**DESIGN DOCUMENT OF EVENT VENUE BOOKING SYSTEM**

A Design Document Presented to the

Faculty of Datamex College of Saint Adeline, Inc.

In Partial Fulfillment of the Requirements for the

Degree of Bachelor of Science in Information Technology

By:

Albania, John Maverick B.

September 2025

**DESIGN DOCUMENTATION**

**INTRODUCTION**

The purpose of this document is to describe the design of the Event Venue Booking System. It offers a clear guide for developers, stakeholders, and technical reviewers to understand how the system will be built, implemented, and maintained.

The Event Venue Booking System is a web application that allows customers to book venues online without needing to create an account. Clients simply choose a venue, complete the booking form, and receive booking confirmations via email. Administrators and staff handle bookings, approve or deny requests, and keep track of venue usage through a centralized backend system.

This document outlines the system's architecture, database schema, user interface design, component interactions, data flows, and security measures. It also includes guidelines for deployment, maintenance strategies, and performance considerations to ensure that the system is reliable, scalable, and easy to use.

**SYSTEM ARCHITECTURE**

**Architecture Type**

Web-based Client

**High-level components**

* Client-side Website for booking submission
* Server-side Booking management, calendar, admin/staff dashboard
* Database stores bookings, users, venues
* Email Notification Service (sends booking confirmation/updates)

**Deployment**

Cloud-based hosting or on-premise server for City of Dreams Manila.

**Communication Protocols**

HTTP for web communication, SMTP for email notifications.

**Interfaces**

Web browser for clients, secured login portal for admin and staff.

**DATABASE DESIGN**

**Entity-relationship diagram (ERD)**

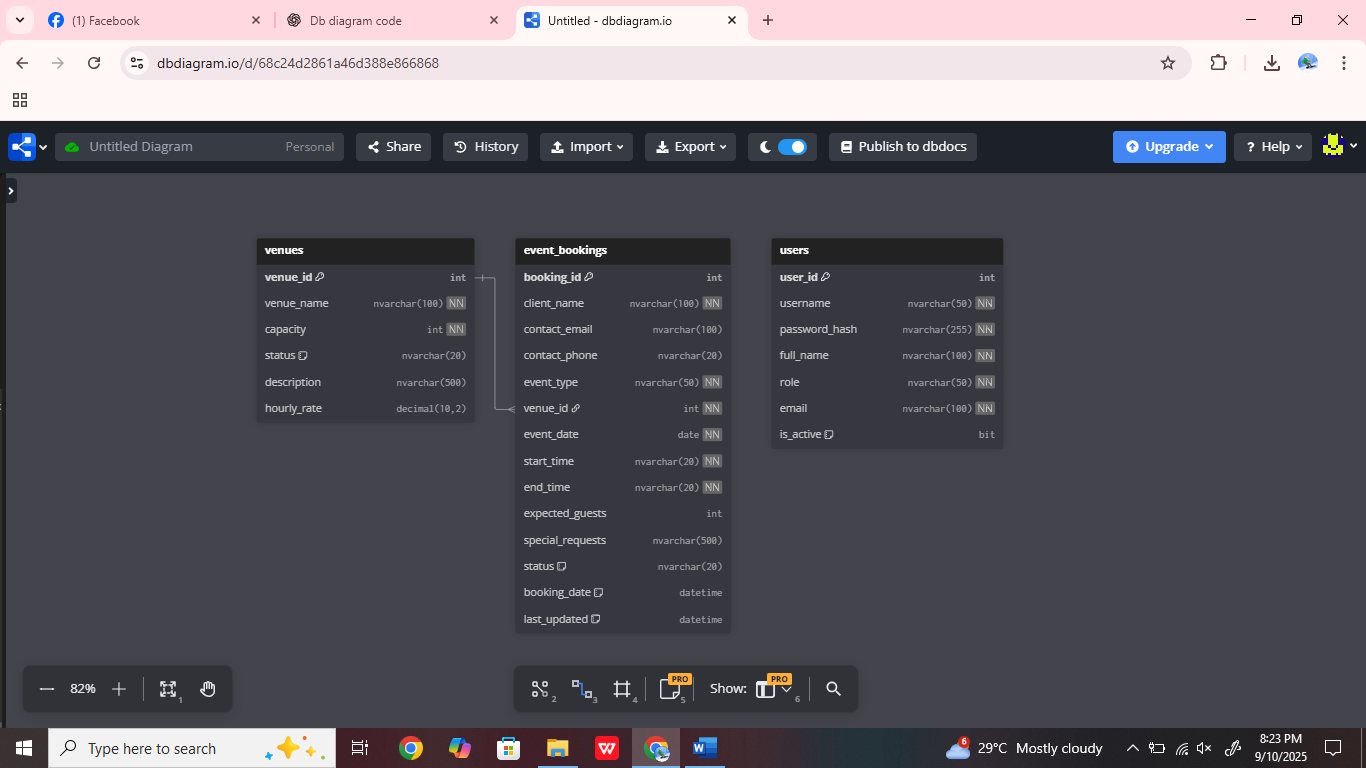


Image 1: Entity-relationship diagram (ERD)

**Description of Database Tables, Fields, and Relationships**

**Core Content Tables**

* **Venues** - stores information about all event venues managed by the system. Each venue has a unique ID, name, seating capacity, description, hourly rental rate, and availability status (Available or Under Maintenance). This table ensures that venue details are consistent and can be referenced when creating bookings.
* **Event Bookings** - captures all client reservations. It stores customer details (name, email, phone), event information (type, date, time, expected number of guests, and special requests), and the booking status (Pending, Confirmed, or Cancelled). Each booking references a venue through a foreign key (venue\_id), which links the reservation to a specific location. The table also records timestamps for when a booking is created and last updated.
* **Users** - contains system login and account information for administrators and staff. Fields include username, hashed password, full name, role (Admin, Event Coordinator, Front Desk, Manager, Setup Team), email, and active status. This table controls authentication and enforces role-based access across the system.

A venue can be linked to many event bookings, but each booking is always tied to a single venue. Users manage the system by creating, updating, or monitoring bookings and venues, with their level of access determined by their role. Administrators have full control, while staff accounts have restricted permissions. Unlike other systems that maintain a dedicated client table, this design embeds client details directly in the Event Bookings table, which simplifies the booking process since clients are not required to create accounts.

**Data Normalization Techniques Used**

The database design adheres to the principles of Third Normal Form (3NF). At the First Normal Form (1NF), all fields contain atomic values with no repeating groups. At the Second Normal Form (2NF), every non-key attribute depends entirely on its primary key, such as booking details depending only on booking\_id. Finally, at the Third Normal Form (3NF), transitive dependencies are eliminated. Venue details, for example, are stored only in the Venues table and referenced through venue\_id in the Event Bookings table, which prevents data redundancy. This approach, combined with embedding client details directly in the bookings table, balances simplicity with data integrity.

**USER INTERFACE DESIGN**

**Wire Frame**

City Of Dreams Manila

Hero Image Here!

City Of Dream Manila

Book Now

Venues

Home City Of Dream Manila

Book Your Event Now

**Shape 1: Wire Frame for Landing Page**

City Of Dreams Manila

Book Now

Venues

Home City Of Dream Manila

Image

Content

Image

Content

Image

Content

Select Venue

Select Venue

Select Venue

**Shape 2: Wire Frame for Venues Page**

City Of Dreams Manila

Book Now

Venues

Home City Of Dream Manila

Book Your Event

Booking Form here

**Shape 3: Wire Frame for Booking Page**

**Admin Side**

Admin login

Username

Password

Login

**Shape 4: Wire Frame for Admin- Login Page**

Log Out

Nav

Dashboard Overview

Upcoming Events

Pending Approval

Today Events

Recent Booking Table Here

**Shape 5: Wire Frame for Admin Dashboard Page**

Log Out

Nav

Event Calendar

Month Week Day

Date Here

Today

t

< >

Calendar Here

**Shape 6: Wire Frame for Event Calendar Page**

Log Out

Nav

Manage Bookings

To Date

From Date

Venues

Status

Reset

Apply Felter

Table Here

**Admin- Venue Page**

**Shape 7: Wire Frame for Manage Booking Page**

Log Out

Nav

Venue Management

Table Here

**Shape 8: Wire Frame for Venue Management Page**

Log Out

Nav

Venue Utilization Report

Generate Report

End Date

Start Date

Table Here

**Admin-Report (Event Type)**

**Shape 9: Wire Frame for Venue Utilization Report**

Log Out

Nav

Event Type Report

Generate Report

End Date

Start Date

Table Here

**Shape 10: Wire Frame for Event Type Report**

Log Out

Nav

Change Password

Form Here

**Shape 11: Wire Frame for Setting Page**

**Client Side**

**Landing Page**: Entry point of the website. Displays system name, introduction, and navigation to venues.

**Venues Page**: Shows available venues with details capacity, description, hourly rate, availability.

**Book Now Page**: Booking form where clients enter personal information name, email, contact, select date, time, and submit booking request.

**Admin Side**

**Login Page**: Secure login form for administrators.

**Dashboard Overview**: Summary of system activity total bookings, pending requests, upcoming events.

**Calendar**: Centralized calendar displaying all bookings with color-coded status Pending, Confirmed, Cancelled.

**Bookings**: Detailed list of all booking requests. Admin can approve, reject, or cancel requests.

**Venues**: Manage venues add, edit, delete, update status/availability.

**Reports**: Generate reports such as venue utilization and event type frequency.

**Venue Utilization**: Visual/statistical representation of venue usage over time.

**Event Types**: Categorization and management of event types weddings, seminars, meetings.

**Settings (Change Password)**: Admin can update their account credentials for security.

**Staff Side**

**Login Page**: Secure login for staff users.

**Dashboard Overview**: Summary of assigned bookings and upcoming events.

**Calendar**: Centralized calendar showing bookings view-only or limited edit.

**View Bookings**: Staff can check booking details and status updates.

**Bookings**: Limited access to booking management update notes, assist in scheduling.

**Venues**: Staff can view venue details and availability but with restricted editing compared to Admin.

**Design Principle and Usability Consideration**

The responsive system is designed so that pages are viewed well on a variety of devices such as computers, tablets, and phones. It also maintains consistency by using the same layout style for headers, buttons, and forms on the client.

For clarify navigation has been simple by using a sidebar menu for admin and staff only. The system also takes accessibility into account by adding form validation, which ensures that inputs such as email and date are entered in the correct format before they are submitted.

**COMPONENT DESIGN**

**Interface Specifications for Each Component**

**API Endpoints**

The backend exposes RESTful endpoints to support frontend operations. Example routes include POST /api/bookings for creating new reservations, GET /api/bookings/:id for retrieving booking details, PATCH /api/bookings/:id for updating status, and DELETE /api/bookings/:id for cancellations. Venue management endpoints include GET /api/venues for listing available venues and POST /api/venues for adding new ones. Authentication endpoints such as POST /api/login validate user credentials and return session tokens.

**Frontend-Backend Interaction**

Frontend components communicate with the backend via AJAX or Fetch API requests. The booking form, for instance, sends validated client data through a POST /api/bookings request, while the venue display retrieves available venues through GET /api/venues. The admin dashboard fetches booking summaries, venue utilization, and system alerts through reporting endpoints.

**Dependency Management and Interaction Between Components**

Frontend dependencies are managed with npm for handling JavaScript libraries and build tools, while styling is supported by standard CSS frameworks for responsiveness. Backend dependencies include Express.js for routing, bcrypt for password hashing, and nodemailer for sending email notifications. SQL Server serves as the primary data store, with queries executed through the mssql driver. The system components interact through a clear separation of concerns: the frontend handles presentation and user interaction, the backend processes requests and enforces business rules, and the database ensures data persistence and integrity.

**DATA FLOW DIAGRAMS**

**Data flow diagrams (DFDs)**

Admin

Process

Client

Database (event\_bookings)

**Shape 12: data flow diagram**

**Data Sources**

* Client inputs booking details: name, email, contact, event info, date, and time.
* Admin provides approval or rejection of the booking.

**Processing Logic**

* System validates client input and checks venue availability.
* System saves the booking in the event\_bookings table with status set to Pending.
* Admin updates booking status Approved, Rejected, Cancelled.
* System sends an email notification to the client after the status update.

**Destinations**

* Booking data is stored in the event\_bookings table.
* Updated status is reflected in the Admin Dashboard and Calendar.
* An email notification is sent to the client as confirmation or rejection.

**SECURETY DESIGN**

**Overview of security requirements and considerations.**

The Event Venue Booking System requires security to ensure confidentiality, integrity, and availability of data. Security considerations include protecting client booking details, safeguarding staff and administrator credentials, and preventing unauthorized access to system functions.

**Authentication and authorization mechanisms.**

**Authentication**:

* Admins and staff log in using unique usernames and password hashes bcrypt/hashed passwords.
* No client accounts are created; clients only provide booking details via the public booking form.

**Authorization**:

* **Admin**: Full system access manage bookings, venues, reports, staff accounts.
* **Staff**: Limited access view bookings, update notes, assist with scheduling.
* **Client**: Public access only for submitting booking requests no login required.

**Data Encryption and Protection Measures**

Database passwords are hashed bcrypt so they cannot be hidden in plain text. All client and booking information sent over HTTPS to keep data secure while being transmitted online. Access to the database is limited to authorized users to prevent unauthorized access.

The system also applies input validation and sanitization to prevent common attacks such as SQL injection. In addition, session management is used for admin and staff accounts to ensure that no one else can use their sessions.

**PERFORMANCE DESIGN**

**Performance requirements and objectives**

The system must be able to handle multiple booking requests simultaneously without downtime. Client bookings should be processed within 2-3 seconds, while admin dashboards and reports should load in under 5 seconds. The system should remain responsive even as the number of venues and booking records increases.

**Strategies for optimizing system performance**

To maintain performance, the system will implement caching of frequently accessed data such as venue availability. Database queries will be optimized using indexing on booking dates, venue IDs, and statuses. Pagination will be applied on large booking lists and reports to reduce server load. In future deployments, the system can be scaled with cloud hosting and load balancers.

**Performance Testing Plan**

Performance will be validated through load testing, stress testing, and response time measurement. Load testing will simulate multiple clients submitting bookings at the same time, while stress testing will push the system beyond expected traffic to identify bottlenecks. Response time tests will measure the time needed for booking submissions, approval updates, and report generation. Database performance testing will also be conducted to monitor and optimize query execution times.

**ERROR HANDILING AND LOGGING**

**Error handling mechanisms and strategies**

The system employs robust error handling to ensure smooth user experience and system reliability. Input validation is enforced on all forms (e.g., valid email format, required fields, date and time validation) to prevent invalid data entry. In case of errors, user-friendly messages are displayed without exposing technical details. Critical operations such as booking submission, venue management, and approval workflows are wrapped in try-catch blocks to ensure system stability.

**Logging Requirements and Specifications**

* All system activities login attempts, booking submissions, status updates, venue changes are logged with a timestamp and user ID.
* Logs are stored securely in the system database and are accessible only by administrators.
* Logging supports troubleshooting, monitoring usage trends, and auditing user actions for accountability.
* Failed login attempts and unexpected system errors are highlighted for security monitoring.

**Error Codes and Messages**

Standardized error codes will be used: 400 Bad Request for invalid client input, 401 (Authentication Failure) for invalid login credentials, 403 Authorization Error when a user tries to access a restricted function, 404 Not Found when a requested resource (e.g., venue or booking) does not exist, and 500 Internal Server Error for unexpected server or database failures. User-facing messages will be simple and helpful (for example, “Please complete all required fields” or “Selected venue is not available for the chosen date and time”), while technical details stack traces, SQL errors are logged securely for the development/ops team to debug. API responses will include a short machine-readable code and a human message so the frontend can show appropriate feedback to the user.

**THIRD-PARTY INTEGRATION**

**List of third-party services or APIs integrated into the system**

* Email Service (SMTP or third-party provider like SendGrid/Gmail SMTP) - used to send booking confirmations, approval/rejection notifications, and status updates to clients.
* Calendar Integration optional, future - synchronization with third-party calendar services (e.g., Google Calendar, Outlook) to display confirmed bookings.

**Description of Integration Points and Data Exchange Formats**

**Email Notifications**

* Integration Point: Booking Confirmation and Admin Updates.
* Data Exchange: System generates structured email messages in plain text or HTML format, containing booking details and status updates.
* Example: Booking ID, Venue, Date/Time, Status.

**Calendar (Future Expansion)**

* Integration Point: Admin Dashboard - Export Bookings to Calendar.
* Data Exchange: Standard iCal/ICS format for compatibility with external calendars.

**DEPLOYMENT PLAN**

**Overview of the Deployment Process**

The Event Venue Booking System will be deployed in a staged approach to ensure stability and minimize disruption. The deployment begins with the installation of the web application on a secure server environment, followed by database configuration and integration of required third-party services, such as the email notification system. After successful internal testing, the system will be rolled out for live use by authorized administrators and staff.

**Hardware and Software Requirements for Deployment**

The system requires a server with a minimum 8 GB RAM, quad-core processor, and at least 100 GB storage to handle database operations and concurrent requests. It is deployed on a Windows or Linux environment with support for IIS or Apache web servers. The database uses Microsoft SQL Server, and the web application runs on standard web technologies (HTML, CSS, JavaScript, and Node Js). Client access requires only a modern browser.

**Configuration Management and Version Control Procedures**

Source code and configuration files are managed through a version control system such as Git. This ensures that updates are properly tracked, with changes documented and reviewed before merging into the production branch. Configuration settings, such as database connection strings and email service credentials, are stored securely and separated from the main source code to prevent unauthorized exposure.

**MAINTENANCE AND SUPPORT**

**Guidelines for System Maintenance and Support**

Regular maintenance will be performed to ensure that the system remains secure, reliable, and efficient. This includes monitoring server performance, ensuring database integrity, and validating that the booking process functions correctly. Scheduled downtime for maintenance will be communicated to users in advance.

**Procedures for Handling Software Updates, Patches, and Bug Fixes**

All updates and patches are first applied in a testing environment to validate compatibility and stability before being deployed to the live system. Bug reports are logged and categorized by severity. Critical issues are addressed immediately, while minor issues and enhancements are scheduled into regular update cycles.

**Escalation Process for Resolving Issues**

In the event of a system issue, support staff will provide first-level troubleshooting. If the problem cannot be resolved, the issue is escalated to the system administrator for advanced diagnosis. For critical or recurring problems, the matter is forwarded to the development team for permanent fixes. This structured escalation process ensures that issues are resolved efficiently and with minimal impact on system availability.

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Description of Changes** | **Author** |
| 1.0 | 2025-07-20 | Initial draft of Design Documentation | John Maverick B. Albania |
| |  | | --- | | 1.1 |  |  | | --- | |  | | 2025-07-30 | Added Database Design and DFD diagrams | John Maverick B. Albania |
| 1.2 | 2025-08-15 | Completed Security, Performance, and Deployment sections | John Maverick B. Albania |

**Table 1: Revision History**

**APPENDIX**

The appendix includes supporting documentation that complements the design specification:

* **Diagrams:** Entity-Relationship Diagram ERD, Data Flow Diagrams DFDs, Component Diagram, and User Interface mockups.
* **Reference Materials**: Project Proposal, Requirement Specification Document, and research materials related to event booking systems.
* **Tables and Screenshots**: Database schema tables, system wireframes, and performance test plans.