IAB207 Assignment 1:

Event Management System Design Report

Submitted by Ananya Aastha

Student no: N12125547

Date 05.09.2025

Table of Contents

1.DESCRIPTION OF SITE 3

2. TECHNOLOGY CONTEXT 4

3. USER STORIES 6

4. CONCEPTUAL MODEL 7

REFERENCE: 14

APPENDIX 1: USE OF AI DECLARATION IN IAB207 15

1.Description of Site

The event management system I chose will be focusing on Hackathons and Tech Workshops for enthusiast students and new people in the industry, across Brisbane. The website would be used as a common platform providing a space for browsing events within various categories such as Web Development, Cybersecurity, Data Science and Networks. A prominent feature of the website is also a comment system where people can post their questions.

Each event listing will include specific metadata such as:

- **Skill level** (e.g., Beginner, Intermediate, Advanced)
- **Prerequisites** (e.g., Laptop required, prior knowledge of Python)
- **Venue and location details**
- **Maximum capacity and available tickets**
- **Event status** (Open, Inactive, Sold Out, Cancelled)

The comment section would make the queries of users resolve quickly and also give a space to discuss about the event with fellow users. Users who are registered will be able to also create and manage their own events, while users who are not registered will be able to browse events, purchase tickets and view their bookings as visitors.

The design demonstrates:

- **Responsive Web Design** (usable across all devices with internet browser) (Marcotte, 2011).
- **Transparent event status** markers for making decisions quickly.
- **User-friendly experience** tailored to various skill levels.

Users can match their skill levels and prerequisites to make informed choices about the events which make their experience more personalized and engaging.

2. Technology Context

2.1 Web Application (Chosen Platform)

Pros:

- No installation required, just need an internet browser.
- Usable across all devices (phones, tablets, laptops).
- Easy to maintain and update for users and developers both.
- Demonstrates responsive design and modern frameworks like bootstrap.

Cons:

- Always needs a stable internet connection.
- None or very limited offline functionality.
- Performance depended on bandwidths; lower ones can affect optimum performance.

2.2 Native Mobile Application (iOS/Android)

Pros:

- Focused for users who are in a hurry; ideally used for quick bookings.
- Has access to the device features such as camera, pop notifications, and location permissions.
- Works offline with cached data for some features.

Cons:

- Needs to be downloaded and installed from app stores.
- Development cost tends to be higher because of different apps for different devices (like android and iOS).
- Maintaining and updating the app depends on user compliance with the app store updates.

2.3 Native Desktop Application (Windows/macOS)

Pros:

- Deeply integrates with desktop systems and can provide a powerful experience.
- Very useful for managing large volumes of data and reports.
- Can work offline as well.

Cons:

- Needs to be downloaded and installed according to the recent versions.
- Limited accessibility for users compared to website and mobile apps, as it requires to be run on a very specific operating system.
- Not feasible for casual attendees who primarily book on mobile apps.

2.4 Platform Comparison and Justification

When compared:

- **Web Applications** can reach widest range of users without barriers like installation and are very accessible and inclusive (Marcotte, 2011; Bootstrap, 2023).
- **Mobile Applications** gives more personalized experience and deep integration with the device but requires high investments and are hard to maintain across all platforms.
- **Desktop Application** are powerful fir back-end admin use but impractical for casual attendees who would like convenience and mobility (Chen, Hsu, & Huang, 2016).

As we know our target audience, which is students and early professionals who want quick, without barrier experience – the **Web Application** platform is the only appropriate choice for this event management system.

3. User Stories

The following user stories have been identified based on the requirements of the Hackathon and Tech Workshop Event Management System. These stories reflect the needs of both visitors and registered users, ensuring that the platform supports browsing, booking, event creation, and community interaction.

As a Visitor

- US1: As a visitor, I want to browse all upcoming hackathon and workshop events so that I can explore opportunities without registering.
- US2: As a visitor, I want to view the details of a selected event (e.g., description, date, skill level, prerequisites, availability) so that I can decide whether to attend.
- US3: As a visitor, I want to filter or search events by categories (e.g., Web, Security, Data, Networks) so that I can quickly find events of interest.
- US4: As a visitor, I want to read comments and questions posted by others on event pages so that I can understand feedback and discussion.

As a Registered User

- US5: As a registered user, I want to create an account with my name, email, phone number, and password so that I can access additional features like booking and event management.
- US6: As a registered user, I want to log in and log out securely so that my personal information and bookings are protected.
- US7: As a registered user, I want to book tickets for events by selecting the number of tickets so that I can reserve my place.
- US8: As a registered user, I want to view my booking history (event name, booking ID, date/time) so that I can track past and upcoming events.
- US9: As a registered user, I want to post comments or questions on event pages so that I can interact with organizers and other participants.
- US10: As a registered user, I want to create, update, or cancel my own events so that I can share opportunities with others and keep details accurate

4. Conceptual Model

This section derives the key concepts from the user stories, refines them into a coherent set of classes, assigns responsibilities, and presents the final UML class model (with attributes, operations, relationships, and multiplicities).

4.1 Concepts Identified from User Stories

From US1–US10, the following candidate concepts emerge:

- **User**, **Visitor** (implicit/unauthenticated), **RegisteredUser** (authenticated)
- **Event**, **EventStatus** (Open/Inactive/SoldOut/Cancelled), **Category**
- **Booking/Order**, **Ticket** (quantity), **BookingID**
- **Comment**
- **Venue** (location details, capacity)
- **Search/Filter** (by category/skill level)
- **SkillLevel**, **Prerequisite** (metadata)
- **Image/Media**
- **Audit data** (createdAt, updatedAt)

4.2 Refined Class List

To keep the model lean and aligned to the assignment:

- **User** (covers both "registered user" and event creator; visitors are simply not persisted/authenticated)
- **Event** (core aggregate for all event data)
- **Category** (taxonomy for browsing/filtering)
- **Venue** (normalised location data; reusable across events)
- **TicketOrder** (represents a booking with quantity and generated orderId)
- **Comment** (discussion on events)
- **Value objects / enums:**

    - **EventStatus** = {Open, Inactive, SoldOut, Cancelled} (derived/controlled by application rules)
    - **SkillLevel** = {Beginner, Intermediate, Advanced}
    - **Prerequisite** (modeled as simple text list on Event for Part 2; can be its own class later if needed)

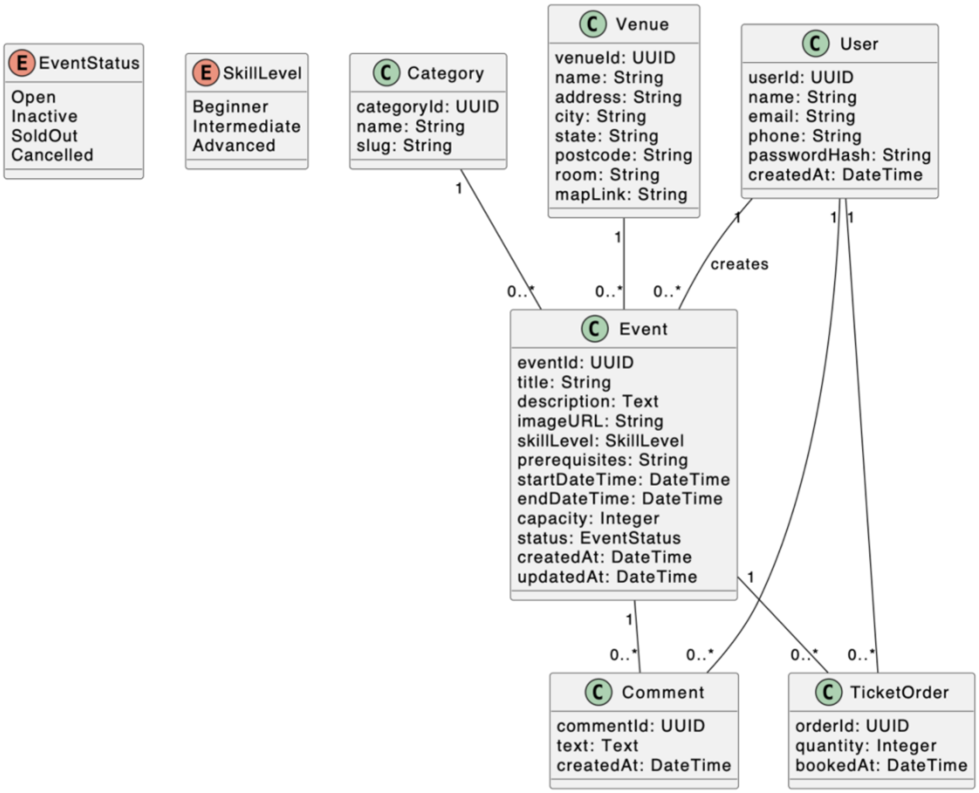
Removed/merged: “RegisteredUser” (folded into User with authentication fields), “Ticket” (captured as quantity within TicketOrder), “Search/Filter” (behavior, not a domain entity), “Image/Media” (kept as URL on Event).

4.3 Class Responsibilities

- **User**
    - Manage profile (name, email, phone, passwordHash)
    - Create/update/cancel **own** events
    - Book tickets (create **TicketOrder**)
    - Post **Comment**
- **Event**
    - Hold event details (title, description, dates/times, capacity, skill level, prerequisites, imageURL)
    - Belong to one **Category**; occur at one **Venue**
    - Track ticketsAvailable (derived from capacity − sum(orders.quantity))
    - Expose computed **status** (Open/Inactive/SoldOut; Cancelled set by creator)
    - Provide rules: reject overbooking; mark SoldOut when ticketsAvailable = 0
- **Category**
    - Name/slug for filtering; associated with many **Event**
- **Venue**
    - Name, address, city, state, postcode, room, mapLink; optional capacity override
- **TicketOrder**

  - Persist booking (orderId, quantity, bookedAt)
  - Link **User** ↔ **Event**
  - Validate quantity (must be > 0 and ≤ ticketsAvailable at booking time)
- **Comment**
  - Persist message (text, createdAt)
  - Link **User** ↔ **Event**
  - Visible to all; creation requires login
- **EventStatus (enum)**
  - Derived: Inactive if eventDate < today; SoldOut if ticketsAvailable = 0; Open otherwise; Cancelled only by creator action

4.4 UML Class Diagram



Data Model (UML-style, with PK/FK)

User

- **PK** user_id: UUID
- name: String
- email: String (unique)
- phone: String
- password_hash: String
- created_at: DateTime

Venue

- **PK** venue_id: UUID
- name: String
- address: String
- city: String
- state: String
- postcode: String
- room: String
- map_link: String

Category

- **PK** category_id: UUID
- name: String
- slug: String (unique)

SkillLevel (lookup)

- **PK** level_code: String *(values: Beginner, Intermediate, Advanced)*
- label: String

EventStatus (lookup)

- **PK** status_code: String *(values: Open, Inactive, SoldOut, Cancelled)*

- label: String

Event

- **PK** event_id: UUID
- **FK** creator_id: UUID → User.user_id
- **FK** venue_id: UUID → Venue.venue_id
- **FK** category_id: UUID → Category.category_id
- **FK** skill_level: String → SkillLevel.level_code
- **FK** status: String → EventStatus.status_code
- title: String
- description: Text
- image_url: String
- prerequisites: String
- start_datetime: DateTime
- end_datetime: DateTime
- capacity: Integer
- created_at: DateTime
- updated_at: DateTime

TicketOrder

- **PK** order_id: UUID
- **FK** user_id: UUID → User.user_id
- **FK** event_id: UUID → Event.event_id
- quantity: Integer
- booked_at: DateTime

Comment

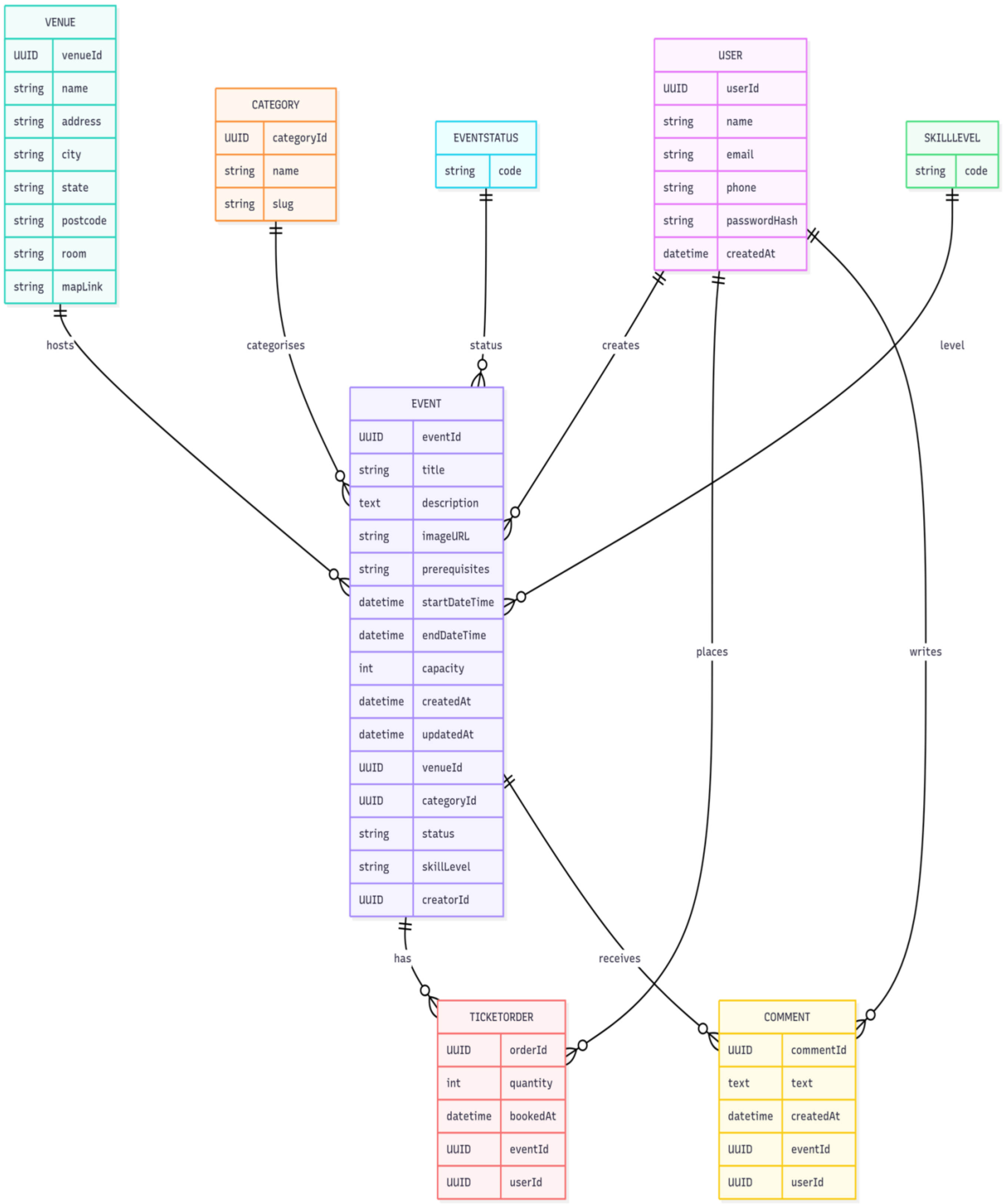
- **PK** comment_id: UUID
- **FK** event_id: UUID → Event.event_id

- **FK** user_id: UUID → User.user_id
- text: Text
- created_at: DateTime

Cardinalities

- User (1) —— (0..*) Event via creator_id
- Venue (1) —— (0..*) Event
- Category (1) —— (0..*) Event
- Event (1) —— (0..*) TicketOrder
- User (1) —— (0..*) TicketOrder
- Event (1) —— (0..*) Comment
- User (1) —— (0..*) Comment

Data Model UML Diagram


VENUE
UUID venueId
string name
string address
string city
string state
string postcode
string room
string mapLink
CATEGORY
UUID categoryId
string name
string slug
EVENTSTATUS
string code
USER
UUID userId
string name
string email
string phone
string passwordHash
datetime createdAt
SKILLLEVEL
string code
hosts
categorises
status
creates
level
EVENT
UUID eventId
string title
text description
string imageURL
string prerequisites
datetime startDateTime
datetime endDateTime
int capacity
datetime createdAt
datetime updatedAt
UUID venueId
UUID categoryId
string status
string skillLevel
UUID creatorId
places
writes
has
receives
TICKETORDER
UUID orderId
int quantity
datetime bookedAt
UUID eventId
UUID userId
COMMENT
UUID commentId
text text
datetime createdAt
UUID eventId
UUID userId

Reference:

Bootstrap. (2023). *Introduction*. In *Bootstrap documentation* (Version 5.3). https://getbootstrap.com/docs/5.3/getting-started/introduction/

Chen, B., Hsu, H.-P., & Huang, Y.-L. (2016). Bringing desktop applications to the web. *IT Professional, 18*(1), 34–40. https://ieeexplore.ieee.org/document/7389272

Marcotte, E. (2011). *Responsive web design*. A Book Apart.

Mozilla Developer Network. (2023). *EventTarget.addEventListener() – Web APIs*. Mozilla Foundation. https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

Appendix 1: Use of AI Declaration in IAB207

Section A:
I confirm that I used AI in an authorized way (as described by the assignment brief).

Section B:
I confirm that I used AI in an authorized way (as described by the assignment brief).

Summary of AI use:

AI assisted me in formatting my report and taught me some concepts of CSS and Bootstrap for the website prototype. Additionally, I used AI to explain basic code and clarify class material. However, I made all the final choices regarding the writing, design, and coding. I also checked my grammar and spelling using Grammarly.

Key examples of AI use:

Your marker needs to clearly understand how you used AI to improve your work. Provide representative samples of how you used AI using the requested format.

{

- **Prompt to AI**: "Format my report headings, subheadings, and bullet points in a professional way according to the assignment requirements."
- **Input provided**: Draft text of my IAB207 report.
- **Output received**: “Use numbered headings (e.g., 1.0, 1.1) for major sections like Introduction, Technology Context, User Stories, Conceptual Model, Data Model, and Appendices. Subheadings should follow a logical hierarchy (e.g., 2.1, 2.2). Use single-line spacing, 12pt sans-serif font (Calibri, Aptos, Helvetica). Bullet points should be used for lists like pros/cons or user story points. Keep consistency across the whole document.”

}

======================== NEXT SAMPLE OF AI USE ====================

{

- **Prompt to AI**: "Explain how to use Bootstrap's grid system and responsive design in simple words."
- **Input provided**: No additional input besides the question.
- **Output received**: "Bootstrap uses a 12-column grid. You divide your page into rows and columns with classes like col-12, col-md-6, col-lg-4. That means on small screens it takes full width, on medium it takes half, and on large it takes one-third. Bootstrap automatically stacks or resizes these columns depending on screen size, so your site looks good on desktop, tablet, and mobile without writing extra CSS."

}

======================== NEXT SAMPLE OF AI USE ====================

{

- **Prompt to AI**: "Explain a tiny JS script that filters event cards when I click a category button."
- **Input provided**: Context of the event filtering feature.
- **Output received**: "You add a click event listener to each category button. When clicked, the script loops through all event cards and checks their data-category attribute. If the card matches the button's category, you keep it visible (.style.display = 'block'). If it doesn't match, you hide it (.style.display = 'none'). This way clicking 'Web' only shows web events, 'Security' shows security events, etc."

}
