

# DLSU Classroom Finder

## Project Documentation

---

### Executive Summary

**DLSU Classroom Finder** is a web-based platform designed to help De La Salle University students locate and book available classrooms for studying. The system provides real-time classroom availability status, interactive heat maps, and a booking system with role-based access control. This project demonstrates secure web development practices including authentication, authorization, input validation, and comprehensive audit logging.

---

### Table of Contents

1. Problem Statement
  2. Solution Overview
  3. User Roles & Permissions
  4. Core Features
  5. Technical Architecture
  6. Database Schema
  7. Security Implementation
  8. User Interface & Pages
  9. Implementation Roadmap
  10. Future Enhancements
- 

### Problem Statement

Students at DLSU frequently struggle to find available classrooms for individual or group study sessions. Current challenges include:

- No centralized system to check classroom availability
- Time wasted wandering buildings looking for empty rooms
- Uncertainty about how long a room will remain available
- Conflicts between student study sessions and scheduled classes

- No way to reserve spaces in advance
- 

## Solution Overview

DLSU Classroom Finder provides:

1. **Real-time Availability Tracking:** Live status of all classrooms across campus
  2. **Interactive Heat Maps:** Visual representation of building occupancy by floor
  3. **Smart Booking System:** Reserve classrooms with conflict prevention
  4. **Calendar Integration:** View class schedules and available time slots
  5. **Role-Based Management:** Different privilege levels for students, faculty, and administrators
- 

## User Roles & Permissions

### User (Student)

#### Capabilities:

- View real-time classroom availability across all buildings
- Search and filter rooms by building, capacity, and amenities
- Book available classrooms for study sessions
- Check-in when arriving at booked classroom
- Check-out when leaving (releases room early)
- View personal booking history
- Report classroom issues (maintenance, cleanliness)
- Update own profile information

#### Restrictions:

- Cannot book during scheduled class times
- Limited to maximum booking duration (3 hours per session)
- Cannot book more than 7 days in advance
- Daily booking limit to prevent hoarding
- Cannot modify other users' bookings

### Manager (Faculty/Building Coordinator)

Capabilities (All User capabilities plus):

- Create and manage recurring class schedules
- Block classrooms for events or maintenance
- Approve/deny student bookings for premium rooms
- View booking analytics for assigned building(s)
- Manage classroom details (capacity, amenities, status)
- Override student bookings when necessary
- Respond to and resolve student reports
- View booking patterns and usage statistics

**Restrictions:**

- Limited to managing assigned building(s)
- Cannot modify system-wide settings
- Cannot manage admin accounts

## **Admin (University IT/Facilities)**

**Capabilities (All Manager capabilities plus):**

- Full CRUD operations on all classrooms and buildings
- Manage all user accounts and assign roles
- Configure system-wide booking rules and policies
- Access comprehensive audit logs
- View system-wide analytics and usage patterns
- Bulk import class schedules
- Manage all buildings regardless of assignment
- System configuration and settings management

---

## **Core Features**

### **1. Landing Page (Public Access)**

**Purpose:** Introduce the platform and encourage user registration






**Content:**

- Hero section with platform overview
- Key features showcase
- Building directory and quick stats
- Real-time availability preview (limited)
- Call-to-action for login/signup
- Contact information and support links

## 2. Real-Time Heat Map Dashboard

**Purpose:** Visual representation of classroom availability

**Features:**

- Building selection interface
- Floor-by-floor heat map visualization
- Color-coded status indicators:
  -  **Available:** Room is free and bookable
  -  **Booked:** Reserved by student
  -  **Class in Session:** Scheduled class time
  -  **Maintenance:** Under repair/cleaning
  -  **Reserved:** Manager-blocked for events
- Live updates via WebSocket connections
- Room details on hover/click
- Quick booking action from heat map
- Time remaining until next class/booking

**Technical Implementation:**

- Supabase Realtime subscriptions
- Auto-refresh every 30 seconds as fallback
- Optimistic UI updates for instant feedback

## 3. Calendar View

**Purpose:** Schedule-based view of classroom availability

**Features:**

- Week and day view modes
- Filter by building, floor, and room
- Visual indication of recurring classes
- Available time slots clearly marked
- Click-to-book functionality
- Current time indicator
- Recurring pattern visualization
- Export to personal calendar (optional)

## 4. Smart Booking System

**Purpose:** Enable students to reserve classrooms with intelligent conflict prevention

**Booking Rules:**

- Maximum 3-hour booking duration
- Maximum 7 days advance booking
- Daily limit: 2 active bookings per student
- 15-minute grace period between bookings
- Auto-cancellation if no check-in within 15 minutes
- Cannot book during scheduled class times
- Cannot overlap with existing bookings

#### **Booking Workflow:**

1. Student selects room and time slot
2. System validates availability and user limits
3. Booking created with "Pending" status
4. Student receives confirmation
5. Check-in required within 15 minutes of start time
6. Status changes to "Confirmed" after check-in
7. Auto check-out at booking end time
8. Student can manually check-out early

#### **Check-in/Check-out System:**

- QR code at classroom entrance (optional)
- Manual check-in button in app
- Location verification (optional)
- Grace period enforcement
- Room immediately available after check-out

## **5. Recurring Class Schedule Management**

**Purpose:** Managers can define regular class schedules

#### **Features:**

- Create recurring events (daily, weekly patterns)
- Specify days of week (e.g., MWF, TTH)
- Set semester start and end dates
- Edit single instance or entire series
- Conflict detection with other schedules
- Bulk import via CSV
- Exception handling for holidays

#### **Example:**

Course: CSSWENG

Days: Monday, Wednesday, Friday

Time: 10:30 AM - 12:00 PM

Room: GK302-A

Semester: August 1, 2024 - December 15, 2024

## **6. Search & Filter System**

**Purpose:** Help users quickly find suitable rooms

**Filters:**

- Building and floor
- Capacity (minimum seats needed)
- Amenities (projector, AC, whiteboard, computers)
- Availability duration (available for next X hours)
- "Find me a room NOW" quick search

**Search Results:**

- List view with room details
- Map view showing locations
- Sort by availability, capacity, distance
- Save favorite rooms

## **7. User Dashboard**

**Purpose:** Personal control center for students

**Components:**

- Upcoming bookings
- Booking history
- Quick book from favorites
- Active booking status
- Profile management
- Submitted reports status

## **8. Manager Dashboard**

**Purpose:** Management interface for faculty and coordinators

**Components:**

- Building occupancy overview
- Pending booking approvals
- Schedule management interface
- Usage analytics and reports

- Student report queue
- Classroom status management

## 9. Admin Panel

**Purpose:** System-wide administration and configuration

**Components:**

- User management (CRUD, role assignment)
- Building and classroom management
- System configuration
- Audit log viewer
- Analytics dashboard
- Bulk operations interface

## 10. Reports System

**Purpose:** Allow users to report issues with classrooms

**Report Types:**

- Maintenance needed
- Cleanliness issues
- Equipment malfunction
- Safety concerns
- Other

**Workflow:**

1. User submits report with description and photos (optional)
2. Manager receives notification
3. Manager updates status (Open → In Progress → Resolved)
4. User receives resolution notification
5. Report archived in history

## 11. Audit Logging

**Purpose:** Track all system activities for security and accountability

**Logged Actions:**

- User login/logout attempts
- Booking creation, modification, cancellation
- Check-in/check-out events
- Role changes and permission modifications

- Schedule creation and updates
- Room status changes
- Report submissions and resolutions
- Failed authorization attempts

#### Log Details:

- User ID and role
  - Action type and resource
  - Timestamp
  - IP address
  - Before/after states (for modifications)
  - Success/failure status
- 

## Technical Architecture

### Frontend Stack

- **Framework:** Next.js 14 (App Router)
- **Language:** TypeScript
- **Styling:** Tailwind CSS
- **UI Components:** shadcn/ui (optional)
- **State Management:** React Context + useState
- **Real-time:** Supabase Realtime client
- **Charts:** Recharts (for analytics)

### Backend Stack

- **Runtime:** Next.js API Routes
- **Database:** Supabase (PostgreSQL)
- **Authentication:** Supabase Auth
- **File Storage:** Supabase Storage (for report photos)
- **Real-time:** Supabase Realtime subscriptions

### Hosting & Deployment

- **Frontend/Backend:** Vercel
- **Database:** Supabase Cloud (Free Tier)
- **CDN:** Vercel Edge Network

### Development Tools



- **Version Control:** Git + GitHub
  - **Code Quality:** ESLint, Prettier
  - **Testing:** Jest, React Testing Library (optional)
- 

## Database Schema

### Users Table

```
users (  
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),  
  email VARCHAR(255) UNIQUE NOT NULL,  
  role VARCHAR(20) NOT NULL CHECK (role IN ('user', 'manager', 'admin')),  
  name VARCHAR(255) NOT NULL,  
  id_number VARCHAR(50) UNIQUE NOT NULL,  
  department VARCHAR(100),  
  created_at TIMESTAMP DEFAULT NOW(),  
  updated_at TIMESTAMP DEFAULT NOW()  
)
```

### Buildings Table

```
buildings (  
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),  
  name VARCHAR(100) NOT NULL,  
  code VARCHAR(10) UNIQUE NOT NULL,  
  floors INTEGER NOT NULL,  
  created_at TIMESTAMP DEFAULT NOW()  
)
```

### Classrooms Table

```
classrooms (  
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),  
  building_id UUID REFERENCES buildings(id) ON DELETE CASCADE,  
  room_number VARCHAR(20) NOT NULL,  
  floor INTEGER NOT NULL,  
  capacity INTEGER NOT NULL,  
  amenities JSONB DEFAULT '[]',  
  current_status VARCHAR(20) DEFAULT 'available'  
  CHECK (current_status IN ('available', 'occupied', 'maintenance', 'reserved')),  
  last_updated TIMESTAMP DEFAULT NOW(),
```

```
created_at TIMESTAMP DEFAULT NOW(),  
UNIQUE(building_id, room_number)  
)
```

### **Amenities JSON Structure:**

```
{  
  "projector": true,  
  "air_conditioning": true,  
  "whiteboard": true,  
  "computers": 0,  
  "sound_system": false  
}
```

### **Class Schedules Table (Recurring)**

```
class_schedules (  
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),  
  classroom_id UUID REFERENCES classrooms(id) ON DELETE CASCADE,  
  course_code VARCHAR(20) NOT NULL,  
  instructor VARCHAR(255),  
  days_of_week INTEGER[] NOT NULL,  
  start_time TIME NOT NULL,  
  end_time TIME NOT NULL,  
  start_date DATE NOT NULL,  
  end_date DATE NOT NULL,  
  created_by UUID REFERENCES users(id),  
  created_at TIMESTAMP DEFAULT NOW(),  
  updated_at TIMESTAMP DEFAULT NOW()  
)
```

**days\_of\_week:** Array of integers (0=Sunday, 1=Monday, ..., 6=Saturday)

### **Bookings Table**

```
bookings (  
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),  
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,  
  classroom_id UUID REFERENCES classrooms(id) ON DELETE CASCADE,  
  booking_date DATE NOT NULL,  
  start_time TIME NOT NULL,  
  end_time TIME NOT NULL,  
  status VARCHAR(20) DEFAULT 'pending'
```

```

    CHECK (status IN ('pending', 'confirmed', 'checked_in', 'completed', 'cancelled',
'auto_cancelled')),
    checked_in_at TIMESTAMP,
    checked_out_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
)

```

## Reports Table

```

reports (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE SET NULL,
  classroom_id UUID REFERENCES classrooms(id) ON DELETE CASCADE,
  issue_type VARCHAR(50) NOT NULL,
  description TEXT NOT NULL,
  photo_urls JSONB DEFAULT '[]',
  status VARCHAR(20) DEFAULT 'open'
    CHECK (status IN ('open', 'in_progress', 'resolved', 'closed')),
  resolved_by UUID REFERENCES users(id),
  resolved_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT NOW()
)

```

## Audit Logs Table

```

audit_logs (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE SET NULL,
  action VARCHAR(100) NOT NULL,
  resource_type VARCHAR(50) NOT NULL,
  resource_id UUID,
  details JSONB,
  ip_address INET,
  timestamp TIMESTAMP DEFAULT NOW()
)

```

### Example Audit Log Entry:

```

{
  "user_id": "abc-123",
  "action": "booking_created",
  "resource_type": "booking",

```

```
"resource_id": "xyz-789",
"details": {
  "classroom": "GK302-A",
  "date": "2024-10-15",
  "time": "14:00-16:00"
},
"ip_address": "192.168.1.1",
"timestamp": "2024-10-09T10:30:00Z"
}
```

---

# Security Implementation

## 1. Authentication & Authorization

### Authentication:

- Email-based registration with DLSU domain verification (@dlsu.edu.ph)
- Password requirements: minimum 8 characters, uppercase, lowercase, number
- Bcrypt password hashing (handled by Supabase Auth)
- JWT tokens for session management
- Refresh token rotation
- Secure httpOnly cookies for token storage

### Authorization:

- Role-Based Access Control (RBAC)
- Middleware protection for API routes
- Row-Level Security (RLS) policies in Supabase
- Client-side route guards for protected pages

### RLS Policy Examples:

```
-- Users can only view their own bookings
CREATE POLICY "Users view own bookings"
ON bookings FOR SELECT
USING (auth.uid() = user_id);
```

```
-- Managers can view all bookings in their assigned buildings
CREATE POLICY "Managers view building bookings"
ON bookings FOR SELECT
USING (
  EXISTS (
```

```

SELECT 1 FROM classrooms c
JOIN buildings b ON c.building_id = b.id
WHERE c.id = bookings.classroom_id
AND b.id = ANY(manager_assigned_buildings(auth.uid()))
)
);

```

## 2. Input Validation & Sanitization

### Client-Side Validation:

- React Hook Form with Zod schema validation
- Real-time field validation feedback
- Type checking with TypeScript

### Server-Side Validation:

- All API routes validate inputs
- Zod schemas for request body validation
- Sanitization using DOMPurify for user-generated content
- File upload validation (type, size, format)

### Example Validation Schema:

```

const BookingSchema = z.object({
  classroom_id: z.string().uuid(),
  booking_date: z.date().min(new Date()),
  start_time: z.string().regex(/^d{2}:d{2}$/),
  end_time: z.string().regex(/^d{2}:d{2}$/),
}).refine(data => {
  // Ensure end_time is after start_time
  // Ensure duration <= 3 hours
  // Ensure not more than 7 days in advance
});

```

## 3. SQL Injection Prevention

- Supabase uses parameterized queries by default
- Additional validation on all database inputs
- No raw SQL concatenation
- Use of prepared statements for complex queries

## 4. Cross-Site Scripting (XSS) Protection

### **Prevention Measures:**

- Content Security Policy (CSP) headers
- DOMPurify for sanitizing user inputs
- React's built-in XSS protection (JSX escaping)
- Sanitize all data before rendering
- Escape HTML in user-generated content

### **CSP Header Example:**

Content-Security-Policy:

```
default-src 'self';  
script-src 'self' 'unsafe-inline' https://cdn.jsdelivr.net;  
style-src 'self' 'unsafe-inline';  
img-src 'self' data: https;;
```

## **5. Cross-Site Request Forgery (CSRF) Protection**

- CSRF tokens for all state-changing operations
- SameSite cookie attribute
- Origin header validation
- Double-submit cookie pattern

### **Implementation:**

```
// Generate CSRF token on login  
const csrfToken = generateSecureToken();  
// Include in hidden form field or header  
headers['X-CSRF-Token'] = csrfToken;  
// Verify on server  
if (req.headers['x-csrf-token'] !== session.csrfToken) {  
  throw new Error('Invalid CSRF token');  
}
```

## **6. Rate Limiting**

### **Limits:**

- Login attempts: 5 per 15 minutes per IP
- Booking creation: 10 per hour per user
- Search queries: 100 per minute per user
- API requests: 1000 per hour per IP

**Implementation:**

- Use Vercel's rate limiting or Upstash Redis
- Return 429 status code when exceeded
- Exponential backoff for repeated violations

## 7. Session Management

**Security Measures:**

- JWT tokens with short expiration (15 minutes)
- Refresh tokens with longer expiration (7 days)
- Automatic token refresh before expiration
- Revoke tokens on logout
- Secure token storage (httpOnly cookies)
- Session timeout after 30 minutes of inactivity

## 8. Data Privacy

**Measures:**

- Students can only see their own booking history
- Personal information hidden from other users
- Managers only see data for assigned buildings
- Audit logs track all data access
- GDPR-compliant data handling (if applicable)

## 9. Audit Logging

**All Security-Relevant Events:**

- Authentication attempts (success/failure)
- Authorization failures
- Booking operations
- Role changes
- Data modifications
- Suspicious activities

**Log Analysis:**

- Dashboard for admins to review logs
- Automated alerts for suspicious patterns
- Retention policy (90 days minimum)

## 10. Error Handling

### Best Practices:

- Never expose stack traces to users
  - Generic error messages for security failures
  - Detailed logging on server-side
  - Graceful degradation
  - User-friendly error pages
- 

## User Interface & Pages

### Public Pages

#### Landing Page (/)

##### Components:

- Navigation bar with Login/Signup buttons
- Hero section with tagline and search preview
- Feature showcase (Heat Map, Booking, Real-time)
- Building directory with quick stats
- Testimonials or usage statistics
- Footer with links and contact info

#### Login Page (/login)

##### Components:

- Email and password fields
- "Forgot Password" link
- "Sign Up" link
- Error message display
- DLSU branding

#### Signup Page (/signup)

##### Components:

- Name, DLSU email, ID number fields
- Password and confirm password
- Terms and conditions checkbox
- Email verification flow
- Success message with next steps



---

## Authenticated Pages

### Dashboard (</dashboard>)

#### Components:

- Welcome message with user name
- Quick stats (active bookings, upcoming classes)
- "Find a Room Now" quick search
- Upcoming bookings list
- Recent activity feed
- Navigation to other features

### Heat Map View (</heatmap>)

#### Components:

- Building selector dropdown
- Floor selector (tabs or buttons)
- Interactive heat map grid
- Room status legend
- Room details modal on click
- Quick book button
- Auto-refresh indicator
- Filter sidebar (capacity, amenities)

### Calendar View (</calendar>)

#### Components:

- Week/Day view toggle
- Date picker
- Building and room filters
- Time grid with color-coded blocks
- Available slot highlighting
- Click to book interaction
- Legend for status colors
- My bookings overlay

### My Bookings (</my-bookings>)

#### Components:

- Tabs: Upcoming, Past, Cancelled

- Booking cards with details
- Check-in/Check-out buttons
- Cancel booking option
- Booking history timeline
- QR code display (optional)

## Search Rooms ([/search](#))

### Components:

- Search filters form
- "Find Now" instant search
- Results list with room cards
- Map view toggle
- Sort options
- Favorite/bookmark rooms
- Quick view modal

## Profile ([/profile](#))

### Components:

- User information display
  - Edit profile form
  - Change password section
  - Booking preferences
  - Notification settings
  - Account statistics
- 

## Manager Pages

### Manager Dashboard ([/manager](#))

#### Components:

- Building occupancy overview
- Pending approval queue
- Today's schedule summary
- Recent student reports
- Quick actions panel
- Analytics snapshot

### Schedule Management ([/manager/schedules](#))

### Components:

- Calendar view of all schedules
- Create recurring schedule form
- Edit/delete schedule options
- Conflict detection alerts
- Bulk import interface
- Filter by course/instructor

### Room Management ([/manager/rooms](#))

#### Components:

- Room list for assigned buildings
- Edit room details form
- Change room status
- Maintenance scheduling
- Room analytics

### Reports Queue ([/manager/reports](#))

#### Components:

- Report list with filters
  - Report details view
  - Status update interface
  - Response/comment section
  - Photo viewer
  - Resolution workflow
- 

## Admin Pages

### Admin Panel ([/admin](#))

#### Components:

- System-wide statistics dashboard
- User management quick access
- Building management
- System health indicators
- Recent admin actions
- Alerts and notifications

### User Management ([/admin/users](#))

**Components:**

- User list with search and filters
- Role assignment interface
- User details modal
- Create/edit/delete users
- Bulk operations
- Activity log per user

**Building & Classroom Management (/admin/buildings)****Components:**

- Building list
- Add/edit building form
- Classroom CRUD interface
- Bulk import classrooms
- Building floor plans (optional)

**System Configuration (/admin/settings)****Components:**

- Booking rules configuration
- Time limits and restrictions
- System-wide settings
- Email templates
- Integration settings

**Audit Logs (/admin/logs)****Components:**

- Searchable log table
- Filter by user, action, date range
- Export logs functionality
- Log detail viewer
- Pattern detection alerts

**Analytics (/admin/analytics)****Components:**

- Usage statistics charts
- Peak hours analysis
- Most booked rooms

- User engagement metrics
  - Building utilization rates
  - Custom report generator
- 

## Implementation Roadmap

### Phase 1: Foundation (Week 1-2)

#### Deliverables:

- Project setup (Next.js, TypeScript, Tailwind)
- Supabase configuration
- Database schema implementation
- Authentication system
- Basic routing structure
- Landing page

#### Security Focus:

- Password hashing setup
- JWT implementation
- Basic RLS policies

### Phase 2: Core Features (Week 3-4)

#### Deliverables:

- Heat map dashboard
- Real-time status updates
- Basic booking system
- Check-in/check-out functionality
- User dashboard
- Profile management

#### Security Focus:

- Input validation schemas
- CSRF token implementation
- Rate limiting setup
- XSS protection

### Phase 3: Management Features (Week 5)

**Deliverables:**

- Recurring schedule management
- Manager dashboard
- Room status management
- Booking approval workflow

**Security Focus:**

- Role-based authorization
- Advanced RLS policies
- Audit logging implementation

**Phase 4: Advanced Features (Week 6)****Deliverables:**

- Calendar view
- Search and filter system
- Reports system
- Notifications (optional)

**Security Focus:**

- SQL injection testing
- Security audit
- Error handling refinement

**Phase 5: Admin & Analytics (Week 7)****Deliverables:**

- Admin panel
- User management interface
- System configuration
- Analytics dashboard
- Audit log viewer

**Security Focus:**

- Admin action logging
- Privilege escalation testing
- Final security review

**Phase 6: Testing & Deployment (Week 8)**

**Deliverables:**

- Comprehensive testing
- Bug fixes
- Performance optimization
- Documentation
- Deployment to production

**Security Focus:**

- Penetration testing
  - Security checklist validation
  - Incident response plan
- 

## **Future Enhancements**

**Short-term (Post-MVP)**

- Email and push notifications
- Mobile app (React Native)
- QR code check-in system
- Room favorites and saved searches
- Integration with DLSU student portal
- Capacity tracking (number of occupants)

**Medium-term**

- AI-powered room recommendations
- Study group formation and matching
- Equipment booking (projectors, laptops)
- Noise level indicators
- Photo uploads of room conditions
- Integration with campus map app

**Long-term**

- IoT sensors for automatic occupancy detection
- Smart lock integration
- Climate control preferences
- Virtual queuing system
- Cross-campus integration (if multiple campuses)
- Predictive analytics for room availability

---

## Conclusion

DLSU Classroom Finder addresses a real need on campus while providing an excellent platform to demonstrate secure web development practices. The three-tier role system (User, Manager, Admin) naturally incorporates various security concepts including authentication, authorization, input validation, audit logging, and more.

The MVP focuses on delivering core functionality with robust security, while the modular architecture allows for easy addition of future features. By using modern technologies like Next.js and Supabase, the project maintains scalability and performance while keeping development efficient.

This project not only fulfills the course requirements but also creates a potentially useful tool for the DLSU community.

---

**Project Repository:** [To be added]

**Live Demo:** [To be added]

**Documentation:** [To be added]

**Course:** Secure Web Development

**Institution:** De La Salle University

**Academic Year:** 2024-2025