ABSOLUTE CINEMA

Cinema Seat Reservation Web Application

Michel MUSTAFOV, Edwin WEHBE

1. Introduction

Absolute Cinema is a full-featured cinema seat reservation web application built with pure PHP, following a strict Model–View–Controller (MVC) architecture without any external framework. The application enables users to browse films currently showing, view available screening sessions, and reserve seats online.

The system implements a dual-role access model: regular users can manage their accounts and reservations, while administrators have full control over films, screening rooms, sessions, reservations, and user accounts.

This document covers the technical design decisions, architecture, implemented features, security measures, and database schema of the project.

2. Architecture Overview

2.1 MVC Pattern

The project strictly separates concerns following the MVC pattern. No SQL queries appear in views, and no HTML is generated in models. The front controller (index.php) acts as a single entry point that routes all requests to the appropriate controller method.

Layer	Responsibility	Files
Model	Database access via PDO, business logic, data validation	Database.php, UserModel.php, MovieModel.php, RoomModel.php, ScreeningModel.php, ReservationModel.php
View	HTML rendering with escaped output, layout templates (header/footer), user-facing interface	views/layouts/*, views/auth/*, views/movies/*, views/reservations/*, views/admin/*
Controller	Request handling, input validation, session management, routing to views	AuthController.php, MovieController.php, ReservationController.php, AdminController.php

2.2 Project Structure

```
cinema/
├── index.php                # Front controller & router
├── config/database.php       # DB config with .env support
├── sql/schema.sql            # Database creation script
├── app/
│   ├── models/               # PDO-based data access layer
│   ├── controllers/          # Request handlers
│   └── views/                # PHP templates
└── public/
    ├── css/                  # Modular stylesheets
    ├── js/                   # Client-side interactivity
    └── img/                  # Static assets
```

2.3 Routing Mechanism

All HTTP requests are handled by index.php, which reads the action query parameter and dispatches to the correct controller method via a switch statement. This provides a clean, centralized routing mechanism without relying on URL rewriting or .htaccess rules.

Example: `index.php?action=movie&id=3` routes to `MovieController::show()` which fetches movie data and renders the detail view.

3. Database Design

3.1 Entity-Relationship Model

The database uses five tables with foreign key constraints and cascading deletes to maintain referential integrity. All tables use utf8mb4 for full Unicode support.

Table	Purpose	Key Fields
users	User accounts with role-based access	id, username, email, password (bcrypt), role (enum: user/admin)
movies	Film catalog with metadata	id, title, description, duration (minutes)
rooms	Screening rooms with configurable capacity	id, name (unique), capacity
screenings	Scheduled film sessions linked to rooms	id, movie_id (FK), room_id (FK), screening_date, screening_time, available_seats
reservations	User bookings linked to screenings	id, user_id (FK), screening_id (FK), seats, created_at

3.2 Relationships

- A movie can have many screenings (one-to-many)
- A room can host many screenings (one-to-many)
- A screening belongs to one movie and one room
- A user can have many reservations (one-to-many)
- A reservation belongs to one user and one screening
- All foreign keys use ON DELETE CASCADE to ensure data consistency

3.3 PDO Singleton Pattern

The Database class implements the Singleton pattern, ensuring a single PDO connection is reused throughout the application. It is configured with exception-based error handling, associative fetch mode, and real prepared statements (emulated prepares disabled).

4. Implemented Features

4.1 Authentication & Account Management

- **Registration:** Validates username uniqueness, email format, password length (min 6 chars), and password confirmation. Passwords are hashed with bcrypt via password_hash().
- **Login:** Verifies credentials with password_verify(). On success, populates the session with user_id, username, role, and last_activity timestamp.
- **Remember Me:** Sets an HttpOnly cookie (remember_email) with a 30-day TTL. The cookie pre-fills the email field on subsequent visits.
- **Session Expiration:** The front controller checks last_activity on every request. Sessions inactive for more than 30 minutes are automatically destroyed.
- **Profile Management:** Users can update their username, email, and password. Account deletion cascades through the database, removing all associated reservations.
- **Logout:** Destroys the session and clears the remember-me cookie.

4.2 Movie Browsing

- **Film Listing:** Displays all films as responsive cards with title, truncated synopsis, and duration.
- **Film Detail:** Shows full description, duration, and all upcoming screenings with a visual seat availability bar (percentage-based).
- **Date Filtering:** Only future screenings are shown (WHERE screening_date >= CURDATE()).

4.3 Reservation System

- **Seat Selection:** Users select 1–10 seats per reservation. Client-side validation provides instant feedback; server-side validation prevents over-booking.
- **Availability Check:** Before confirming, the system verifies available_seats >= requested seats using an atomic UPDATE with a WHERE guard clause.
- **Cancellation:** Users can cancel their own reservations. The system restores seats atomically, with an upper bound check to prevent exceeding room capacity.
- **Access Control:** The requireLogin() guard redirects unauthenticated users to the login page.

4.4 Administration Panel

The admin panel is protected by requireAdmin() which checks the session role. It provides full CRUD operations for:

- **Films:** Create, edit (via modal dialog), and delete movies. Deleting a film cascades to its screenings and reservations.
- **Rooms:** Manage screening rooms with custom names and capacities. Each room has an independent seat count.
- **Screenings:** Schedule sessions by selecting a film, room, date, and time. The system validates room availability by checking for time-slot overlaps considering movie duration.
- **Reservations:** View all reservations across all users with the ability to cancel any booking.
- **Users:** View registered users with role badges. Admins can delete any user except themselves.

4.5 Room Availability Validation

When creating a screening, the system prevents scheduling conflicts through the isRoomAvailable() method. This method calculates time intervals based on movie duration and checks for overlapping sessions in the same room on the same date. Two screenings overlap if:

```
newStart < existingEnd  AND  newEnd > existingStart
```

5. Security Implementation

Security was a core design priority throughout the application. Every layer of the stack implements protections against common web vulnerabilities.

Threat	Protection	Implementation
SQL Injection	All queries use PDO prepared statements with parameter binding	$stmt->execute([$param]) throughout all models
XSS	All user-generated output is escaped before rendering	h() helper wraps htmlspecialchars(ENT_QUOTES, UTF-8)
Password Theft	Passwords are never stored in plaintext	password_hash(PASSWORD_BCRYPT) + password_verify()
Session Hijacking	Sessions expire after inactivity	30-minute timeout checked on every request
Privilege Escalation	Role-based access control	requireAdmin() guard on all admin routes
Cookie Theft	Remember-me cookie is HttpOnly	setcookie() with httponly flag = true

5.1 PDO Configuration

The database connection is hardened with three critical PDO options:

- **ERRMODE_EXCEPTION:** Ensures database errors are thrown as exceptions rather than silently ignored.
- **FETCH_ASSOC:** Returns clean associative arrays without duplicate numeric keys.
- **EMULATE_PREPARES = false:** Forces the database driver to use real prepared statements, providing true SQL injection protection at the protocol level.

6. Frontend Design

6.1 CSS Architecture

The stylesheet is modular, split into 16 dedicated files imported via main.css using @import. Each file has a single responsibility:

File	Responsibility
vars.css	CSS custom properties (design tokens): colors, spacing, border radius
reset.css	Box-sizing reset, base typography, scrollbar styling, body gradient background
nav.css	Sticky navigation bar with backdrop-filter blur, responsive collapse
buttons.css	Button variants: primary (glow), ghost, danger, sizes
movies.css	Responsive card grid (auto-fill), hover animations, seat progress bar
modal.css	Overlay with pop animation, close button, dark backdrop

6.2 JavaScript

- **main.js:** Auto-dismiss success messages (4s fade), confirmation dialogs via data-confirm attribute, modal open/close system, active nav-link highlighting.
- **reservation.js:** Real-time seat input validation with feedback text and submit button disable/enable.
- **admin.js:** Movie edit modal population from data attributes on the trigger button.

6.3 Dark Theme & Responsive Design

The UI uses a dark color scheme with purple accent (#7c3aed) and subtle gradient backgrounds. CSS custom properties enable consistent theming. The layout is responsive: card grids collapse from multi-column to single-column, the navigation links hide on mobile (< 700px), and tables scroll horizontally.

7. Development Workflow

7.1 Task Distribution

Area	Michel	Edwin
Backend	All models, controllers, routing, DB schema, security logic, session management	(assists, debugging)
Frontend	(assists, debugging)	All views (PHP templates), CSS architecture, JavaScript interactivity
Integration	Route wiring, availability validation, profile feature	Layout templates, admin views, bug fixes
Documentation	README, technical documentation	View-layer documentation

7.2 Methodology

The project followed a backend-first approach: models and controllers were built first to establish the data layer and business logic, followed by views that consume controller data. This ensured clean separation and avoided coupling between layers.

8. Installation & Setup

8.1 Prerequisites

- PHP 8.0 or higher
- MySQL 5.7+ or MariaDB
- Web server (Apache, Nginx) or the built-in PHP development server

8.2 Database Setup

Import the schema file to create the database, tables, and seed data:

```
mysql -u root -p < sql/schema.sql
```

This creates the cinema database with all tables, an admin account (admin@cinema.com / password), sample movies, rooms, and screenings.

8.3 Environment Configuration

Create a .env file at the project root:

```
DB_HOST=localhost
DB_NAME=cinema
DB_USER=root
DB_PASS=your_password
DB_CHARSET=utf8mb4
```

8.4 Running the Server

```
php -S localhost:8000
```

Then open `http://localhost:8000` in a browser.

9. Conclusion

Absolute Cinema successfully implements all required features of the assignment: authentication with session management, movie browsing, seat reservation with availability checking, and a role-based administration panel. The application follows a strict MVC architecture with no SQL in views and no HTML in models.

Beyond the core requirements, the project includes additional features such as room management with per-room capacity, screening conflict detection based on movie duration, remember-me functionality with secure cookies, and a fully modular CSS architecture with dark theme.

The security implementation covers all specified protections: prepared statements against SQL injection, htmlspecialchars against XSS, bcrypt password hashing, and session expiration after inactivity. The codebase is clean, well-structured, and ready for further extension.