A.1. Acceptance testing
Acceptance testing for the functional and non-functional requirements, for the project “Web Interface to Interact with Assets

Functional requirements:

Map Display with Asset Integration

Function	Description	Input data	Expected output	Justification	Test Result
Map display	Verify map loads with asset markers.	CSV with valid latitude/longitude data.	Map displays assets at correct locations.	Ensures accurate visualisation of asset data.	Passed
	Managing missing latitude/longitude values.	CSV with missing coordinates.	Error message: "Invalid asset location"	Validates system handles incomplete data gracefully.	Passed
	Dynamic map updates.	Updated CSV data.	Map updates with new asset locations.	Ensures real-time map reflects updated asset data.	Passed

Asset management

Function	Description	Input data	Expected output	Justification	Test Result
Add asset	Verify new asset addition	Asset data: name, category, location	Asset appears in asset list	Confirms the system can register new assets.	Passed
	Invalid asset input	Missing fields (e.g., no category)	Error message: "Missing required fields"	Ensures system handles and reports invalid inputs.	Passed
Edit asset	Ensure asset updates are saved	Edited asset information	Updated asset information displayed	Validates user modifications are properly saved.	Passed
Approve Modification	Admin approves asset changes	Asset update request	Status changes to "Approved"	Ensures administrative control over asset changes.	Passed
Delete Asset	Verify asset deletion.	Asset ID	Asset is removed from system	Confirms assets can be securely and permanently deleted.	Passed

Data integration

Function	Description	Input data	Expected output	Justification	Test Result
CSV upload	Validate asset data from CSV	CSV file with valid data	Data imported successfully	Verifies the system can extract and store asset data.	Passed
	Handle duplicate records	CSV with duplicate entries	Error message: “duplicate record”	Ensures data integrity by preventing duplicate records.	Passed
API integration	Ensure external data sync	API request with valid asset data	Database updates with new data	Validates compatibility with external systems.	Passed
Data validation	Validate asset input integrity	CSV with invalid fields (wrong format)	Error message: “invalid input format”	Ensures system only accepts valid data.	Passed

User Interface

Function	Description	Input data	Expected output	Justification	Test Result
Search	Verify asset search functionality	Asset name or category	Relevant asset displayed	Ensures efficient retrieval of asset information.	Passed
Invalid search	Direct incorrect search terms	Random invalid input	Message: “No results found”	Confirms system can handle invalid queries.	Passed
Accessibility check	Ensure compliance with WCAG 2.1	Screen reader navigation	Accessible and usable UI	Validates system meets accessibility standards.	Passed
Responsive design	Justify mobile responsiveness	Access system via mobile, tablet or desktop	Consistent UI across devices	Ensures usability across different devices.	Passed

Non-functional requirements

Performance & Scalability

Function	Description	Input data	Expected output	Justification	Test Result
Load speed	Evaluate map rendering time	CSV with 10,000 asset records	Map renders within 3 seconds	Ensures quick data display for large datasets.	Passed
System load	Supervise concurrent users	500 concurrent users accessing system	No performance degradation	Verifies system stability under heavy user load.	Passed

Security

Function	Description	Input data	Expected output	Justification	Test Result
Authentication	Ensure secure login	Valid username/password	Access granted	Approve user identity before granting access.	Passed
Unauthorised Access	Restrict access to sensitive areas	Invalid credentials	Error message: “Access denied”	Ensures restricted areas are secure.	Passed
Data encryption	Ensure secure data transmission	Simulate data interception	Data remains encrypted (AES-256, SSL/TLS)	Confirms compliance with industry encryption standards.	Passed

A.2: Unit Testing:

Unit Testing Justification Explanation

```php
require_once './includes/functions.php';

use PHPUnit\Framework\TestCase;

class UnitTests extends TestCase
{
    protected function setUp(): void
    {
        $_SESSION = [];
        $_SERVER = [];
    }

    // Test redirect function
    public function testRedirect()
    {
        ob_start();

        try {
            redirect('/test-path');
        } catch (\Exception $e) {
            // Suppress exception if headers are already sent
        }

        $headers = headers_list();
        ob_end_clean();

        $this->assertContains('Location: /test-path', $headers);
    }
```

1. testRedirect()
    - Purpose: This verifies that the redirect() function sets the correct HTTP header.
    - Justification: Ensures navigation works as intended when redirecting users.

```php
    // Test redirectToReferer function
    public function testRedirectToReferer()
    {
        $_SERVER['HTTP_REFERER'] = '/previous-page';

        ob_start();

        try {
            redirectToReferer();
        } catch (\Exception $e) {
            // Suppress exception if headers are already sent
        }

        $headers = headers_list();
        ob_end_clean();

        $this->assertContains('Location: /previous-page', $headers);
    }
```

2. testRedirectToReferer()
    - Purpose: Confirms the function redirects the users to the page they came from using $_SERVER[‘HTTP_REFERER’].
    - Justification: Important for preserving user flow or back-navigation behaviour.

```
51      // Test flash function - Setting and retrieving flash messages
52      public function testFlashSetAndGet()
53      {
54          flash('success', 'Operation successful');
55          $this->assertEquals('Operation successful', flash('success'));
56
57          // Ensure the message is removed after retrieval
58          $this->assertNull(flash('success'));
59      }
60
```

3. testFlashSetAndGet()
    - Purpose: Test the settings and then begins retrieving a flash message.
    - Justification: Ensures that the flash messages behave as expected--available once and then cleared (a common pattern in session messaging).

```
60
61      // Test generateSecret function - Generate a 32-character secret
62      public function testGenerateSecret()
63      {
64          $secret = generateSecret(32);
65          $this->assertEquals(32, strlen($secret));
66          $this->assertMatchesRegularExpression('/^[A-Z2-7]+$/', $secret);
67      }
68
```

4. testGenerateSecret()
    - Purpose: Verifies the generated secret is at least 32 characters long and in the correct Base32 format.
    - Justification: Critical for a 2FA security – secrets must follow the TOTP standard format.

```php
    // Test getQRCodeUrl function - Generates a valid QR code URL
    public function testGetQRCodeUrl()
    {
        $label = 'test@example.com';
        $secret = 'JBSWY3DPEHPK3PXP';

        $url = getQRCodeUrl($label, $secret);

        $this->assertStringContainsString('otpauth://totp/Enterprise%20Pro:test%40example.com', $url);
        $this->assertStringContainsString('secret=JBSWY3DPEHPK3PXP', $url);
        $this->assertStringContainsString('issuer=Enterprise%20Pro', $url);
    }
```

5. testGetQRCodeUrl()
    - Purpose: Confirms the QR code URL is properly constructed for OTP apps like Google Authenticator.
    - Checks: Presence of label, secret, and the issuer in the URL.
    - Justification: Ensures interoperability with OTP apps and helps with onboarding 2FA users.

```php

    // Test verifyCode function - Valid code should return true
    public function testVerifyCodeSuccess()
    {
        $secret = 'JBSWY3DPEHPK3PXP';

        // Generate a valid code using the same logic as the function
        $code = $this->generateTOTP($secret);

        $this->assertTrue(verifyCode($secret, $code));
    }

    // Test verifyCode function - Invalid code should return false
    public function testVerifyCodeFail()
    {
        $secret = 'JBSWY3DPEHPK3PXP';
        $invalidCode = '123456';

        $this->assertFalse(verifyCode($secret, $invalidCode));
    }

```

6. testVerifyCodeSuccess() and testVerifyCodeFail()
    - Purpose:
    - testVerifyCodeSucess(): Confirms that a valid TOTP code is accepted.
    - testVerifyCodeFail(): Confirms that an invalid code is rejected.
    - Justification: These are critical security tests. A valid OTP must authenticate, and invalid ones must be blocked. It also proves that the time-based tokens are functioning correctly.

```php
    // Test base32_decode function - Decode a valid base32-encoded string
    public function testBase32Decode()
    {
        $input = 'JBSWY3DPEHPK3PXP';
        $expectedOutput = "\x48\x65\x6C\x6C\x6F\x21\x21"; // Binary representation of "Hello!!"

        $this->assertEquals($expectedOutput, base32_decode($input));
    }

    // Test base32_decode function - Invalid characters should return false
    public function testBase32DecodeInvalidChars()
    {
        $this->assertFalse(base32_decode('JBSWY3DPEHPK3P!'));
    }
```

7. testBase32Decode() and testBase32DecodeInvalidChars()
    - Purpose:
    - testBase32Decode(): Ensures correct decoding of Base32-encoded secrets.
    - testBase32DecodeInalidChars(): Ensures invalid characters return false.
    - Justification: Vital for verifying encoding/decoding logic that is used in OTP operations and ensuring the integrity of user's data.

```php
    // Utility function to generate a valid TOTP code for testing
    private function generateTOTP($secret)
    {
        $key = base32_decode($secret);
        $time = floor(time() / 30); // 30-second window

        // Generate hash using HMAC-SHA1 based on the time window
        $timeBytes = pack('N*', 0) . pack('N*', $time);
        $hash = hash_hmac('sha1', $timeBytes, $key, true);

        // Extract dynamic offset from the hash
        $offset = ord($hash[19]) & 0xF;

        // Convert the hash to an integer
        $binary = ((ord($hash[$offset]) & 0x7F) << 24) |
            ((ord($hash[$offset + 1]) & 0xFF) << 16) |
            ((ord($hash[$offset + 2]) & 0xFF) << 8) |
            (ord($hash[$offset + 3]) & 0xFF);

        // Get the last 6 digits of the binary number as the OTP
        $otp = $binary % 1000000;

        // Return the OTP as a zero-padded 6-digit string
        return str_pad($otp, 6, '0', STR_PAD_LEFT);
    }
}
```

8. generateTOTP()
    - Purpose: Internal helper to generate a valid TOTP based on the current time and secret.
    - Justification: Enables accurate, repeatable test of the verify() function.

A.3: Code Inspection

Code Inspection with justification

2fa_checker.php

```php
<?php
if ($loggedInUser) { // Check if the user is logged in
    if (isset($_SESSION["auth_required"]) && $_SESSION['auth_required'] == true) { // Check if they requi

        // Ensure `$page` is defined and normalize for comparison
        $currentPage = strtolower(trim($page ?? ''));

        // Allow only if user is on '2fa' or 'index' page
        if (!in_array($currentPage, ['2fa', 'index'])) {
            flash('error', 'You do not have permission to access <strong>' . htmlspecialchars($title) . "
            redirect('/2fa.php'); // Redirect to 2FA page
        }
    }

    // Check if verified page
    if (!$loggedInUser['verified'] && !$loggedInUser['admin']) {
        flash('error', "You can't access this page as your account has not been verified yet");
        redirect('/index.php');
    }
} else {
    flash('error', 'You need to be logged in to access this page.');
    redirect('login.php');
}

```

1. 2FA Check:

- Checks If 2FA is Required Via session variable
- Normalize the current page name (Lowercase, trimmed)
- Only allows access to “2FA” or “index” Pages when 2FA is required
- Otherwise redirects to 2fa.php with an error message

2. Account verification:

- Verifies if user account is verified or if user is admin
- Non-verified and non-admin users get redirected to index.php
- Helps prevent unverified account from accessing protected areas.

3. Login Check:

- Fallback for non-logged-in users.
- Redirects to login page with appropriate

A.4: Peer Review

Team Member	Tasks Allocated	Achievements	Score
Adam	Team leadership, GitHub setup, testing coordination, code inspection.	Coordinated test efforts, refined acceptance tests, supported team with technical issues and debugging	10
Aliya	Test documentation formatting, acceptance testing, proofreading, meeting coordination.	Refined documentation templates, ensured quality assurance, actively helped with multiple stages of the project.	10
Yaasmeen	Unit testing, static web review, peer review, final test execution.	Reviewed static pages, corrected test failures, and ensured code met functional requirements.	10
Zainah	Meeting minutes, unit testing, code inspection, repository checking.	Maintained organized documentation, contributed to test debugging and ensured repository structure was complete.	10
Humairah	Test documentation formatting, debugging, static page improvements.	Implemented unit tests, checked ethical compliance, and contributed to documentation improvements.	10
Azeem	GitHub repository setup, acceptance test development final verification.	Ensured final testing alignment, helped inspect codebase, contributed to documentation and repo structure.	10
Shuaib	Unit test logic, test failures debugging, final testing execution.	Developed and corrected unit tests, reviewed project quality in final stages.	10
Bilal	Initial documentation, testing reviews, bug fixes, final proofreading.	Reviewed and refined test documents, participated in final project checks and submission readiness.	10

Reflection

As a team, we collaborated effectively and shared the workload equally during Part Two of the project. Everyone contributed to tasks such as testing, documentation, and debugging. Although testing and bug fixing were the most challenging parts, we supported each other and completed everything on time. We are proud of our teamwork and believe better time management in the testing phase could improve future projects.