Střední průmyslová škola elektrotechnická Ječná

Informační technologie

Praha 2, Ječná 30

Database project

Anna Marie Mičková

Programové vybavení

2026

Structure

1. Overview

Study Room Reservation System is application that enables users to book seats in study rooms for specific time slots. The system prevents double bookings, manages room inventory, tracks reservations, and provides administrative features. Built with ASP.NET Core and MySQL, it implements the Repository Pattern for clean separation between business logic and data access.

2. System architecture, design pattern

The Study room reservation system implements a three-tier architecture with the Repository pattern - solution D1.

2.1. Repository pattern

The repository pattern abstracts all database operations into dedicated repository classes. Each repository handles CRUD operations for specific entity. Business logic is separate from database logic, data access details are hidden from services, database changes only affect repositories, better for testing.

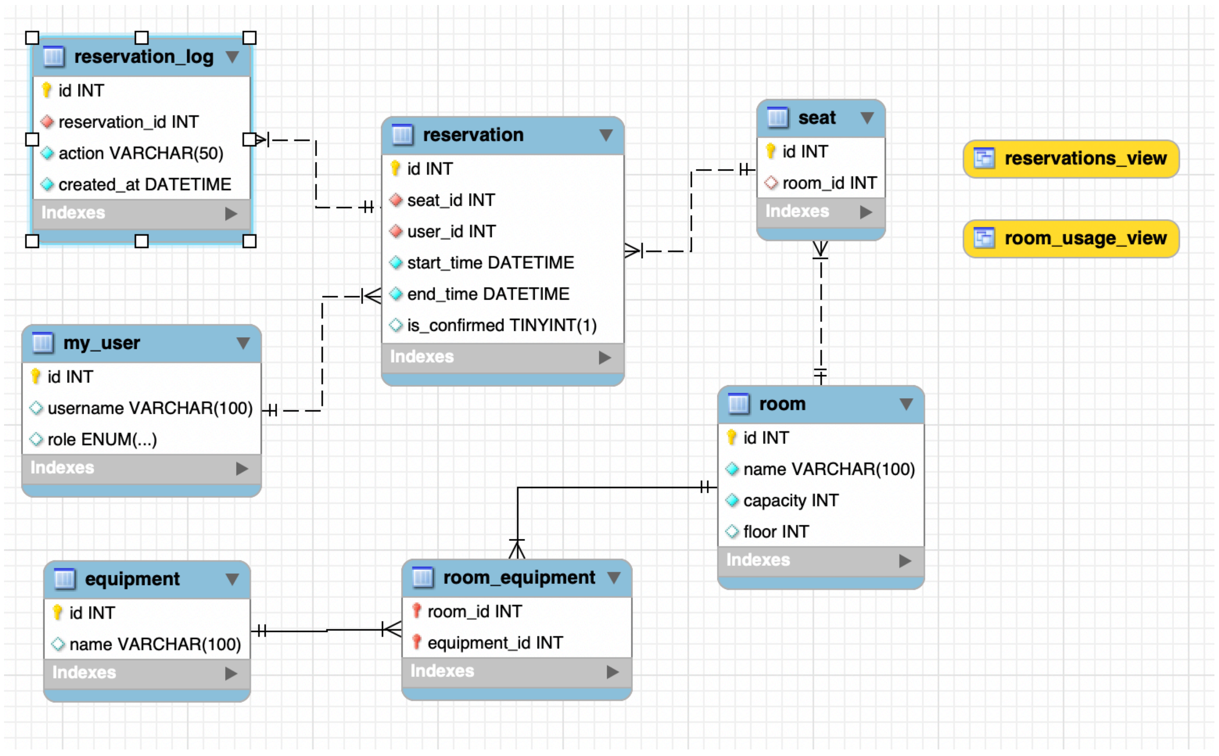
Repository classes with examples of methods:

- ReservationRepository - Manages reservation CRUD operations
    - AddReservation, GetAllReservations, UpdateReservation, UpdateReservation
- RoomRepository - Manages room and seat operations
    - AddRoom, GetRoomById, GetAllRooms
- UserRepository - Manages user accounts
    - AddUser, GetUserByUsername
- ImportRepository - Handles data import
    - ImportRoomsFromCsv, ImportEquipmentFromCsv

3. Database design

3.1. Tables specification

- room: represents room for studying
- seat: individual seat
- my_user: user account
- equipment: room study tools
- room_equipment: binding table
- reservation_log: tracks history of reservations



ER diagram

3.2. Database views

- reservations_view: joins reservation information
- room_usage_view - reservation statistics per room

4. Concurrency management

4.1. ReservationProcessor Architecture

The system uses configurable worker threads for concurrent request processing.

Incoming request -> ConcurrentQueue -> Worker -> Process reservation:

1. Validate room and seat
2. Create reservation via service
3. Complete TaskCompletionSource
4. Set result or exception

Configuration:

- Worker count defaults to Environment.ProcessorCount

- Can be configured in appsettings.json
- Each worker runs on separate thread
- Thread-safe ConcurrentQueue manages requests

Thread Safety Features:

- ConcurrentQueue provides lock-free operations
- Database constraints prevent double bookings
- TaskCompletionSource manages async/await pattern
- AutoResetEvent signals new items in queue

4.2. Double booking prevention

1. Time Overlap Detection (Repository layer)
    - Query: (start_time < @end_time) AND (end_time > @start_time)
    - Checked before insert
2. Database Transaction (ACID guarantee)
    - Check for conflicts
    - Insert reservation
    - Insert audit log
    - Atomically commit or rollback
3. Error Response (API layer)
    - Returns 400 error if conflict detected
    - User informed to choose different time/seat

5. Configuration & Error handling

5.1. Configuration

- Loads configuration at application startup
- Builds MySQL connection string
- Static ConnectionString property used by all repositories

appsettings.json:

- Server: localhost
- Port: 3306
- Database: study_rooms
- User: root
- Password: (database password)
- WorkerCount: (optional, defaults to processor count)

5.2. Logging framework

Logger class:

- Multiple log levels: DEBUG, INFO, WARNING, ERROR
- Console output with colored text
- File output to logs directory
- Thread-safe with lock mechanism
- Timestamps on all messages

5.3. Error handling

Client side:

- Username validation
- Time validation
- Empty field detection

Server side:

- Input format validation
- Business logic validation
- Room/seat existence checks

Database:

- Transaction rollback on error

- Time conflict detection
- Foreign key constraints
- NOT NULL constraints

5.4. Security

SQL Injection Prevention: Parameterized queries with @ parameters

6. API endpoints

Endpoint	Method	Purpose	Handler
/api/rooms	GET	Retrieve rooms with availability	RoomService
/api/reserve	POST	Create new reservation	ReservationService, ReservationProcessor
/api/reservations	GET	List all reservations	ReservationRepository
/api/reservations/{id}	PUT	Modify reservation times	ReservationRepository
/api/reservations/{id}	DELETE	Delete reservation	ReservationRepository
/api/reservations/{id}/confirm	POST	Confirm reservation	ReservationRepository
/api/report	GET	Generate statistics	ReportService
/api/import/rooms	POST	Import rooms from CSV	ImportRepository
/api/import/equipment	POST	Import equipment from CSV	ImportRepository

7. Technology

Component	Technology	Version
Runtime	.NET Core	9.0+
Language	C#	11.0+
Database	MySQL	5.7+
Web Framework	ASP.NET Core	Minimal APIs
Configuration	appsettings.json	JSON format

8. Requirements

Requirement	**Implementation**	**Status**
Real relational database system	MySQL 5.7+ with proper relationships	✓
5+ tables, 2 views, M:N binding	7 tables + 2 views + room_equipment	✓
All data types (int, string, datetime, bool, enum)	INT, VARCHAR, DATETIME, BOOLEAN, ENUM	✓
Multi-table CRUD with transactions	Reservation spans 4 tables with ACID	✓
Transaction support with rollback	BeginTransaction/Commit/Rollback	✓
Summary report from 3+ tables	6 reports from room, seat, reservation, equipment	✓
CSV data import (2+ tables)	Rooms and Equipment import	✓
Configuration file	appsettings.json with database settings	✓
Error handling for all scenarios	3-layer validation + logging	✓
Design Pattern	Repository Pattern (Solution D1)	✓