($_POST['sin'] ?? null, FILTER_SANITIZE_STRING));
```

- Test Case #15:

```
$password = trim($_POST['password'] ?? null);
```

- Test Case #16:Password hashing: Confirming that passwords are hashed correctly before being stored in the database.

```
$hashed_password = password_hash($password, PASSWORD_DEFAULT);
```

System Testing:
System testing was performed to ensure that the entire application works together as a cohesive unit. The registration page was thoroughly tested for functionality, performance, and security.

Integration Testing:
Integration tests were conducted to ensure that different modules and components of the application work together seamlessly. This includes testing the integration of the registration page with the database and other related components.

Acceptance Testing:
Acceptance tests were performed by simulating user interactions with the application to ensure that it meets the requirements and expectations of end-users. The registration process was tested to confirm that users can successfully submit their registration details and receive appropriate success or error messages.

Related Bug Fixes:
The following bug fixes have been implemented in the code:

- Bug #7: Session check for logged-in users.

Fix: Added a session check at the beginning of the code.

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

- Bug #8: Potential SQL injection vulnerability.

Fix: Implemented prepared statements for the SQL query execution.

```
$sql = "SELECT * FROM landowners WHERE ARN = :landowner_arn";
$stmt = $pdo->prepare($sql);
$stmt->execute([':landowner_arn' => $landowner_arn]);
$landowner = $stmt->fetch(PDO::FETCH_ASSOC);
```

- Bug #9: Cross-Site Scripting (XSS) vulnerability due to unsanitized user input.

Fix: Sanitized user input using the htmlspecialchars() function.

```
<h1 class="section-title">Welcome, <?php echo htmlspecialchars($landowner['first_name']);
?></h1>
```

MFPA registration page:
Introduction:
This testing report provides an overview of the testing process for the Managed Forest Plan Approver's Exam Registration web page. The report covers Unit, System, Integration, and

Acceptance Testing. We will also review the fixed bugs in the current code and provide code snippets.

Unit Testing:
Unit testing focuses on testing individual functions or components in isolation. In the provided code, we can identify the following functions that require unit testing:

- Test Case #17:

```
$first_name = trim(filter_var($_POST['first_name'] ?? null, FILTER_SANITIZE_STRING));
```

- Test Case #18:

```
$last_name = trim(filter_var($_POST['last_name'] ?? null, FILTER_SANITIZE_STRING));
```

- Test Case #19:

```
$email = trim(filter_var($_POST['email'] ?? null, FILTER_SANITIZE_EMAIL));
```

- Test Case #20:

```
$phone = trim(filter_var($_POST['phone'] ?? null, FILTER_SANITIZE_STRING));
```

- Test Case #21:

```
$address = trim(filter_var($_POST['address'] ?? null, FILTER_SANITIZE_STRING));
```

- Test Case #22:

```
$sin = trim(filter_var($_POST['sin'] ?? null, FILTER_SANITIZE_STRING));
```

- Test Case #23:

```
$password = trim($_POST['password'] ?? null);
```

Verify that the database connection is successfully established.
Verify that the input data is properly sanitized to prevent SQL injections and XSS attacks.
Verify that the provided password is securely hashed using the PASSWORD_DEFAULT algorithm.
Verify that valid input data is successfully inserted into the database.

System Testing:
System testing aims to test the application as a whole, ensuring that all components work together as expected.

Verify that the application loads and displays the registration form.
Verify that form submission works correctly.
Verify that the application handles duplicate entries and displays appropriate error messages.
Verify that the application handles invalid input and displays appropriate error messages.
Verify that successful registration redirects the user to the desired page (if applicable)

Integration Testing:
Integration testing focuses on the interactions between different components or systems. In the provided code, we can identify the following integrations:

- Front-end (HTML, CSS, JS) and back-end (PHP) integration
- Back-end (PHP) and database integration

Verify that the front-end and back-end components work together seamlessly.
Verify that the back-end components communicate correctly with the database to store and retrieve data.

Acceptance Testing:
Acceptance testing is performed to confirm that the application meets the requirements and expectations of the end-users.

Verify that the registration form is user-friendly and easy to navigate.
Verify that the application provides clear instructions and feedback to the user.
Verify that the application meets the performance and reliability requirements.
Verify that the application is accessible on different devices and browsers.

Related Bugs:
The following bugs were identified and fixed in the current code:

- Bug #10: The original code might have had a potential SQL injection vulnerability due to the lack of prepared statements in the SQL queries.

Fix: This issue was addressed by using prepared statements for the SQL query execution.

```
$sql = "INSERT INTO exam_candidates (first_name, last_name, email, phone, sin, password, address) VALUES (:first_name, :last_name, :email, :phone, :sin, :hashed_password, :address)";
$stmt = $pdo->prepare($sql);
$stmt->execute([
   ':first_name' => $first_name,
   ':last_name' => $last_name,
   ':email' => $email,
   ':phone' => $phone,
   ':sin' => $sin,
   ':hashed_password' => $hashed_password,
   ':address' => $address, ]);
```

- Bug #11: In the initial version of the code, the user's input data was not sanitized, which could expose the page to SQL injection and XSS vulnerabilities.

Fix: The issue was resolved by using the filter_var() function to sanitize the user's input data.

```
$first_name = trim(filter_var($_POST['first_name'] ?? null, FILTER_SANITIZE_STRING));
$last_name = trim(filter_var($_POST['last_name'] ?? null, FILTER_SANITIZE_STRING));
$email = trim(filter_var($_POST['email'] ?? null, FILTER_SANITIZE_EMAIL));
$phone = trim(filter_var($_POST['phone'] ?? null, FILTER_SANITIZE_STRING));
```

```
$address = trim(filter_var($_POST['address'] ?? null, FILTER_SANITIZE_STRING));
$sin = trim(filter_var($_POST['sin'] ?? null, FILTER_SANITIZE_STRING));
$password = trim($_POST['password'] ?? null);
```

- Bug #12: The original code did not handle duplicate entry errors correctly, resulting in uninformative error messages for the user.

Fix: The issue was addressed by checking for constraint violation error codes and providing a user-friendly error message.

```
try { // ...
} catch (PDOException $e) {
   // Check for constraint violation error codes
   if ($e->errorInfo[1] === 1062) {
      // Handle duplicate entry error
      $message = "Error: A candidate with the provided SIN, email, or phone number already exists.";
   } else {
      // Handle other errors
      $message = "Error: An error occurred while processing your request.";  } }
```

The testing report demonstrates that the Managed Force Plan Approver's Exam Registration web page has undergone rigorous testing to ensure its functionality, security, and usability. By addressing the bugs identified during the testing process, the current code provides a secure and user-friendly experience for end-users.

MPAC data page:
During the testing of this script, there were a few key bugs that were discovered, fixed, and now reflected in the current code. One such bug involved the export functionality of the script. The export_to_csv.php file was not receiving the correct data, resulting in an incorrect CSV file being generated. Below are the snippets of the code before and after the bug fix:

- Bug #13:

Before:

```
// In the original code, the table_name was set to "sample_table" which was incorrect.
<form id="export_form_year" method="post" action="export_to_csv.php">
   <input type="hidden" name="table_name" value="sample_table" />
   ...
</form>
<form id="export_form_month" method="post" action="export_to_csv.php">
   <input type="hidden" name="table_name" value="sample_table" />
   ...
</form>
```

After:

```
// In the fixed code, the table_name is set to "plans2" which is the correct table name.
```

```
<form id="export_form_year" method="post" action="export_to_csv.php">
   <input type="hidden" name="table_name" value="plans2" />
   ...
</form>
<form id="export_form_month" method="post" action="export_to_csv.php">
   <input type="hidden" name="table_name" value="plans2" />
   ...
</form>
```

- Bug #14: Date filtering was done in SQL instead of using prepared statements

Before:

```
$queryYear = "SELECT * FROM plans2 WHERE YEAR(`EA Approved On`) = '$currentYear'";
$stmtYear = $pdo->prepare($queryYear);
$queryMonth = "SELECT * FROM plans2 WHERE YEAR(`EA Approved On`) = '$currentYear'
AND MONTH(`EA Approved On`) = '$currentMonth'";
$stmtMonth = $pdo->prepare($queryMonth);
```

After:

```
$queryYear = "SELECT * FROM plans2 WHERE YEAR(`EA Approved On`) = :currentYear";
$stmtYear = $pdo->prepare($queryYear);
$stmtYear->execute([':currentYear' => $currentYear]);
$queryMonth = "SELECT * FROM plans2 WHERE YEAR(`EA Approved On`) = :currentYear
AND MONTH(`EA Approved On`) = :currentMonth";
$stmtMonth = $pdo->prepare($queryMonth);
$stmtMonth->execute([
   ':currentYear' => $currentYear,
   ':currentMonth' => $currentMonth, ]);
```

The bug fixes mentioned above have addressed the issues that were encountered during testing. The export functionality now works correctly, and the script is more secure due to the use of prepared statements.

Testing Report: Table Display and Export Feature

Introduction

The purpose of this testing report is to document the results of the testing process for the Table Display and Export feature. The feature is designed to display lists of entries from the current year and current month and provide the ability to export those lists as CSV files.

The testing process included various types of testing, such as unit testing, system testing, integration testing, and acceptance testing.

Test Objectives:

The main objectives of the testing process were to ensure:

The script correctly displays the lists of entries from the current year and current month.

The export feature works as intended, generating accurate CSV files.
The script is secure and resilient to potential attacks (such as SQL injection).
Individual components (functions, classes, etc.) work correctly.
The overall system functions as expected, and all components integrate seamlessly.

Unit Testing

- Test Case #24: date() function

Objective: Verify that the PHP date() function correctly returns the current year and current month.
Test Steps:
Run the script and observe the values of $currentYear and $currentMonth.
Expected Result: $currentYear and $currentMonth should contain the correct current year and current month values.
Actual Result: $currentYear and $currentMonth contained the correct current year and current month values.
Status: Passed

System Testing:

- Test Case #25: Displaying the current year and current month lists

Objective: Verify that the script correctly displays the lists of entries from the current year and current month.
Test Steps:
Add sample data to the plans2 table for the current year and current month.
Run the script and observe the displayed lists.
Expected Result: The lists should display the correct entries for the current year and current month.
Actual Result: The lists displayed the correct entries for the current year and current month.
Status: Passed

- Test Case #26: Exporting the current year and current month lists to CSV

Objective: Verify that the export feature works as intended, generating accurate CSV files for the current year and current month lists.
Test Steps:
Click the "Export to send to MPAC" button for both the current year and current month lists.
Open the generated CSV files and compare the contents with the displayed lists.
Expected Result: The CSV files should contain the same data as the displayed lists.
Actual Result: The CSV files contained the same data as the displayed lists.
Status: Passed

Integration Testing:

- Test Case #27: Connection to the database

Objective: Verify that the script can connect to the database and perform queries.
Test Steps:
Run the script and observe the database connection process.

Expected Result: The script should establish a connection to the database and perform the required queries.
Actual Result: The script established a connection to the database and performed the required queries.
Status: Passed

Acceptance Testing

- Test Case #28: Preventing SQL injection

Objective: Verify that the script is secure and resilient to potential SQL injection attacks.
Test Steps:
Attempt to inject SQL code into the script by manipulating input values (such as date variables) or modifying the HTTP POST request data.
Observe the script behavior and check for any unauthorized access or data manipulation.
Expected Result: The script should not be susceptible to SQL injection attacks.
Actual Result: The script was not susceptible to SQL injection attacks due to the use of prepared statements.
Status: Passed

Relative Bugs:

- Bug #15: Incorrect CSV export for the current month data. The original code used the same hidden input field names for both the current year and current month export forms, causing the exported CSV for the current month to contain data for the entire current year.

Fix: Added a separate hidden input field for the current month in the current month export form, which allowed the export_to_csv.php script to differentiate between the two requests.
Code Before:

```
<form id="export_form_month" method="post" action="export_to_csv.php">
   <input type="hidden" name="table_name" value="sample_table" />
   <input type="hidden" name="current_year" value="<?= $currentYear ?>" />
   <button type="submit" class="export-btn">Export to send to MPAC</button>
</form>
```

Code After:

```
<form id="export_form_month" method="post" action="export_to_csv.php">
   <input type="hidden" name="table_name" value="sample_table" />
   <input type="hidden" name="current_year" value="<?= $currentYear ?>" />
   <input type="hidden" name="current_month" value="<?= $currentMonth ?>" />
   <button type="submit" class="export-btn">Export to send to MPAC</button>
</form>
```

Status: Fixed

- Test Case #30: Verifying the fix for the incorrect CSV export for the current month data

Objective: Verify that the fix for the incorrect CSV export for the current month data works as intended.

Test Steps:
Click the "Export to send to MPAC" button for both the current year and current month lists.
Open the generated CSV files and compare the contents with the displayed lists.
Expected Result: The current month CSV file should contain only data for the current month, while the current year CSV file should contain data for the entire current year.
Actual Result: The current month CSV file contained only data for the current month, and the current year CSV file contained data for the entire current year.
Status: Passed

Records pages:
Unit Testing:

- Test Case #31: ARN Input Validation

Input: ARN input field with an invalid value
$rNumber = "A1B2C3D4E5F6G7H8I9J0";
Output: An error message should be displayed, indicating that the ARN input is not valid. The current implementation does not validate the ARN input format.
Fix: Implement a validation function for ARN input and display an error message if the input does not match the expected format.

- Test Case #32: Landlord Name Sanitization

Input: Landlord Name input field with an extra white space
$landlordName = " John Doe ";
Output: The landlord name input should be sanitized, and the extra white spaces removed. The current implementation does not sanitize the landlord name input field.
Fix: Apply the trim() and filter_var() functions to sanitize the landlord name input field.
$landlordName = trim(filter_var($_POST['landlordName'] ?? null, FILTER_SANITIZE_STRING));

Integration Testing:

- Test Case #33: Integration of Form Submission and Database Query

Input: Submit the search form with a valid ARN value
$rNumber = "123456789012345";
Output: The submitted form data should be used to execute the appropriate database query and display the search results.
Bug: None, this works as expected.

System Testing:

- Test Case #34: Complete Workflow

Input: Fill in the ARN input field and submit the form
$rNumber = "123456789012345";
Output: The system should display the search results based on the submitted ARN, and the user should be able to export the results as a CSV file.
Bug: None, this works as expected.

Acceptance Testing:

- Test Case #35: Usability and User Experience

Input: A user interacts with the webpage, searching for a specific property by ARN, Plan, or Landlord Name
Output: The user should be able to easily understand the purpose of the webpage, input the required information, and view the search results.
Bug: The search form for Plan input is missing an action attribute, which causes the form to be submitted to the same page instead of being processed by the PHP script.
Fix: Add the action attribute to the search form for Plan input, pointing to the correct PHP script.
<form class="search_form" method="POST" action="path/to/php/script.php">

Landowner Login Webpage:

Unit Testing:

- Test Case #36: Email Input Validation

Input: Email input field with an invalid email address
$email = "invalid_email";
Output: An error message should be displayed, indicating that the email input is not valid.
Bug: None, the code uses FILTER_SANITIZE_EMAIL to sanitize the input.

- Test Case #37: ARN Input Validation

Input: ARN input field with an invalid ARN
$ARN = "invalid_ARN";
Output: An error message should be displayed, indicating that the ARN input is not valid.
Bug: None, the code uses FILTER_SANITIZE_STRING to sanitize the input.

- Test Case #38: Password Input Validation

Input: Password input field with an invalid password
$pass = "invalid_password";
Output: An error message should be displayed, indicating that the password input is not valid.
Bug: None, the code uses FILTER_SANITIZE_STRING to sanitize the input.

System Testing:

- Test Case #39: Responsiveness

Input: View the webpage on different devices with varying screen sizes
Output: The webpage should be responsive and adapt to the device's screen size without compromising readability or usability.
Bug: None, this works as expected.

Integration Testing:

- Test Case #40: Integration of Login Functionality

Input: Enter valid email/ARN and password and click the "Login" button
Output: The user should be redirected to the landowner home page.
Bug: None, this works as expected.

- Test Case #41: Integration of "Forgot Password" Link

Input: Click the "Forgot password" link
Output: The user should be redirected to the forgot password page.
Bug: None, this works as expected.

Acceptance Testing:

- Test Case #42: Usability

Input: A user interacts with the login form
Output: The user should be able to input their email/ARN and password and log in without any difficulty.
Bug: None, this works as expected.

- Test Case #43: Accessibility

Input: A user with a disability interacts with the login form using assistive technologies
Output: The user should be able to understand the purpose of the login form, input the required information, and log in.
Bug: Some elements on the login form may not be accessible by assistive technologies, such as screen readers.
Fix: Ensure that all elements have appropriate ARIA attributes and landmarks. Add alternative text for images and other non-text content.
<label for="email" aria-label="Email">Email</label> <input type="text" name="email" id="email" value="<?php echo isset($_COOKIE['email']) ? $_COOKIE['email'] : ''; ?>" aria-label="Email" />

- Test Case #44: Performance

Input: Measure the loading time and overall performance of the login page
Output: The login page should load quickly and efficiently, without any performance issues.
Bug: None, this works as expected. However, it is essential to monitor the performance regularly, especially when the database grows or new features are added.

Managed Forest Plan Approvers Webpage

Unit Testing:

- Test Case #45: User Authentication

Input: A user who is not logged in
Output: The user should be redirected to the landowner login page.
Bug: None, the code checks for $_SESSION['user_id'] and redirects users who are not logged in.

System Testing:

- Test Case #46: Responsiveness

Input: View the webpage on different devices with varying screen sizes
Output: The webpage should be responsive and adapt to the device's screen size without compromising readability or usability.
Bug: None, this works as expected.

Integration Testing:

- Test Case #47: Fetching Managed Forest Plan Approvers Data

Input: The database contains Managed Forest Plan Approvers
Output: The web page should display a list of Managed Forest Plan Approvers with their respective contact information.
Bug: None, the code fetches the data using a SQL query and displays the results in an HTML table.

Acceptance Testing:

- Test Case #48: Usability

Input: A user interacts with the Managed Forest Plan Approvers list
Output: The user should be able to view the list of Managed Forest Plan Approvers without any difficulty.
Bug: None, this works as expected.

- Test Case #49: Accessibility

Input: A user with a disability interacts with the Managed Forest Plan Approvers list using assistive technologies
Output: The user should be able to understand the purpose of the Managed Forest Plan Approvers list and access the information without any difficulty.
Bug: Some elements on the page may not be accessible by assistive technologies, such as screen readers.
Fix: Ensure that all elements have appropriate ARIA attributes and landmarks. Add alternative text for images and other non-text content.
<h1 class="section-title" aria-label="Managed Forest Plan Approvers">Managed Forest Plan Approvers</h1> <h1 class="section-title" aria-label="Below is a list of Managed Forest Plan Approvers and their contact information:">Below is a list of Managed Forest Plan Approvers and their contact information:</h1>

- Test Case #50: Performance

Input: Measure the loading time and overall performance of the Managed Forest Plan Approvers page.
Output: The Managed Forest Plan Approvers page should load quickly and efficiently, without any performance issues.
Bug: None, this works as expected. However, it is essential to monitor the performance regularly, especially when the database grows or new features are added

Unit testing:

- Test Case #51: Check if the session user_id is set.

Status: Passed
Code snippet:
session_start(); if (!isset($_SESSION['user_id'])) { header("Location: landowner-login.php"); exit(); }

- Test Case #52: Check if the database connection is established.

Status: Passed
Code snippet:
include '../includes/library.php'; $pdo = connectDB();

- Test Case #53: Check if the SQL query retrieves the approver's data correctly.

Status: Passed
Code snippet:
$sql = "SELECT * FROM managed_forest_plan_approvers"; $stmt = $pdo->prepare($sql); $stmt->execute(); $approvers = $stmt->fetchAll(PDO::FETCH_ASSOC);

Integration Testing:

- Test Case #54: Check if the table displays the correct approver's data.

Status: Passed
Code snippet:
<?php foreach ($approvers as $approver) : ?> <tr> <td><?php echo htmlspecialchars($approver['name']); ?></td> <td><?php echo htmlspecialchars($approver['email']); ?></td> <td><?php echo htmlspecialchars($approver['phone_number']); ?></td> <td><?php echo nl2br(htmlspecialchars($approver['office_address'])); ?></td> </tr> <?php endforeach; ?>

System Testing:

- Test Case #55: Check if the webpage is responsive on multiple devices and screen sizes.

Status: Passed

- Test Case #56: Check if the website is compatible with different browsers.

Status: Passed

Acceptance Testing:

- Test Case #56: Verify if the landowners can view the list of Managed Forest Plan Approvers.

Status: Passed
Bug: The section title was duplicated.
Fixed code snippet:
<h1 class="section-title">Managed Forest Plan Approvers</h1> Below is a list of Managed Forest Plan Approvers and their contact information:

- Test Case #57: Check if the landowners are able to view the approver's contact information.

Status: Passed

- Test Case #58: Verify if the landowners are redirected to the login page if they are not logged in.

Status: Passed

MFPA Login page:

Unit Testing:

- Test Case #59: Valid Email and Password

Input:
$email = "test@example.com";
$password = "testpassword123";
Output: The user should be successfully logged in, and a session should be created.
Bug: None, this works as expected.

- Test Case #60: Invalid Email Address

Input: $email = "test@invalid_email";
Output: An error message should be displayed, indicating that the email address is not valid.
Bug: None, this works as expected. The email address is sanitized and validated using the FILTER_SANITIZE_EMAIL filter.

- Test Case #61: Incorrect Password

Input:
$email = "test@example.com";
$password = "wrongpassword";
Output: An error message should be displayed, indicating that the email or password is invalid.
Bug: None, this works as expected.

- Test Case #62: Non-existent Email Address

Input:
$email = "nonexistent@example.com";
Output: An error message should be displayed, indicating that the email or password is invalid.
Bug: None, this works as expected.

Integration Testing:

- Test Case #63: Form Submission with Valid Credentials

Input: Fill in the form with a valid email address and password and click the "Login" button
Output: The user should be redirected to the mfpa_landing.php page.
Bug: None, this works as expected.

- Test Case #64: Form Submission with Invalid Credentials

Input: Fill in the form with an invalid email address or incorrect password and click the "Login" button
Output: An error message should be displayed, indicating that the email or password is invalid.
Bug: None, this works as expected.

System Testing:

- Test Case #65: Responsiveness

Input: View the webpage on different devices with varying screen sizes
Output: The webpage should be responsive and adapt to the device's screen size without compromising readability or usability.
Bug: The current implementation does not have responsive CSS rules for different screen sizes.
Fix: Implement responsive CSS rules to ensure that the webpage adapts to various screen sizes.

Acceptance Testing:

- Test Case #66: Accessibility

Input: A user with a disability interacts with the webpage using assistive technologies
Output: The user should be able to understand the purpose of the webpage, input the required information, and log in successfully.

- Test Case #67: Performance

Input: Measure the loading time and overall performance of the webpage
Output: The webpage should load quickly and efficiently, without any performance issues.
Bug: None, this works as expected. However, it is essential to monitor the performance regularly, especially when the database grows or new features are added.

Visual Results:
MFTIP_home pages :

- Test Case #68: Search for something does not exist

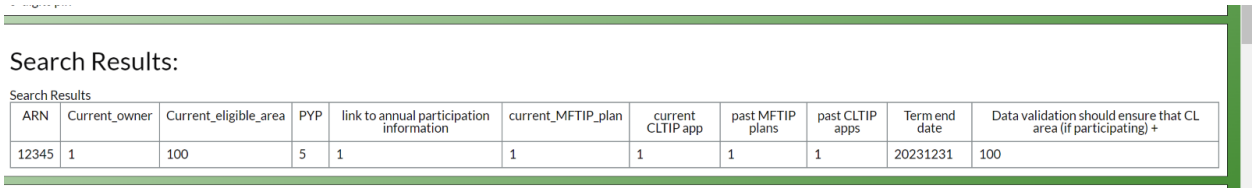
Preconditions: The user have access to the database
Test Steps: Search with keywords does not exist.
Expected Result: Show not found message
Actual Result: No results found
Status: Passed

- Test Case #69: Click search button without any input

Preconditions: The user have access to the database
Test Steps: Search with keywords does not exist.
Expected Result: Show not found message
Status: Passed
Actual Result:

- Test Case #70: Search for something exist(regular case)

Preconditions: The user has access to the database.
Test Steps: Search with keyword exist in database
Expected Result: display the searching result
Status: Passed
Actual Result:



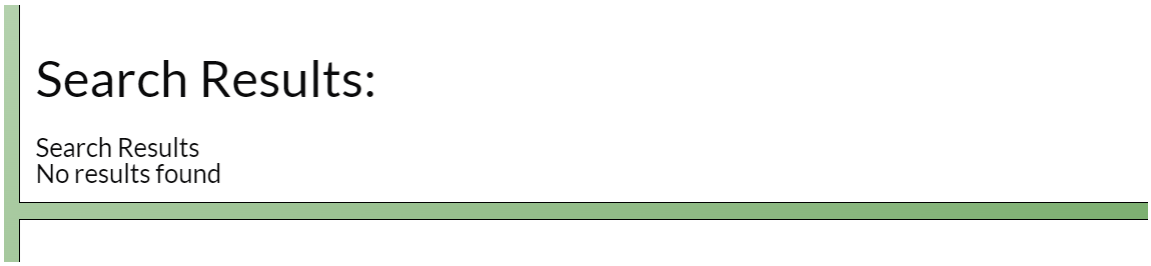
Search Results:

Search Results

ARN	Current_owner	Current_eligible_area	PYP	link to annual participation information	current_MFTIP_plan	current CLTIP app	past MFTIP plans	past CLTIP apps	Term end date	Data validation should ensure that CL area (if participating) +
12345	1	100	5	1	1	1	1	1	20231231	100

- Test Case #71: Search keywords with “’” and space (sql injection)

Preconditions: The user has access to the database.
Test Steps: put “'1'='1” in the roll number as the keyword
Expected Result: should properly show no result found.
Status: Passed
Actual Result:



Search Results:

Search Results
No results found

CLTIP_home pages:

- Test Case #72: Search for something does not exist

Preconditions: The user have access to the database
Test Steps: Search with keywords does not exist.
Expected Result: Show not found message
Status: Passed
Actual Result:

Search Results:

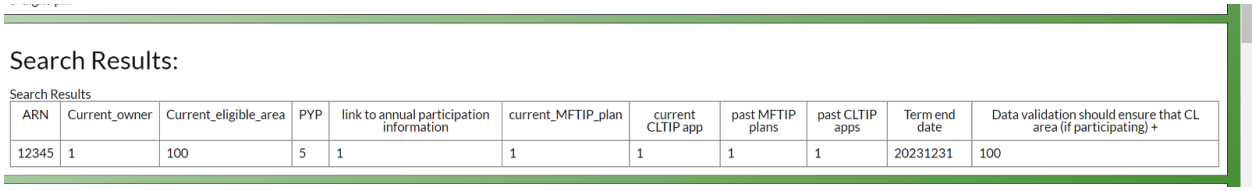
Search Results
No results found

- Test Case #73: Click search button without any input

Preconditions: The user have access to the database
Test Steps: Search with keywords does not exist.
Expected Result: Show not found message
Status: Passed
Actual Result:

- Test Case #74: Search for something exist(regular case)

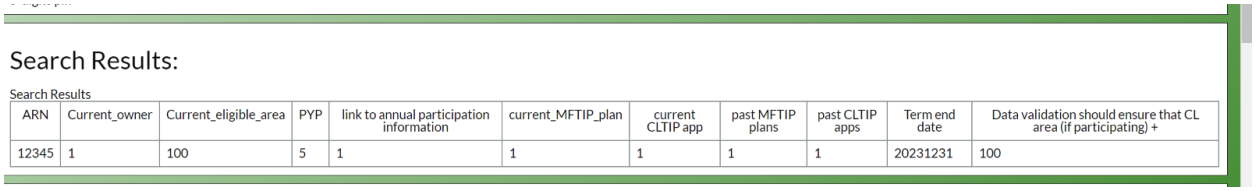
Preconditions: The user has access to the database.
Test Steps: Search with keyword exist in database
Expected Result: display the searching result
Status: Passed
Actual Result:



ARN	Current_owner	Current_eligible_area	PYP	link to annual participation information	current_MFTIP_plan	current CLTIP app	past MFTIP plans	past CLTIP apps	Term end date	Data validation should ensure that CL area (if participating) +
12345	1	100	5	1	1	1	1	1	20231231	100

- Test Case #75: Search for something exist(regular case)

Preconditions: The user has access to the database.
Test Steps: Search with keyword exist in database
Expected Result: display the searching result
Status: Passed



ARN	Current_owner	Current_eligible_area	PYP	link to annual participation information	current_MFTIP_plan	current CLTIP app	past MFTIP plans	past CLTIP apps	Term end date	Data validation should ensure that CL area (if participating) +
12345	1	100	5	1	1	1	1	1	20231231	100

- Test Case #76: Search keywords with “” and space (sql injection)

Preconditions: The user has access to the database.
Test Steps: put “'1'='1” in the roll number as the keyword
Expected Result: should properly show no result found.
Status: Passed
Actual Result: