Student details:

- STUDENT NAME: Ludovico Riccardo Nocco
- UTU STUDENT NUMBER: 268903
- UTU EMAIL ADDRESS: lrnocc@utu.fi
- PERSONAL EMAIL ADDRESS: l.r.nocco@gmail.com
- GITHUB REPOSITORY FOR THIS PROJECT: https://github.com/LudoNocco/movie-theatre-reservation-system

Movie Theater Booking System Documentation

1. Project Overview

1.1 Topic and Description

The Movie Theater Booking System is a Python-based application designed to manage the operations of a movie theater. It provides functionality for both administrators and customers, allowing for efficient management of movie screenings, hall allocations, and seat reservations.

Key features include:

- Adding and managing movies
- Scheduling screenings in specific halls
- Reserving seats for customers
- Viewing daily schedules and reservation details

1.2 Solution Principle

The solution is built on a SQLite database system, providing persistent storage for all theater data. The program uses a command-line interface to interact with users, offering different menus for administrators and customers. It employs modular programming principles, with distinct functions for each major operation, ensuring code reusability and maintainability.

2. Project Structure

2.1 Overall Architecture

The project follows a modular architecture, with the main script serving as the entry point and controller. It's structured into several key components:

1. Database Management: handles the creation and initialization of the SQLite database.
2. Data Operations: functions for adding, retrieving, and updating data in the database.
3. User Interfaces: separate interfaces for administrators and customers.
4. Utility Functions: helper functions for input validation and data formatting.

2.2 OOP Principles

While the current implementation primarily uses procedural programming, it incorporates some object-oriented principles:

- Encapsulation: data operations are encapsulated within specific functions, hiding the implementation details from the user interface.
- Modularity: the code is organized into logical units (functions) that perform specific tasks.

To fully leverage OOP, future versions could refactor the code into classes, such as `Theater`, `Movie`, `Screening`, and `Reservation`.

3. Function Details

3.1 Program Division into Functions

The program is divided into several key function categories:

1. Database Setup Functions
    - `create_tables()`
    - `initialize_halls()`
2. Data Management Functions
    - `add_movie()`
    - `add_screening()`
    - `reserve_seats()`
3. Data Retrieval Functions
    - `browse_reservations()`
    - `print_daily_schedule()`
4. User Interface Functions
    - `admin_interface()`

    - `customer_interface()`

5. Utility Functions
    - `validate_date()`
    - `validate_time()`

3.2 Function Descriptions

Database Setup Functions

`create_tables()`

- Goal: create necessary tables in the SQLite database if they don't exist.
- Operation: executes SQL commands to create tables for movies, halls, screenings, and reservations.
- Dependencies: none

`initialize_halls()`

- Goal: initialize the database with predefined hall information.
- Operation: inserts data for five halls with their names and capacities.
- Dependencies: requires the `halls` table to be created by `create_tables()`.

Data Management Functions

`add_movie(title, duration)`

- Goal: add a new movie to the database.
- Operation: inserts a new record into the `movies` table.
- Dependencies: none

`add_screening(movie_title, hall_name, date, time)`

- Goal: schedule a new movie screening.
- Operation: inserts a new record into the `screenings` table, linking it to the appropriate movie and hall.
- Dependencies: requires existing movie and hall records.

`reserve_seats(movie_title, screening_date, screening_time, customer_name, num_seats)`

- Goal: reserve seats for a customer for a specific screening.
- Operation: creates a new reservation record and updates the available seats for the screening.
- Dependencies: requires an existing screening record.

Data Retrieval Functions

`browse_reservations()`

- Goal: display all current reservations.
- Operation: retrieves and prints all reservation records, grouped by date and screening.
- Dependencies: none

`print_daily_schedule(date)`

- Goal: display the schedule for a specific date.
- Operation: retrieves and prints all screenings for the given date.
- Dependencies: none

User Interface Functions

`admin_interface()`

- Goal: provide a menu-driven interface for administrative operations.
- Operation: presents options for adding movies, scheduling screenings, browsing reservations, and viewing daily schedules.
- Dependencies: calls various data management and retrieval functions based on user input.

`customer_interface()`

- Goal: provide a menu-driven interface for customer operations.
- Operation: presents options for reserving seats.
- Dependencies: calls the `reserve_seats()` function based on user input.

Utility Functions

`validate_date(date_str)`

- Goal: ensure that date input is in the correct format (YYYY-MM-DD).
- Operation: attempts to parse the input string as a date.
- Dependencies: none

`validate_time(time_str)`

- Goal: ensure that time input is in the correct format (HH:MM).
- Operation: attempts to parse the input string as a time.

- Dependencies: none

3.3 Function Dependencies

- `add_screening()` depends on existing movie and hall records created by `add_movie()` and `initialize_halls()`.
- `reserve_seats()` depends on existing screening records created by `add_screening()`.
- User interface functions (`admin_interface()` and `customer_interface()`) depend on various data management and retrieval functions.
- All database operations depend on the tables created by `create_tables()`.

4. External Libraries

The program uses the following Python standard libraries:

1. `os`: for handling file paths.
2. `sqlite3`: for database operations.
3. `datetime`: for date and time operations and validations.

No external third-party libraries are required for this implementation.

5. Conclusion

The Movie Theater Booking System provides a comprehensive solution for managing a movie theater's operations. Its modular design allows for easy maintenance and potential future enhancements. The use of SQLite as the database system ensures data persistence without the need for a separate database server, making the application self-contained and easily deployable.