3004ICT - Individual Project

Due Date: 5pm Friday 3 October 2025 (Week 11)
Weighting: 90%

Introduction

In this major project, you will create a comprehensive web application for managing and booking events. This system will allow organisers to list events and attendees to register and book spots for these events.

As this project constitutes 90% of your final grade, it is designed to assess a wide range of skills expected of a graduating developer. This includes not only core application development using the Laravel framework and SQLite but also database design, advanced query implementation, automated testing, and professional documentation. You will be expected to demonstrate a deep understanding of the code you submit. A key part of this assignment involves you designing and implementing a significant feature based on a general requirement area.

Terminology

- **Organiser:** A user type responsible for creating and managing events.
- **Attendee:** A user type who can register, view events, and book attendance.
- **Event:** A scheduled occasion (e.g., workshop, seminar, social gathering) with details like date, time, location, and capacity.
- **Booking:** A record indicating an Attendee has reserved a spot for a specific Event.

Core Application Requirements

Your implementation must use Laravel's migrations, seeders, models, ORM/Eloquent, route, controllers, validator, and view/blade. You have some freedom in designing your website's look and feel, however, it must satisfy the following requirements:

1. User Types & Registration/Seeding:

1. There are two types of users: Organiser and Attendee.
2. At least 2 Organiser users must be seeded into the database via a seeder. Organisers require name, email, and password.

3. Attendee users must be able to **register** through a form, providing their name, email, and password. User type should be implicitly 'Attendee' upon registration.

2. Authentication:

1. All users (Organisers and Attendees) must **login** to access the system's functionalities beyond a public event listing (see point 3). Login should use email and password.
2. Once logged in, the user's name and type (Organiser or Attendee) must be displayed at the top of every page.
3. A logged-in user must be able to **log out**.

3. Home Page / Event Listing:

1. The main page (accessible without login) should display a list of all **upcoming** events (events where the date/time is in the future).
2. Each event listing should show at least the Event Title, Date, Time, and Location.
3. Implement **pagination** for the event list, displaying a maximum of 8 events per page. Ensure you have enough seeded/created events to demonstrate pagination.
4. Clicking on an event title leads to the Event Details page.

4. Event Details Page:

1. Displays all details for a specific event: Title, Description, Date, Time, Location, Capacity, Name of the Organiser.
2. Shows the number of currently available spots (Capacity - Current Bookings).
3. If the user is logged in as an Attendee and the event is not full and is upcoming, a "Book Now" button should be visible.
4. If the user is logged in as the Organiser who created the event, "Edit" and "Delete" buttons should be visible.

5. Organiser: Event Management (CRUD):

1. Logged-in Organisers can **create** new events via a form.
2. Inputs: Title (required, max 100 chars), Description (textarea, optional), Date & Time (required, must be in the future), Location (required, max 255 chars), Capacity (required, integer, min 1, max 1000).
3. Standard Laravel validation should be used for these fields.
4. Organisers can **update** events they created. The update form should be pre-filled with existing data. Validation rules apply.
5. Organisers can **delete** events they created, BUT only if there are **no bookings** for that event. If deletion is attempted on an event with bookings, redirect back with an informative error message.

6. Organiser: Dashboard & Reporting

1. An Organiser must have a "Dashboard" page. This page must feature a report that lists **all events created by that organiser**.
2. For each event, the report must display:
   * Event Title
   * Event Date
   * Total Capacity
   * **Number of Current Bookings**
   * **Remaining Spots** (Capacity - Bookings)
   * **Raw SQL Requirement:** The database query that generates the data for this report **must be written using a raw SQL statement** (e.g., via DB::select()). This query will require joining the events table with an aggregate count from the bookings table. The rest of the application's queries should use Eloquent.

7. Attendee: Booking & Manual Validation:

1. Logged-in Attendees can **book** an upcoming event by clicking the "Book Now" button on the Event Details page.
2. An attendee **cannot** book the same event more than once. This can be handled by standard validation (e.g., a unique rule).
3. An attendee **cannot** book an event if it is already full.
   * **Requirement:** The validation check for **event capacity** MUST be implemented **manually** within your Controller action (or a Form Request). You must explicitly query the current number of bookings for the event and compare it to the event's capacity *before* creating the booking record. If the event is full, redirect back with a clear error message. This is the **mandatory manual validation**.

8. Attendee: View My Bookings:

1. Logged-in Attendees must have a page listing all the events they have successfully booked (Event Title, Date, Time, Location).

9. User Consent and Privacy Policy Integration:

To ensure ethical data collection, the application includes an informed consent mechanism during user registration. Users must explicitly agree to the Privacy Policy and Terms of Use before creating an account. This is implemented via a required checkbox in the registration form, validated server-side to prevent bypassing. The Privacy Policy outlines:

- What data is collected (name, email, password, booking history)
- Why it is collected (authentication, event participation)
- How it is stored and protected (hashed passwords, access control)
- User rights (view, manage, and delete their data)

You can use AI to generate your privacy policy.

---

10. Design and Implement ONE Advanced Feature

Task: Choose ONE of the following feature areas. You must design and implement a solution that meets the Core Requirements listed.

To achieve a high distinction, your implementation must also include two Excellence Markers:

1. The specific Mandatory Excellence Marker provided for your chosen feature area.
2. A Student-Designed Excellence Marker of your own creation. Your designed feature must be non-trivial and you must be able to justify how it demonstrates excellence by aligning with the characteristics defined below.

What Defines an "Excellent" Feature?

A basic feature simply works. An excellent feature demonstrates deeper thought about the user, the business process, and the technology. When designing your own Excellence Marker, ensure it aligns with one or more of these characteristics:

- Improves User Experience: It makes the system significantly easier, faster, or more informative for the user (either Attendee or Organiser). It anticipates a user's next step or solves a point of friction.
- Automates a Process: It intelligently automates a task that would otherwise need to be done manually, saving time and reducing the chance of human error.
- Introduces Complex Business Logic: It handles a non-trivial rule, edge case, or state management that goes beyond the simple CRUD operations of the core requirements.
- Demonstrates Advanced Technical Skill: It correctly uses a more advanced feature of the Laravel framework or modern web development (e.g., Event Listeners, Queues, Scheduled Tasks, API interactions, AJAX, etc.).

Feature Area Options (Choose ONE):

(A) Event Waiting List
High-Level Goal: When an event is full, both Attendees and Organisers must be able to properly interact with a waitlist system.

Core Functional Criteria (What you MUST achieve):

- **Attendee Perspective:**

    - A logged-in attendee must have a clear way to join a waitlist for an event that is full.
    - An attendee must be able to view which waitlists they are currently on and have the option to leave one.
- **Organiser Perspective:**
    - An organiser must be able to see a list of all attendees who are on the waitlist for any of their specific events.
- **Underlying Principles:**
    - **Data Persistence:** The system must reliably store and manage the association between a user and a waitlisted event.
    - **Conditional UI:** The user interface must intelligently change to show waitlist options only when an event is full and the user is eligible to join.

Excellence Markers (Path to High Distinction):

- 1. Mandatory Marker to Implement: Automated Notification. When a booking is cancelled for a full event, an email notification must be automatically triggered for the first user on the waitlist. (This must be tested with Mail::fake() and demonstrated with the log mailer).
- 2. Your Designed Marker: You must design, document, and implement one additional feature for the waitlist system that demonstrates excellence by aligning with the characteristics defined above.

(B) Event Categories/Tags
High-Level Goal: To allow Organisers to tag events with relevant categories and for Attendees to use these tags for discovery.

Core Functional Criteria (What you MUST achieve):

- **Organiser Perspective:**
    - An organiser must be able to assign one or more categories/tags to an event during its creation or update process.
- **Attendee/Public Perspective:**
    - The categories assigned to an event must be clearly visible on its details page.

        - The main event listing page must provide a way for users to filter the list of events based on a selected category.
    - **Underlying Principles:**
        - **Flexible Data Structure:** The database design must allow for a flexible relationship between events and categories. You must justify your chosen database structure (e.g., one-to-many, many-to-many, etc.) in your documentation.
        - **Intuitive Filtering:** The UI for filtering must be clear and easy for a non-technical user to understand and operate.

Excellence Markers (Path to High Distinction):

- 1. Mandatory Marker to Implement: AJAX Filtering. The category filtering on the homepage must be implemented using AJAX, so the event list updates smoothly without a full page reload.
- 2. Your Designed Marker: You must design, document, and implement one additional feature for the category system that demonstrates excellence by aligning with the characteristics defined above.

---

11. Comprehensive Automated Testing

A significant component of this project is demonstrating professional development practices, including robust automated testing. You are required to implement a comprehensive suite of feature tests that validate the core functionality and business rules of your application.

Your tests must be logically organized into the specific files and use the method names as instructed below. This standardization allows for rapid, automated verification during your demonstration using the testdox runner.

Core Functionality Test Requirements

The following tests for the application's core functionality are required.

A. Guest / Public User Scenarios

- Required Test File: tests/Feature/GuestAccessTest.php

You must implement methods to test the following requirements:

- Requirement: A guest can view the paginated list of upcoming events.

    - Method: test_a_guest_can_view_the_paginated_list_of_upcoming_events()
- Requirement: A guest can view the details page for a specific event.
    - Method: test_a_guest_can_view_a_specific_event_details_page()
- Requirement: A guest is redirected to the login page for authenticated routes.
    - Method: test_a_guest_is_redirected_when_accessing_protected_routes()
- Requirement: A guest viewing an event details page cannot see action buttons.
    - Method: test_a_guest_cannot_see_action_buttons_on_event_details_page()

B. Attendee User Scenarios

- Required Test File: tests/Feature/AttendeeActionsTest.php

You must implement methods to test the following requirements:

- Requirement: A user can successfully register as an Attendee.
    - Method: test_a_user_can_successfully_register_as_an_attendee()
- Requirement: A registered Attendee can log in and log out.
    - Method: test_a_registered_attendee_can_log_in_and_log_out()
- Requirement: A logged-in Attendee can book an available, upcoming event.
    - Method: test_a_logged_in_attendee_can_book_an_available_upcoming_event()
- Requirement: After booking, the event is on their "My Bookings" page.
    - Method: test_after_booking_an_attendee_can_see_the_event_on_their_bookings_page()
- Requirement: An Attendee cannot book the same event more than once.
    - Method: test_an_attendee_cannot_book_the_same_event_more_than_once()
- Requirement: An Attendee cannot book a full event (manual capacity check).
    - Method: test_an_attendee_cannot_book_a_full_event()

- Requirement: An Attendee cannot see "Edit" or "Delete" buttons on an event page.
    - Method: test_an_attendee_cannot_see_edit_or_delete_buttons_on_any_event_page()

C. Organiser User Scenarios

- Required Test File: tests/Feature/OrganiserActionsTest.php

You must implement methods to test the following requirements:

- Requirement: An Organiser can log in and view their specific dashboard.
    - Method: test_an_organiser_can_log_in_and_view_their_specific_dashboard()
- Requirement: An Organiser can successfully create an event with valid data.
    - Method: test_an_organiser_can_successfully_create_an_event_with_valid_data()
- Requirement: An Organiser gets validation errors for invalid event data.
    - Method: test_an_organiser_receives_validation_errors_for_invalid_event_data()
- Requirement: An Organiser can successfully update an event they own.
    - Method: test_an_organiser_can_successfully_update_an_event_they_own()
- Requirement: An Organiser cannot update an event created by another Organiser.
    - Method: test_an_organiser_cannot_update_an_event_created_by_another_organiser()
- Requirement: An Organiser can delete an event they own with no bookings.
    - Method: test_an_organiser_can_delete_an_event_they_own_that_has_no_bookings()
- Requirement: An Organiser cannot delete an event with active bookings.

- Method: test_an_organiser_cannot_delete_an_event_that_has_active_bookings()

D. User Registration

- Update tests/Feature/Auth/RegistrationTest.php
- Requirement: Users must explicitly agree to the Privacy Policy and Terms of Use before creating an account
  - Existing Method test_new_users_can_register()
- Requirement: User cannot register without agreeing to Privacy Policy
  - Method test_user_cannot_register_without_agreeing_to_privacy_policy()

Advanced Feature Test Requirements

If you implement an advanced feature, you must also provide tests for it in a dedicated test file. This file must contain tests for all Core Functional Criteria of the feature, the Mandatory Excellence Marker, and your Student-Designed Excellence Marker.

Testing Policy and Execution

- AI for Test Generation: You are permitted to use AI tools (e.g., ChatGPT, Copilot) to help generate the code for your automated tests. However, you are fully responsible for understanding, implementing, debugging, and explaining these tests during the demonstration. The logic of your main application must be your own.
- Test Execution: All tests you write must pass when php artisan test is run from the project root. If a feature test fails, that feature will be marked as incorrect.

Technical & Professional Practice Requirements

- Use Laravel's **migration** for all database table creation.
- Use Laravel's **seeder** to insert sufficient test data (Minimum: 2 Organisers, 10 Attendees, 15 Events).
- Use Laravel's **ORM/Eloquent** for all database operations, **with the explicit exception of the raw SQL query required in Requirement 6**.

- Implement proper server-side **input validation** for ALL inputs.
- Implement proper **authorisation** to ensure users can only perform actions they are permitted to (e.g., an attendee cannot edit an event).
- **Professional Code Commenting:** Your code must be clearly commented to a professional standard. This is not about quantity, but quality.
    - **Requirement**: Every method in your Controllers, Models, and any other custom classes you create must have a comment block above it. This block should succinctly explain the method's purpose, its parameters (@param), and its return value (@return).
    - **Purpose over Redundancy**: Comments should explain the why, not just repeat the how. They should clarify complex logic, business rules, or the intent behind a piece of code for a future developer.
    - **Expectation for Demonstration**: Comments are for code maintainability. They are not a script for your demonstration. During your final demonstration, you will be required to explain the underlying logic, control flow, and design choices of your implementation. Simply reading your comments aloud will does not demonstrate you understand the code, and will be used as an indication that you are not able to explain your code.

AI Use Policy & Understanding

You are free to use AI tools to assist you. However, you must be the author of your application's logic and you must be able to professionally articulate your design and implementation choices.

- **Permitted AI-Generated Code:** The *only* code you may submit that was primarily generated by an AI is the automated test suite (Requirement 11). You can ask AI to generate your Privacy Policy and Terms of Use (Requirement 9)
- **Referencing AI:** For any other assistance (e.g., debugging help, conceptual questions), you must provide a link to the chat log or copy/paste the relevant prompts and AI responses into a references.txt file.
- **Verification of Authorship:** Your understanding and authorship of the project will be rigorously assessed during the final demonstration. An inability to explain any part of your application's core logic, architecture, or database structure will be treated as a failure to meet the assignment requirements and will result in significant mark deductions. "The AI wrote it for me" or "I don't remember" are not acceptable explanations.

Assignment Demonstration and Verification

You must attend your Week 12 lab session to present your work in a formal assessment with your tutor. This session will be conducted as a live code review to verify your work and understanding. Be prepared.

The demonstration will follow this structure:

1. **Automated Test Execution:** You will run your test suite via the php artisan test command. The tutor will observe the output to get an initial assessment of your application's correctness.
2. **Live Functionality Demonstration:** You will walk the tutor through the features of your application as directed by your tutor.
3. **Code Review and Q&A:** The tutor will ask you to navigate to specific parts of your codebase and explain your implementation.

Failure to attend to demonstrate and explain your work will result in your submission being marked as incomplete submission and you will not receive a mark.

Documentation, Self-assessment & Submission

Documentation (Submit as a single PDF file, 2 pages max, excluding appendix):

1. An **ER diagram** for your database, clearly highlighting additions/modifications for your chosen feature.
2. **Reflection:** Describe your development process, challenges, and solutions.
3. **Chosen Feature (Requirement 10) Design & Implementation:**
   - **A. Feature Choice:** State clearly which feature area you chose (A: Waiting List, or B: Categories/Tags).
   - **B. Database Design & Justification:**
     - Describe the database changes (new tables, columns, relationships) you implemented for the feature, referencing your ERD.
     - Justify *why* you chose this specific database structure. For example, if implementing categories, explain why a many-to-many relationship was or was not the appropriate choice.
   - **C. Student-Designed Excellence Marker:**
     - Describe the custom excellence marker you designed and implemented.

    - Explain how your feature meets the criteria of an "excellent" feature by justifying it against one or more of the characteristics defined in the specification (e.g., Improves User Experience, Automates a Process, etc.).

- **D. Implementation Overview:**
    - Briefly explain how your overall feature works, from the user's perspective. You can reference key code components (e.g., specific Controllers, Models, Policies) to support your explanation. You may also include simple sketches of key UI elements if they help clarify your design.

Self-assessment

Self-assess again requirements 1 to 19 in the rubric. Award yourself full mark for that requirement if you have correctly implemented the requirements AND it passes the test. Otherwise, award yourself 0.

Submission Requirements:

1. A compressed **(.zip)** file containing ALL source files.
2. A **PDF** file containing your documentation.
3. An **Excel** file containing your self-assessment rubric. You only need to self-assess against requirements 1 to 19.