z

CONTENT MANAGEMENT SYSTEM GREEN LEAF MEDIA

Assessment: Formative Assessment 1

NADINE NAUDE
20230395
04/09/2025

NadineNaude_20230395

Table of Contents

1. Client Brief Analysis

3.1 Overview of Client

Green Leaf Media started in 2015 as a group of people passionate about the environment and sustainable living. Over time, it grew into a trusted online platform that shares eco-friendly lifestyle tips, organic living practices, and wellness advice. The organisation's mission is to make green living easier and more accessible for everyone. Their vision is to become a leading platform for people who want to live in a way that supports sustainability and health.

3.2 Target Audience

The main audience includes eco-conscious individuals, families, and wellness communities who want reliable information about living sustainably. These users are often interested in topics like organic food, reducing waste, energy efficiency, and health tips that fit into an environmentally friendly lifestyle. Because their audience is diverse, the platform needs to be simple enough for everyday users while still offering depth for those who want to explore content more seriously.

3.3 Project Objectives

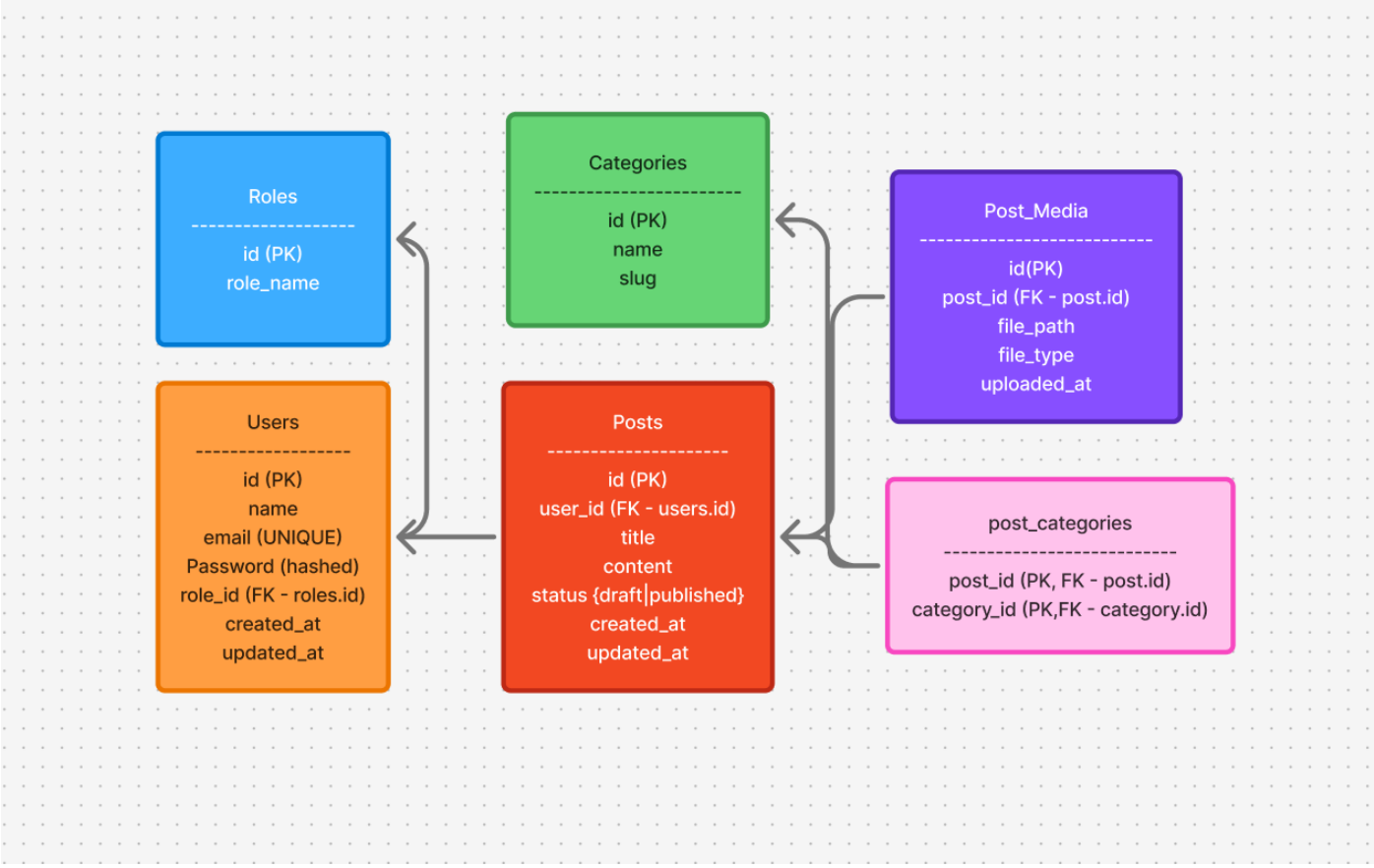
The project aims to create a custom Content Management System that makes it easier for Green Leaf Media to manage and share its growing content library. The system must allow multiple types of users, such as administrators, editors, and contributors, each with different levels of control and permissions. The CMS should also allow for uploading media, categorising content, and handling basic functions like creating, editing, and deleting posts. The design needs to be user-friendly and scalable so it can grow with the organisation as their content and audience expand.

3.4 Interpretation of Brief

This project is about giving Green Leaf Media the tools to keep up with their mission in a professional and efficient way. A CMS will help them manage all their articles, images, and categories in one system while making sure user permissions are secure. By creating a structured database and easy-to-use interface, the platform will support their long-term goal of becoming a leader in eco-friendly digital content. This system will not only help them deliver content more effectively but will also make sure the organisation can keep expanding without technical barriers.

2. Database Design Documentation

2.1 Entity Relationship Diagram



2.2 Database Schema

```sql
CREATE TABLE roles (
    id INT AUTO_INCREMENT PRIMARY KEY,
    role_name VARCHAR(50) NOT NULL
);
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role_id INT,
    FOREIGN KEY (role_id) REFERENCES roles(id)
);
CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
CREATE TABLE posts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    status ENUM('draft','published') DEFAULT 'draft',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
CREATE TABLE post_media (
    id INT AUTO_INCREMENT PRIMARY KEY,
    post_id INT,
    file_path VARCHAR(255) NOT NULL,
    file_type VARCHAR(50),
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (post_id) REFERENCES posts(id)
);
CREATE TABLE post_categories (
    post_id INT,
    category_id INT,
    PRIMARY KEY (post_id, category_id),
    FOREIGN KEY (post_id) REFERENCES posts(id),
    FOREIGN KEY (category_id) REFERENCES categories(id)
);

```

2.3 Design Notes

Why Users & Roles are separated

Keeping roles in their own table makes the system scalable. If Green Leaf Media wants to add a new role in the future, it can be done easily without changing the user structure.

Why Post_Categories exists

Since posts can belong to multiple categories, a junction table avoids data duplication and maintains relational integrity.

Why Media is separate

Media files can be heavy, so storing them in their own table (linked by post_id) makes the database cleaner and prevents unnecessary duplication in the Posts table.

Normalisation (3NF):

Each table has a unique purpose

No repeating groups of data

Non-key attributes depend only on the primary key

This database design ensures security, scalability, and efficiency, directly supporting Green Leaf Media's goal of managing a large and growing library of eco-friendly content.

3. User Roles & Permissions Documentation

3.1 Roles Overview

Admin – Full control. They can manage everything: users, roles, posts, media, categories, and site settings.

Editor – Content lead. They can create, edit, publish/unpublish, and delete content. They can't manage users/roles.

Contributor – Writer. They can create drafts and edit their own posts. They can't publish or delete other people's content.

3.2 Roles & Permissions Table

Permission	Admin	Editor	Contributor
Create post	✓	✓	✓
Edit own post	✓	✓	✓
Edit any post	✓	✓	
Delete own Post	✓	✓	✓
Delete any post	✓	✓	
Publish / Unpublish	✓	✓	
Upload Media	✓	✓	✓
Manage Categories	✓	✓	
Manage Users	✓		
Manage Roles	✓		
View Audit Log	✓	✓	

3. 3 SQL Scripts

```sql
CREATE TABLE IF NOT EXISTS roles (
  id INT AUTO_INCREMENT PRIMARY KEY,
  role_name VARCHAR(50) NOT NULL UNIQUE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS permissions (
  id INT AUTO_INCREMENT PRIMARY KEY,
  perm_key VARCHAR(50) NOT NULL UNIQUE,
  description VARCHAR(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS role_permissions (
  role_id INT NOT NULL,
  permission_id INT NOT NULL,
  PRIMARY KEY (role_id, permission_id),
  CONSTRAINT fk_rp_role FOREIGN KEY (role_id) REFERENCES roles(id) ON DELETE CASCADE,
  CONSTRAINT fk_rp_perm FOREIGN KEY (permission_id) REFERENCES permissions(id) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(100) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  role_id INT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  CONSTRAINT fk_users_role FOREIGN KEY (role_id) REFERENCES roles(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT IGNORE INTO roles (id, role_name) VALUES
  (1,'Admin'), (2,'Editor'), (3,'Contributor');
```

```sql
INSERT IGNORE INTO permissions (perm_key, description) VALUES
('post.create',      'Create post'),
('post.edit.own',    'Edit own post'),
('post.edit.any',    'Edit any post'),
('post.delete.own',  'Delete own post'),
('post.delete.any',  'Delete any post'),
('post.publish',     'Publish/unpublish posts'),
('media.upload',     'Upload media'),
('category.manage',  'Manage categories'),
('user.manage',      'Manage users'),
('role.manage',      'Manage roles'),
('audit.view',       'View audit log');

INSERT IGNORE INTO role_permissions (role_id, permission_id)
SELECT r.id, p.id FROM roles r CROSS JOIN permissions p WHERE r.role_name='Admin';

INSERT IGNORE INTO role_permissions (role_id, permission_id)
SELECT r.id, p.id FROM roles r
JOIN permissions p ON p.perm_key IN (
  'post.create','post.edit.own','post.edit.any',
  'post.delete.own','post.delete.any',
  'post.publish','media.upload','category.manage','audit.view'
)
WHERE r.role_name='Editor';

INSERT IGNORE INTO role_permissions (role_id, permission_id)
SELECT r.id, p.id FROM roles r
JOIN permissions p ON p.perm_key IN (
  'post.create','post.edit.own','post.delete.own','media.upload'
)
WHERE r.role_name='Contributor';

DROP VIEW IF EXISTS user_effective_permissions;
CREATE VIEW user_effective_permissions AS
SELECT u.id AS user_id, u.email, r.role_name, p.perm_key
FROM users u
JOIN roles r ON r.id = u.role_id
JOIN role_permissions rp ON rp.role_id = r.id
JOIN permissions p ON p.id = rp.permission_id;
```

```sql
DROP FUNCTION IF EXISTS has_permission;
DELIMITER $$
CREATE FUNCTION has_permission(p_user_id INT, p_perm_key VARCHAR(50))
RETURNS TINYINT(1)
READS SQL DATA
BEGIN
  DECLARE has_perm TINYINT(1) DEFAULT 0;
  SELECT 1 INTO has_perm
  FROM users u
  JOIN role_permissions rp ON rp.role_id = u.role_id
  JOIN permissions p ON p.id = rp.permission_id
  WHERE u.id = p_user_id AND p.perm_key = p_perm_key
  LIMIT 1;
  RETURN IFNULL(has_perm,0);
END$$
DELIMITER ;
```

3.4 Security Notes

Least privilege: People only get what they need to do their job. That lowers the risk of accidents or abuse.

Separation of duties: Contributors write, Editors control quality, Admins control access. This keeps content trustworthy and the system stable.

Auditing & accountability: With an audit. view permission, we can trace who changed what and when. That speeds up debugging and prevents finger-pointing.

Defense in depth: Roles/permissions are one layer. We still use secure password, prepared statements, CSRF tokens, server-side validation, and file-type checks on uploads.

Scalability: Adding a new role is as simple as inserting 1 role row and mapping a set of permissions.

4. CMS Functionality & Design

4.1 System Features

4.1.1 CRUD Operations (Posts)

What it does: create, read, update, and delete posts with titles, content, status (draft/published), categories, and media.

How it works:

Create: form server validates insert post attach categories/media.

Read: public/front shows published posts; CMS lists all.

Update: author or editor can edit; timestamps auto-update.

Delete: only Editor/Admin can delete any; Contributors can delete their own.

The brief requires full CMS features with CRUD and clear explanations.

4.1.2 Media Uploads

What it does: upload images (PNG/JPG/WebP) or videos linked to a post.

Safety checks I apply:

Check MIME type and file extension.

Restrict size.

Rename files to unique hashed names.

Store path in post_media table; never trust the original name.

The brief expects a user-friendly content interface incl. media uploads.

4.1.3 Categorisation

What it does: one post can live under multiple categories.

Why: keeps browsing clean and helps users find related content quickly.

How: many-to-many via post_categories.

Categorisation is specifically called out in the CMS requirements.

4.1.4 User Login & Role Management

What it does: secure login, and role-based access for Admin, Editor, and Contributor.

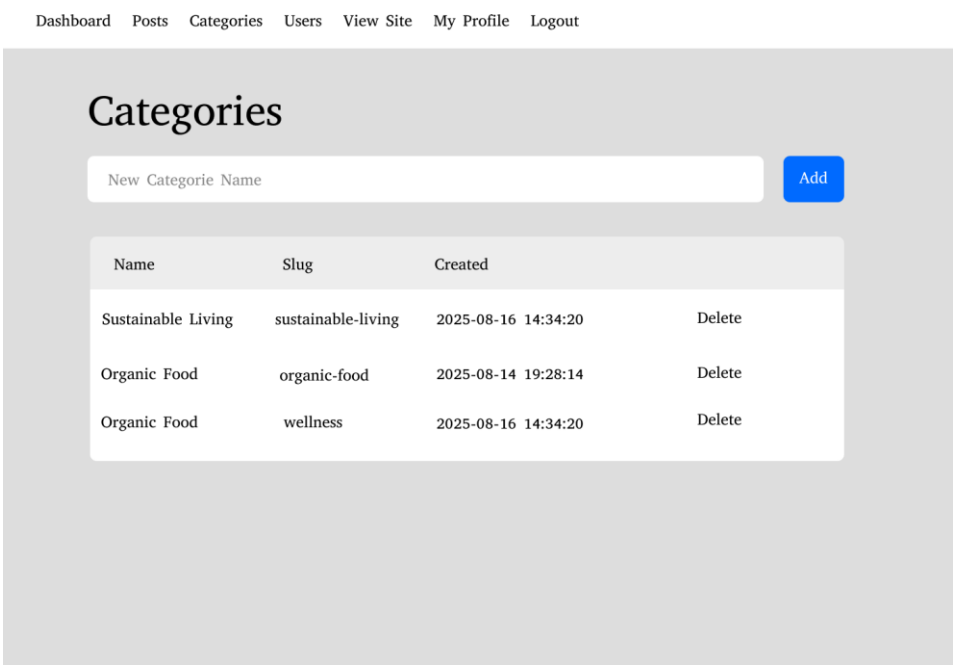
Admin: manage users, roles, categories, all content.

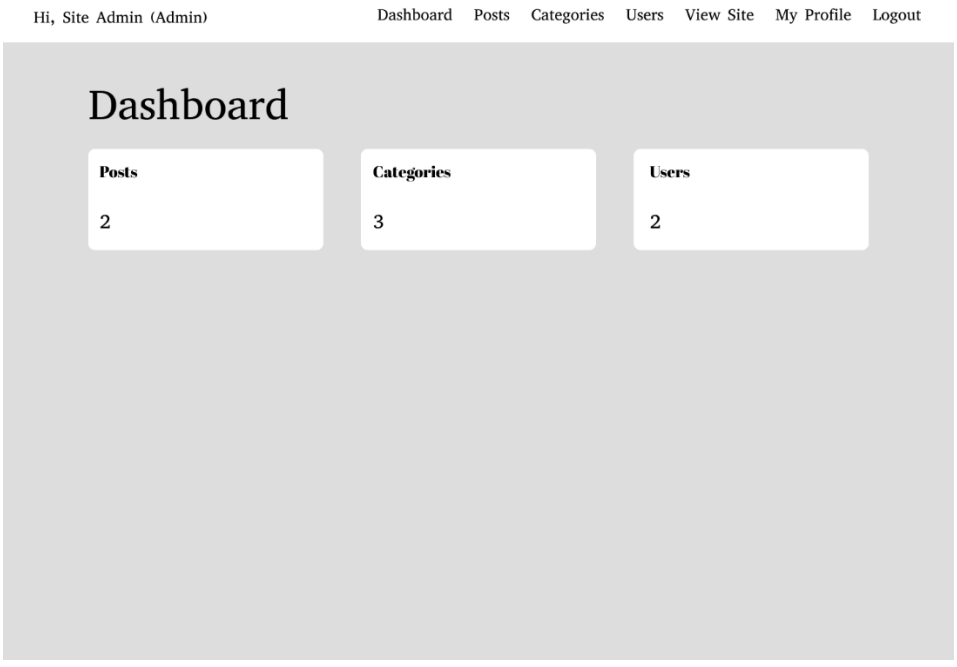
Editor: publish/unpublish, edit/delete any content.

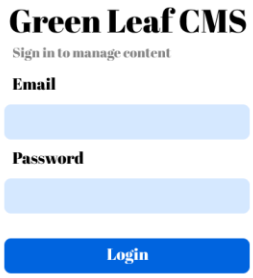
Contributor: create drafts, edit/delete own content only.

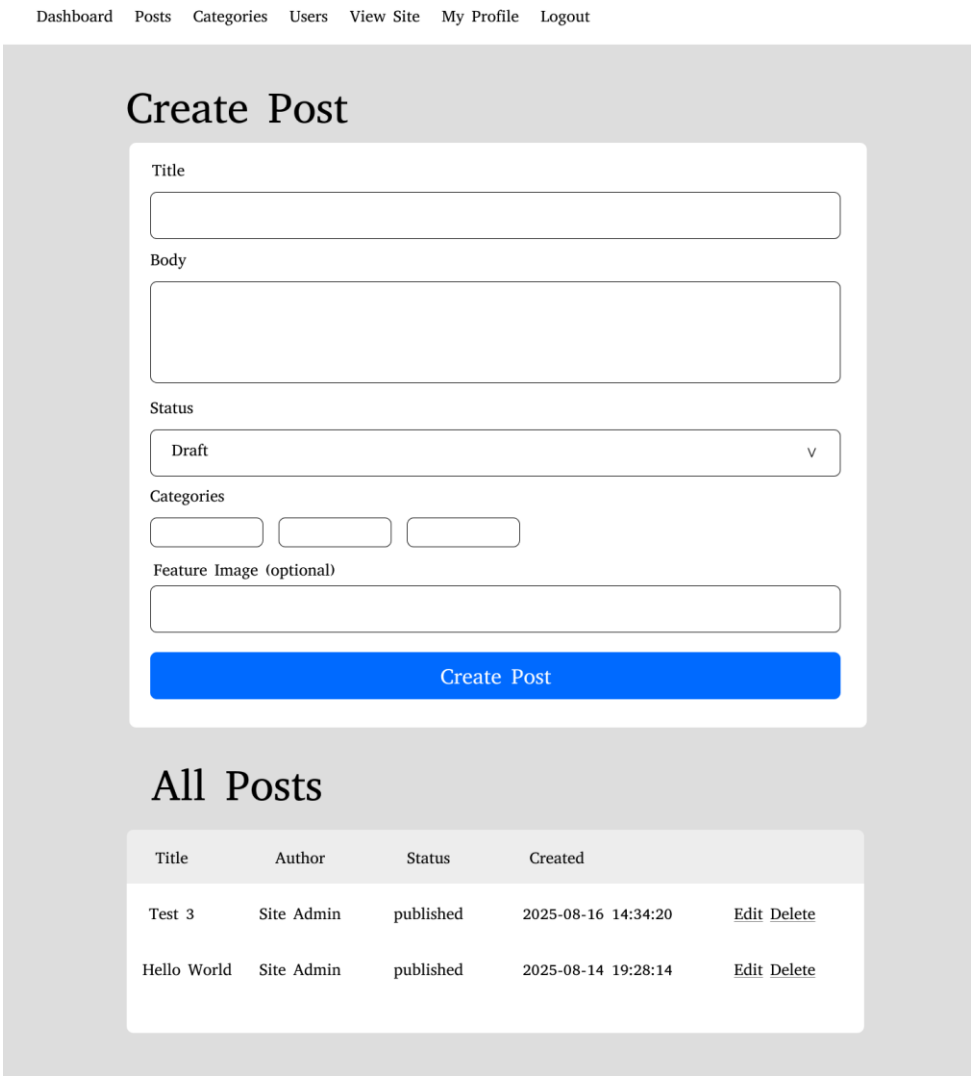
Enforcement: every sensitive action checks the user's role/permission on the server.

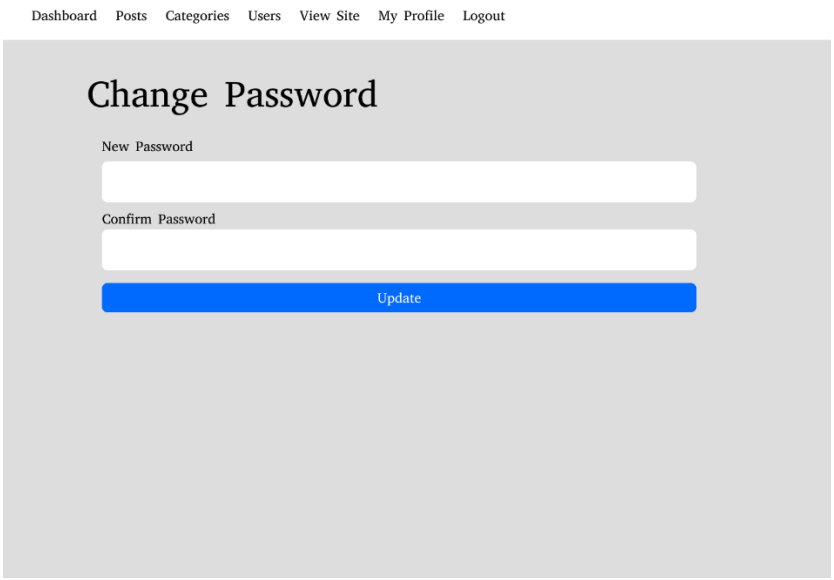
4.2 UI Mock-ups

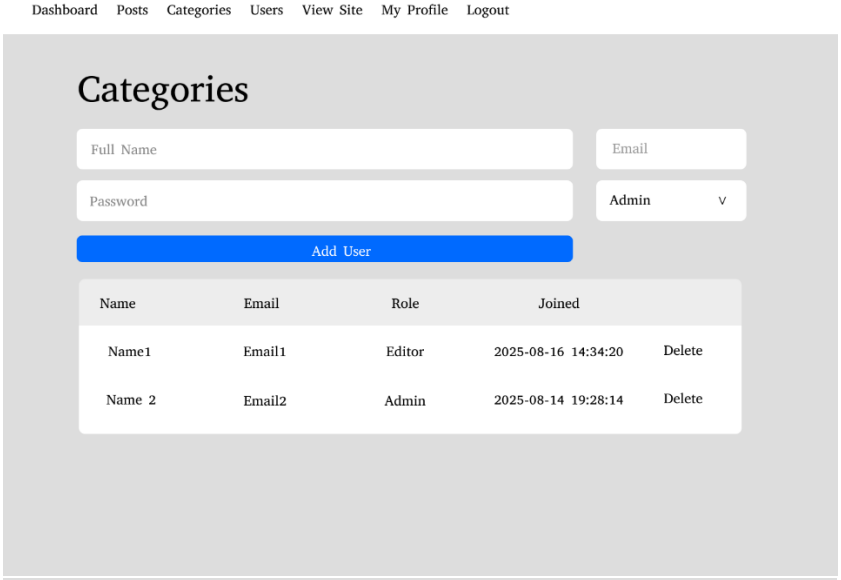


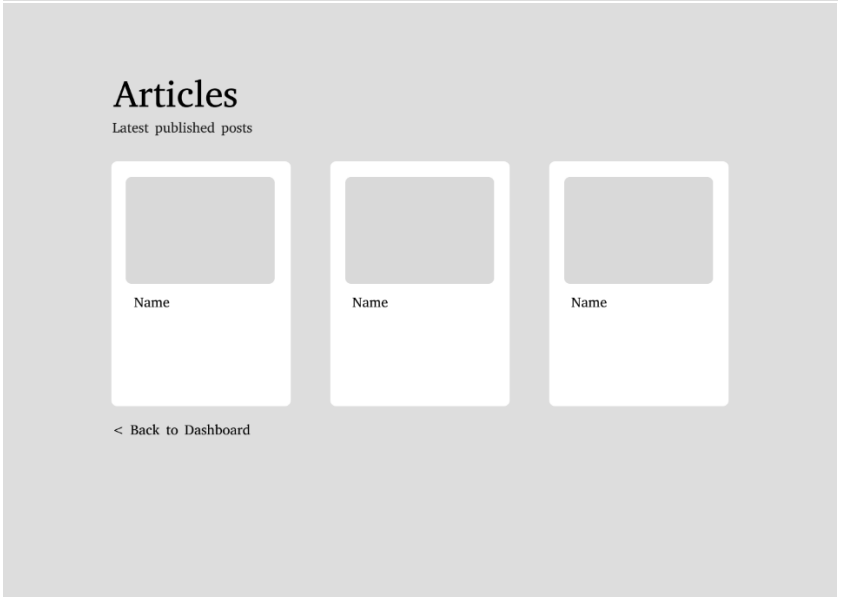



Green Leaf CMS
Sign in to manage content
Email
Password
Login


Dashboard Posts Categories Users View Site My Profile Logout
Create Post
Title
Body
Status
Draft
Categories
Feature Image (optional)
Create Post
All Posts
Title Author Status Created
Test 3 Site Admin published 2025-08-16 14:34:20 Edit Delete
Hello World Site Admin published 2025-08-14 19:28:14 Edit Delete


Dashboard Posts Categories Users View Site My Profile Logout
Change Password
New Password
Confirm Password
Update


Dashboard Posts Categories Users View Site My Profile Logout
Categories
Full Name
Email
Password
Admin
Add User
Name Email Role Joined
Name1 Email1 Editor 2025-08-16 14:34:20 Delete
Name 2 Email2 Admin 2025-08-14 19:28:14 Delete


Articles
Latest published posts
Name
Name
Name
< Back to Dashboard

4.3 Code Documentation

4.3.1 Setup: DB + CSRF helper

```php
<?php
// db.php
$pdo = new PDO(dsn: "mysql:host=localhost;dbname=greenleaf;charset=utf8mb4", username: "root", password: "", options: [
  PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
  PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
]);

// auth.php
session_start();
1 reference
function current_user(): mixed { return $_SESSION['user'] ?? null; }
0 references
function require_role(array $roles): void {
  $u = current_user();
  if (!$u || !in_array(needle: $u['role_name'], haystack: $roles, strict: true)) { http_response_code(response_code: 403); exit("Forbidden"); }
}

// csrf.php
0 references
function csrf_token(): mixed|string {
  if (empty($_SESSION['csrf'])) $_SESSION['csrf'] = bin2hex(string: random_bytes(length: 32));
  return $_SESSION['csrf'];
}
0 references
function csrf_check($token): void {
  if (!hash_equals(known_string: $_SESSION['csrf'] ?? '', user_string: $token ?? '')) { http_response_code(response_code: 400); exit("Invalid CSRF"); }
}

```

4.3.2 Create Post (with validation + categories)

```php
<?php
// create_post.php
require 'db.php'; require 'auth.php'; require 'csrf.php';
require_role(['Admin','Editor','Contributor']);

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  csrf_check($_POST['csrf'] ?? '');
  $title = trim(string: $_POST['title'] ?? '');
  $content = trim(string: $_POST['content'] ?? '');
  $status = ($_POST['status'] ?? 'draft') === 'published' ? 'published' : 'draft';
  $cats = $_POST['categories'] ?? []; // array of category IDs

  // Basic validation (server-side, always)
  $errors = [];
  if ($title === '') $errors['title'] = "Please add a title.";
  if ($content === '') $errors['content'] = "Please write some content.";
  if (!is_array(value: $cats)) $cats = [];

  if (!$errors) {
    // Insert post
    $st = $pdo->prepare("INSERT INTO posts (user_id, title, content, status) VALUES (?,?,?,?)");
    $st->execute([ current_user()['id'], $title, $content, $status ]);
    $postId = (int)$pdo->lastInsertId();

    // Link categories (if any)
    if ($cats) {
      $pc = $pdo->prepare("INSERT INTO post_categories (post_id, category_id) VALUES (?,?)");
      foreach ($cats as $cid) {
        if (ctype_digit(text: (string)$cid)) $pc->execute([$postId, (int)$cid]);
      }
    }
    header(header: "Location: edit_post.php?id=".$postId);
    exit;
  }
}

```

4.3.3 Upload Media (safe handling)

```php
<?php
// upload_media.php
require 'db.php'; require 'auth.php'; require 'csrf.php';
require_role(['Admin','Editor','Contributor']);
csrf_check($_POST['csrf'] ?? '');

$postId = (int)($_POST['post_id'] ?? 0);
if ($postId < 1) exit("Invalid post.");

if (!isset($_FILES['file']) || $_FILES['file']['error'] !== UPLOAD_ERR_OK) {
  exit("Upload failed.");
}

$f = $_FILES['file'];
$allowed = ['image/png'=>'png','image/jpeg'=>'jpg','image/webp'=>'webp'];
$maxBytes = 3 * 1024 * 1024;

if ($f['size'] > $maxBytes) exit("File too large.");
$finfo = new finfo(flags: FILEINFO_MIME_TYPE);
$mime = $finfo->file(filename: $f['tmp_name']);
if (!isset($allowed[$mime])) exit("Unsupported type.");

$ext = $allowed[$mime];
$basename = bin2hex(string: random_bytes(length: 16)).".$ext";
$dest = __DIR__."/uploads/".$basename;

if (!move_uploaded_file(from: $f['tmp_name'], to: $dest)) exit("Could not move file.");

$st = $pdo->prepare("INSERT INTO post_media (post_id, file_path, file_type) VALUES (?,?,?)");
$st->execute([$postId, "uploads/".$basename, $mime]);

echo "ok";
```

4.3.4 Delete Post (role-restricted)

```php
<?php
// delete_post.php
require 'db.php'; require 'auth.php'; require 'csrf.php';
require_role(['Admin','Editor']); // only these may delete any
csrf_check($_POST['csrf'] ?? '');

$id = (int)($_POST['id'] ?? 0);
$pdo->beginTransaction();
$pdo->prepare("DELETE FROM post_media WHERE post_id=?")->execute([$id]);
$pdo->prepare("DELETE FROM post_categories WHERE post_id=?")->execute([$id]);
$pdo->prepare("DELETE FROM posts WHERE id=?")->execute([$id]);
$pdo->commit();

header(header: "Location: posts_list.php");
```

4.3.6 Form Validation & Error Handling

Server-side first (title/content required, category IDs are integers, upload checks).

Short messages right under the field.

CSRF token on every POST form.

Graceful fails: if an error occurs, keep the user's input and show what went wrong.

Prepared statements everywhere.

File uploads: MIME check + size limit + random filename; store only the relative path in the DB.

5. Testing & Debugging

5.1 Test Plan

Test Case ID	Feature	Action	Expected Result	Actual Result	Status
T01	Login	Enter correct email + password	User is redirected to dashboard	Works as expected	✓ Pass
T02	Login	Enter wrong password	Error message shown, no login	Error message shown	✓ Pass
T03	Create post	Add title + content + category	New post saved as draft	Post saved and listed under drafts	✓ Pass
T04	Publish Post	Change status to “Published”	Post visible under published posts	Works as expected	✓ Pass
T05	Edit post	Contributor edits own post	Post updates successfully	Works correctly	✓ Pass
T06	Edit post	Try editing	Access denied	Error/Forbidden	✓ Pass
T07	Delete post	Delete any post	Post removed	Works as expected	✓ Pass
T08	Media Upload	Upload JPG under 3MB	File uploaded and linked to post	Works correctly	✓ Pass
T09	Media Upload	Upload .exe file	Upload rejected	File blocked	✓ Pass
T10	Categories	Add new category	Category saved + listed	Works as expected	✓ Pass
T11	Responsive Layout	Open CMS on mobile width	Menu collapses, layout adapts	Works fine	✓ Pass
T12	Security – CSRF	Submit form with wrong token	Request blocked	Form rejected	✓ Pass
T13	Security – SQL	Input ' OR 1=1 in login	Login blocked, error shown	Attack prevented	✓ Pass

5.2 Test Results

Login Tests

Correct login: Worked, user redirected to dashboard.

Incorrect password: Error message displayed, no access.

Empty fields: Validation triggered with “Please enter email/password.”

CRUD Operations

Create Post: Draft saved successfully, appeared in the post list.

Publish Post: Post marked as “Published” and visible in published list.

Edit Post: Contributor could edit their own draft without issues.

Edit Post : Contributor denied with a “Forbidden” message.

Delete Post: Editors and Admins could delete any post. Contributors only deleted their own.

Media Uploads

Valid file: Uploaded successfully and linked to post.

Invalid file: Upload rejected with error message.

Oversized file (5MB PNG): Blocked, size error shown.

Categories

Add Category: New category saved and displayed.

Assign Post to Category: Post correctly linked to multiple categories.

Remove Category: Category deleted and unlinked from posts.

Responsiveness

Desktop view: Dashboard and tables displayed correctly.

Mobile view: Navigation collapsed into menu, post list stacked as cards.

Security Checks

CSRF token test: Submitting form with wrong token rejected the request.

SQL injection attempt: ' OR 1=1 in login blocked by prepared statements.

Session expiry: After logout, user could not access restricted pages.

5.3 Bug Report

Bug ID	Description	Cause	Fix Applied	Verification
B01	Posts not updating “updated_at” correctly	Missing ON UPDATE in SQL schema	Added updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Verified – timestamp updates on edit
B02	File upload accepted any type	Only file extension was checked	Added MIME type + size validation with finfo in PHP	Verified – .exe blocked
B03	Contributor could delete any post	Permission check missing on delete action	Added require_role(['Admin','Editor']) + ownership check	Verified – Contributors can only delete their own
B04	CSRF token sometimes missing on forms	Forgot to include hidden token field	Added <?= csrf_token() ?> to every POST form	Verified – invalid token now blocked
B05	Category links not saving	Loop didn’t check category ID type	Added ctype_digit check before inserting into post_categories	Verified – categories linked properly

6. Final Documentation & Presentation

6.1 Portfolio Compilation

The Entity Relationship Diagram (ERD) for the database.

Full SQL scripts for all tables, roles, and permissions.

Screenshots of the CMS UI (dashboard, add/edit post, manage users, manage categories).

Annotated PHP code showing CRUD operations with validation and role checks.

Testing & debugging documentation (test plan, test results, bug report).

The Client Handover Guide so Green Leaf Media can take over the system without needing a developer present.

6.2 Presentation Summary

DATABASE & ROLES

- ERD shows users, roles, posts, categories, media
- Tables linked through relationships (normalised to 3NF)
- Roles mapped to secure permissions
- Principle of least privilege followed

CMS FEATURES

- CRUD operations for posts
- Media uploads with validation (file type & size)
- Categorisation (posts can belong to multiple categories)
- Secure user login & role management
- Screenshots of dashboard, add/edit post, manage users

TESTING & DEBUGGING

- Test plan covered login, CRUD, roles, responsiveness, security
- Screenshots of successful test cases
- Bugs identified & fixed:
  - File upload filter
  - Contributor delete permissions
  - Updated_at timestamp

OUTCOMES & CLIENT VALUE

- CMS makes content management efficient and secure
- Scalable design supports future growth
- Professional UI improves usability for team
- Directly supports Green Leaf Media's mission of promoting sustainable living

6.3 Client Handover Guide

Step 1: Logging In

Go to the CMS login page (/login.php).

Enter your email and password.

If credentials are correct, you'll be taken to the dashboard.

If you forget your password, click "Forgot Password" to reset via email.

Step 2: Adding & Editing Content

On the dashboard, click "Add New Post."

Enter a title, content, and select categories.

(Optional) Upload an image or video.

Choose Draft (save only) or Publish (visible online).

To edit a post, click Posts Manage Posts, then select Edit.

Step 3: Managing Categories

Click Categories Manage Categories.

To add a new one, type the category name and click Add.

To edit or delete, use the actions next to the category in the list.

Step 4: Managing Users (Admin only)

Go to Users Manage Users.

Admin can add, edit, or remove users.

Each user must be assigned a role (Admin, Editor, Contributor).

Step 5: Basic Troubleshooting

Can't log in? Check your email/password or reset it.

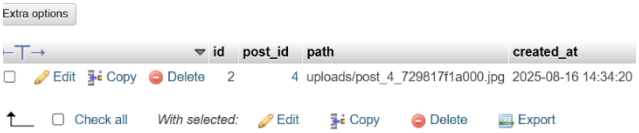
Upload fails? Make sure file is JPG/PNG/WebP and under 3MB.

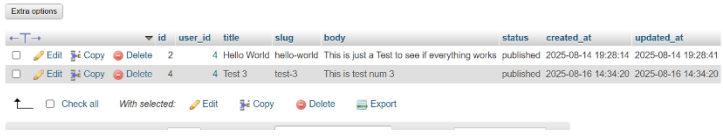
Post not showing? Make sure it is marked as "Published" and not "Draft."

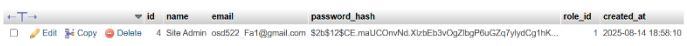
Error message appears? Note the steps and send it to the system admin.

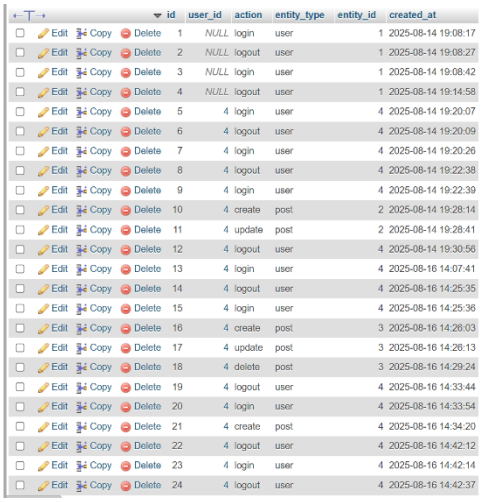
7. Assessment Exercises

7.1 Exercise 1 – SQL Query Practice









NadineNaude_20230395

7.2 Exercise 2 – CRUD Operations

Objective: Practice basic Create, Read, Update, Delete in PHP with validation

```php
if (isset($_GET['delete'])) {
    $pid = (int)$_GET['delete'];

    $ms = $pdo->prepare("SELECT path FROM post_media WHERE post_id=?");
    $ms->execute([$pid]);
    foreach ($ms->fetchAll() as $m) {
        @unlink(filename: __DIR__ . '/' . $m['path']);
    }

    $pdo->prepare("DELETE FROM post_categories WHERE post_id=?")->execute([$pid]);
    $pdo->prepare("DELETE FROM posts WHERE id=?")->execute([$pid]);
    log_action($pdo, current_user()['id'], 'delete', 'post', $pid);

    header(header: "Location: posts.php");
    exit;
}
```

```php
if (isset($_GET['edit'])) {
    $id = (int)$_GET['edit'];
    $stmt = $pdo->prepare("SELECT * FROM posts WHERE id=?");
    $stmt->execute([$id]);
    $editing = $stmt->fetch();

    if ($editing) {
        $st = $pdo->prepare("SELECT category_id FROM post_categories WHERE post_id=?");
        $st->execute([$id]);
        $selected_cats = array_column(array: $st->fetchAll(), column_key: 'category_id');

        $st2 = $pdo->prepare("SELECT * FROM post_media WHERE post_id=? ORDER BY id ASC");
        $st2->execute([$id]);
        $editing_media = $st2->fetchAll();
    }
}
```

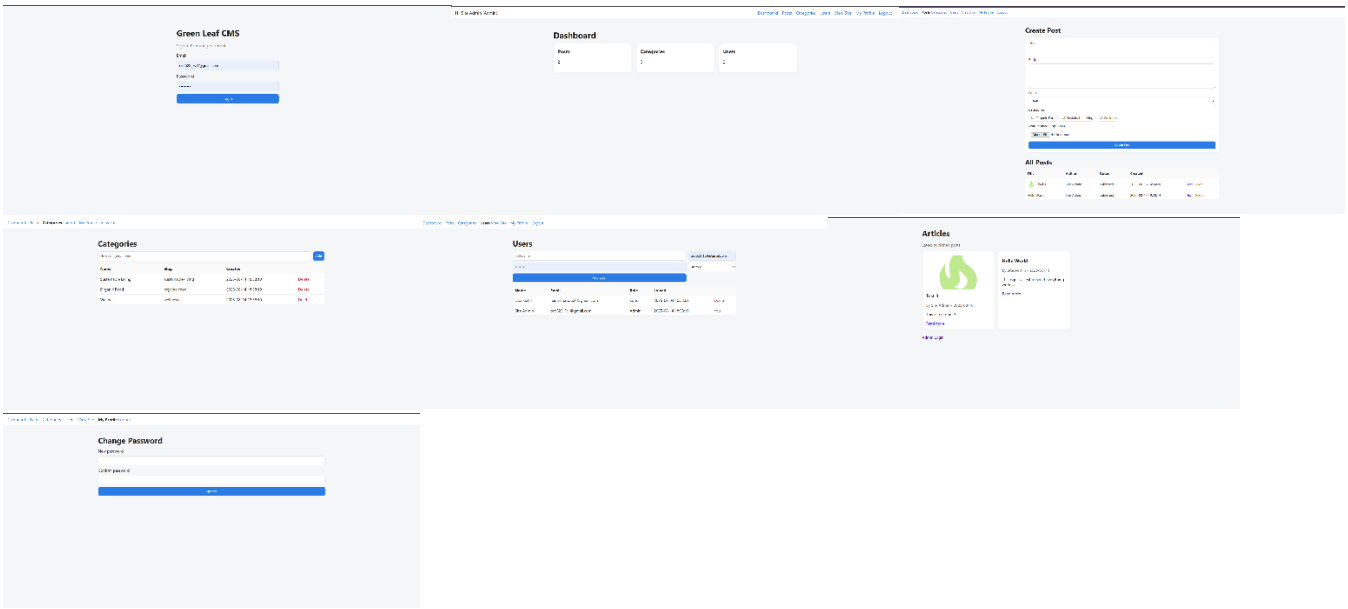
```php
1 reference
function ensure_uploads_dir(): string {
    $uploadsDir = __DIR__ . "/uploads";
    if (!is_dir(filename: $uploadsDir)) {
        @mkdir(directory: $uploadsDir, permissions: 0775, recursive: true);
    }
    return $uploadsDir;
}

if (isset($_GET['delmedia'])) {
    $mid = (int)$_GET['delmedia'];
    $st = $pdo->prepare("SELECT * FROM post_media WHERE id=?");
    $st->execute([$mid]);
    if ($row = $st->fetch()) {
        @unlink(filename: __DIR__ . '/' . $row['path']);
        $pdo->prepare("DELETE FROM post_media WHERE id=?")->execute([$mid]);
    }
    $redir = 'posts.php';
    if (!empty($_GET['edit'])) $redir .= '?edit=' . (int)$_GET['edit'];
    header(header: "Location: " . $redir);
    exit;
}

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $title   = trim(string: $_POST['title'] ?? '');
    $body    = trim(string: $_POST['body'] ?? '');
    $status  = $_POST['status'] ?? 'draft';
    $cat_ids = $_POST['categories'] ?? [];
    $slug    = strtolower(string: preg_replace(pattern: '/[^a-z0-9]+/i', replacement: '-', subject: $title));

    if ($title === '' || $body === '') {
        die("Title and Body are required.");
    }
```

7.3 Exercise 3 – CMS Interface Design



Responsive Notes:

Grid Layout: content tables collapse into stacked cards on mobile.

Forms: input fields go full width on small screens.

Navigation: side menu turns into a top hamburger menu for mobile.

NadineNaude_20230395

7.4 Exercise 4 – Integration Testing

Tests Done:

CRUD Testing

Created, edited, published, and deleted posts.

Checked category linking and media uploads.

Role Testing

Contributor: could add draft, but not publish.

Editor: could publish, edit any post, but not manage users.

Admin: full access, including roles and users.

Performance Testing

Inserted 100 dummy posts CMS still loads without errors.

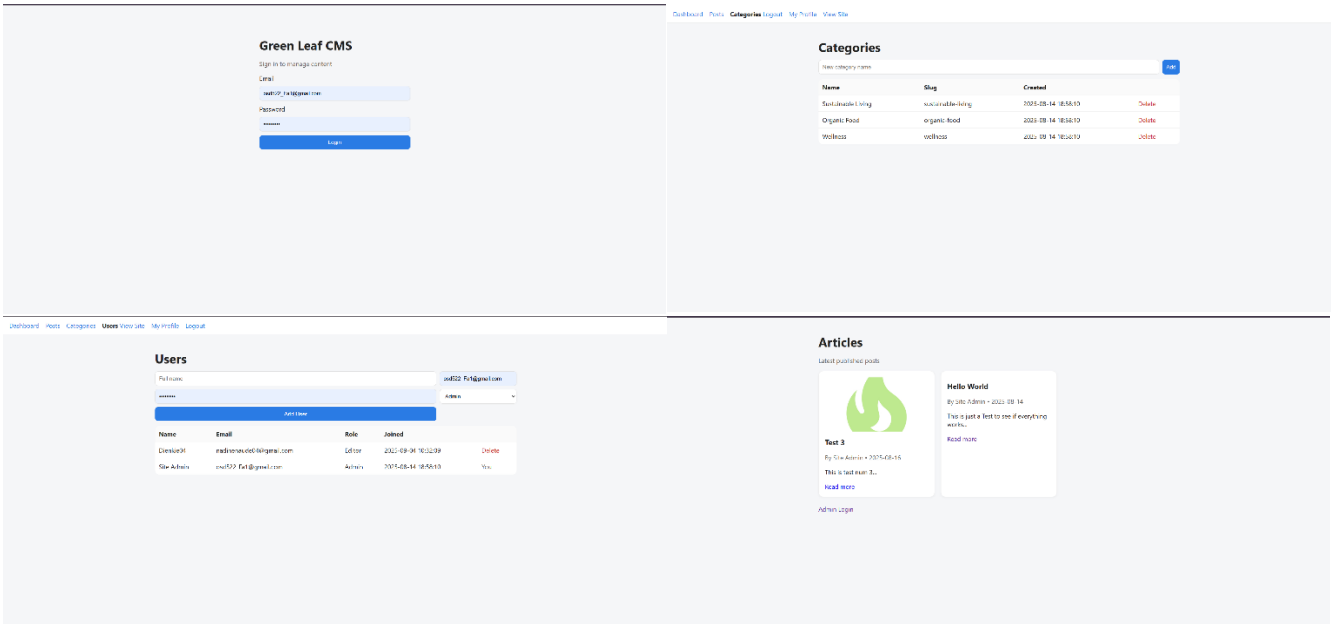
Security Testing

Invalid CSRF token form rejected.

Wrong file type upload blocked.

SQL injection attempt blocked by prepared statements.

Evidence:



8. Reflection

Building this CMS taught me a lot about how different parts of a system connect and depend on each other. I learned how important it is to first design the database properly because everything else in the project relies on a clean, well-structured schema. By normalising the database to third normal form and carefully setting up relationships, I saw how it makes the system easier to maintain and less likely to break as it grows.

One of the main challenges I faced was making sure the user roles and permissions were set up correctly. At first, I tried to handle permissions directly in the users table, but it quickly became messy. I overcame this by separating roles and mapping permissions in a more flexible way. This not only made the system secure but also showed me how the principle of least privilege works in practice. Another challenge was debugging the PHP code for CRUD operations. I often ran into errors with validation and session handling, but by testing step by step and using clear error outputs, I was able to fix them.

Through this project, I gained real skills in SQL for creating and managing relational databases, PHP for building dynamic pages and CRUD functionality, and testing techniques to make sure everything worked as planned. I also got a stronger understanding of system security, like why hashed passwords, CSRF tokens, and file type checks are necessary.

This project gave me hands-on experience that felt close to what a real developer would do when building a CMS for a client. It also boosted my confidence in combining technical skills (coding, database design) with good practices.

9. Bibliography / References

CTU Training Solutions. (2025). OSD522 – Online Systems Development Formative Assessment 1 Brief. Bloemfontein: CTU Training Solutions.

Connolly, T. & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation and Management. 6th ed. Harlow: Pearson Education.

W3Schools. (2025). PHP & MySQL Tutorial.

https://www.w3schools.com/php/

W3Schools. (2025). SQL Tutorial.

https://www.w3schools.com/sql/

Mozilla Developer Network (MDN). (2025). Form Validation – Client & Server Side.

https://developer.mozilla.org/

OWASP Foundation. (n.d.). Security Guidelines: Input Validation, CSRF & Role-Based Access Control. https://owasp.org/

Lecturer Feedback