**Project API**

Media Equipment Rental System API Documentation

# Overview

The Media Equipment Rental System is a Java‐based application designed to manage the rental process for media equipment. It supports multiple user roles (Admin, Lecturer, Student, MediaStaff) and provides features for user authentication, equipment management, reservation handling, and equipment check-out/in. The system uses a Swing GUI for its user interface and follows a layered architecture with models, data access objects (DAOs), controllers, exceptions, and utility classes.

# Package Structure

* **model**: Contains data models (Equipment, Reservation, User) and DAO classes (EquipmentDAO, ReservationDAO, UserDAO, CheckoutDAO) for database interactions.
* **controller**: Contains business logic classes and utilities that process requests from the GUI and interact with the model layer.
* **exception**: Contains custom exceptions used for error handling.
* **view**: Contains Swing-based GUI classes for different user interfaces (AdminFrame, LecturerStudentFrame, MediaStaffFrame, LoginFrame, MainGUI).

# API Documentation

## Model Layer

### Equipment

* **Description**: Represents a piece of equipment.
* **Fields**:
  + equipmentId (String): Unique identifier.
  + name (String): Equipment name.
  + type (String): Equipment type.
  + description (String): Equipment description.
  + status (String): Equipment status (e.g., Available, Reserved, CheckedOut).
  + state (String): Condition of the equipment (e.g., New, Good, Fair, Poor).
* **Methods**: Standard getters and setters.
  + getEquipmentId(), setEquipmentId(String equipmentId)
  + getName(), setName(String name)
  + getType(), setType(String type)
  + getDescription(), setDescription(String description)
  + getStatus(), setStatus(String status)
  + getState(), setState(String state)

### Reservation

* **Description**: Represents a reservation request for equipment.
* **Fields**:
  + reservationId (int): Unique reservation ID.
  + userId (String): ID of the user who made the reservation.
  + equipmentId (String): ID of the reserved equipment.
  + reservationDate (Date): Date when the reservation was made.
  + returnDate (Date): Expected return date.
  + status (String): Reservation status (e.g., Pending, Approved, Rejected).
* **Methods**: Standard getters and setters.

### User

* **Description**: Represents a system user.
* **Fields**:
  + userId (String): Unique identifier (e.g., C001, L001).
  + email (String): Email address.
  + name (String): Full name.
  + password (String): Hashed password.
  + role (String): User role (Student, Lecturer, Admin, MediaStaff).
  + department (String): Department (if applicable).
  + course (String): Course (for students).
  + year (Integer): Year of study (for students).
* **Methods**: Standard getters and setters plus:
  + boolean isValid(): Validates that required fields (email, password, name, role) are present.

## DAO (Data Access Object) Layer

### EquipmentDAO

* **Description**: Manages database operations for equipment.
* **Key Methods**:
  + List<Equipment> getAllEquipment()
  + List<Equipment> getEquipmentByStatus(String status)
  + List<Equipment> getEquipmentByType(String type)
  + List<Equipment> getEquipmentByTypeAndStatus(String type, String status)
  + boolean addEquipment(Equipment equipment, String userId)
  + boolean updateEquipment(Equipment equipment, String requesterId)
  + boolean deleteEquipment(String equipmentId, String userId)

### ReservationDAO

* **Description**: Handles reservation-related database operations.
* **Key Methods**:
  + List<Reservation> getAllReservations(String userId, boolean isAdminOrMediaStaff)
  + boolean createReservation(String userId, String equipmentId, Date reservationDate)
  + boolean approveReservation(int reservationId, String staffId, String status)

### UserDAO

* **Description**: Manages user-related database operations.
* **Key Methods**:
  + User authenticateUser(String email, String password)
  + List<User> getAllUsers()
  + List<User> getLecturersAndStudents()
  + User getUserById(String userId)
  + String getUserRole(String userId)
  + boolean addUser(User user, String creatorId)
  + boolean updateUser(User user, String adminId)
  + boolean deleteUser(String userId, String requesterId)

### CheckoutDAO

* **Description**: Performs operations related to equipment check-out and check-in.
* **Key Methods**:
  + List<String> getPendingCheckouts()
  + boolean checkOutEquipment(int reservationId, String staffId)
  + boolean checkInEquipment(int reservationId, String staffId, String equipmentState)
  + List<String> getCheckedOutEquipment()

## Exception Classes

* **AuthenticationException**: Thrown when user authentication fails.
* **DatabaseOperationException**: Thrown when a database error occurs.
* **InvalidInputException**: Thrown when input validation fails.
* **ReservationConflictException**: Thrown when there is a conflict with an existing reservation.
* **RoleAccessException**: Thrown when a user tries to perform an action not permitted for their role.

## Controller Layer

### CheckoutController

* **Description**: Provides operations for checking equipment in and out.
* **Key Methods**:
  + List<String> getPendingCheckouts()
  + boolean checkOutEquipment(int reservationId, String staffId)
  + List<String> getCheckedOutEquipment()
  + boolean checkInEquipment(int reservationId, String staffId, String equipmentState)
* **Error Handling**: Logs errors to a file (checkout\_errors.log).

### EquipmentController

* **Description**: Manages equipment operations, including filtering by type/status and CRUD operations.
* **Key Methods**:
  + List<Equipment> getAllEquipment(String userRole)
  + List<Equipment> getEquipmentByType(String type, String userRole)
  + List<Equipment> getEquipmentByStatus(String status)
  + List<Equipment> getEquipmentByTypeAndStatus(String type, String status)
  + boolean addEquipment(Equipment newEquipment, String userId)
  + boolean updateEquipment(Equipment updatedEquipment, String userId)
  + boolean deleteEquipment(String equipmentId, String userId)
* **Error Handling**: Logs errors to a file (equipment\_errors.log).

### ReservationController

* **Description**: Handles reservation requests and status updates.
* **Key Methods**:
  + boolean requestReservation(String userId, String equipmentId, Date reservationDate)
  + List<Reservation> getAllReservations(String userId)
  + List<Reservation> getUserReservations(String userId)
  + boolean updateReservationStatus(int reservationId, String status, String staffId)
* **Error Handling**: Logs errors to a common log file (e.g., error.log).

### UserController

* **Description**: Manages user authentication and CRUD operations.
* **Key Methods**:
  + User login(String email, String password)
  + List<User> getAllUsers(String requesterRole)
  + List<User> getLecturersAndStudents()
  + User getUserById(String userId)
  + boolean addUser(User newUser, String adminId)
  + boolean updateUser(User updatedUser, String adminId)
  + boolean deleteUser(String userId, String adminId)
* **Error Handling**: Logs errors to a log file (error.log).

### LoginManager

* **Description**: Handles the login process.
* **Key Methods**:
  + User loginUser(): Prompts for credentials and authenticates.
  + User loginUser(String email, String password): Authenticates using provided credentials.
* **Error Handling**: Throws appropriate authentication or database exceptions.

### MethodsUtil

* **Description**: Provides helper methods to retrieve lists of departments and courses.
* **Key Methods**:
  + String[] getDepartments()
  + String[] getCoursesForDepartment(String department)

### InputValidator

* **Description**: Provides methods for validating user input.
* **Key Methods**:
  + String validateEmail(String email, String role)
  + String validateName(String name)
  + String validateEquipmentName(String equipmentName)
  + String validateEquipmentDescription(String description)

### PasswordUtils

* **Description**: Utility class for password hashing and verification.
* **Key Methods**:
  + String hashPassword(String plainPassword)
  + boolean verifyPassword(String plainPassword, String hashedPassword)

### RoleValidator

* **Description**: Validates if a user’s role is permitted for a given operation.
* **Key Methods**:
  + void validateRole(String userRole, String... allowedRoles) throws RoleAccessException

**Refreshable (Interface)**

* **Description**: Defines a contract for UI components that can refresh their displayed data.
* **Method**:
  + void refresh() throws DatabaseOperationException

## View (GUI) Layer

The view layer is built using Swing and includes the following classes:

### AdminFrame

* **Description**: Main interface for Admin users.
* **Features**:
  + Displays tabs for Home (dashboard with charts), Profile, User Management, Equipment Management, and Reservations Management.
  + Supports real-time refresh of data (implements Refreshable).

### LecturerStudentFrame

* **Description**: Interface for Lecturer and Student users.
* **Features**:
  + Provides a Home tab for viewing profile details.
  + Offers a Reservations tab for making and viewing equipment reservations.

### MediaStaffFrame

* **Description**: Interface for Media Staff users.
* **Features**:
  + Displays tabs for Home, Profile, User Management, Equipment Management, and Reservations Management.
  + Includes functionality for check-out and check-in operations.

### LoginFrame

* **Description**: Login window for users.
* **Features**:
  + Collects email and password.
  + Initiates authentication through LoginManager.

### MainGUI

* **Description**: Application entry point.
* **Features**:
  + Sets up the look and feel (using FlatDarkLaf).
  + Launches the LoginFrame on the Event Dispatch Thread.

# API Flow Summary

1. **Authentication**:
   * The user logs in via LoginFrame.
   * LoginManager and UserController verify credentials using UserDAO and PasswordUtils.
   * On success, the appropriate main GUI (AdminFrame, LecturerStudentFrame, or MediaStaffFrame) is launched.
2. **Equipment Management** (Admin/MediaStaff):
   * CRUD operations are performed through the GUI, with EquipmentController mediating calls to EquipmentDAO.
3. **Reservation Processing**:
   * Students and Lecturers request reservations via the GUI.
   * ReservationController processes these requests using ReservationDAO.
   * Admins/MediaStaff can update reservation statuses (approve/reject).
4. **Equipment Check-out/Check-in** (MediaStaff):
   * CheckoutController handles the check-out and check-in processes, interacting with the database through CheckoutDAO and calling stored procedures.
5. **Error Handling and Logging**:
   * Each controller captures exceptions (e.g., DatabaseOperationException, RoleAccessException) and logs detailed errors to respective log files.
   * Users receive generic error messages through the GUI.