# Python Term Project – Movie Ticket Booking System

## 1. Project Overview

- **Goal:** Build a terminal-driven cinema booking platform handling showtimes, seat reservations, and ticket confirmations.
- **Scenario:** A boutique theater wants a minimal prototype to coordinate ticket sales and reporting before investing in a web solution.
- **Core Skills:** matrix-based seat management, file persistence, modular design, and user-friendly CLI menus.

## 2. Learning Outcomes

- Represent theatres, showtimes, and seat maps in Python data structures.
- Manage reservations, cancellations, and pricing tiers.
- Persist bookings to disk and maintain consistent seat availability.
- Provide reporting on attendance and revenue.
- Apply defensive programming to prevent double bookings and invalid selections.

## 3. Deliverables

1. `README.md` with setup, usage walkthrough, and sample booking flow.
2. Python modules:
    - `main.py` – application entry point and menu navigation.
    - `movies.py` – movie metadata and showtime scheduling.
    - `seating.py` – seat map creation, selection, and updates.
    - `bookings.py` – reservation logic, ticket issuance, cancellations.
    - `storage.py` – load/save, backups, and data validation.
    - `reports.py` – attendance and revenue analytics.
3. Data files (`data/movies.json`, `data/showtimes.json`, `data/bookings.json`).
4. Automated tests covering seat selection, double-book prevention, and cancellation.

## 4. Functional Requirements

### 4.1 Movie & Showtime Management

- Maintain catalog of movies with title, genre, duration, rating, and description.
- Schedule showtimes with theatre screen, date/time, language, and pricing tier.
- Admins can add, update, or retire movies and showtimes.

**Required Functions (`movies.py`):**

```
def load_movies(path: str) -> list: ...
def save_movies(path: str, movies: list) -> None: ...
def add_movie(movies: list, movie_data: dict) -> dict: ...
def schedule_showtime(showtimes: list, showtime_data: dict) -> dict: ...
def list_showtimes(showtimes: list, movie_id: str | None = None, date: str
```

```
| None = None) -> list: ...
def update_showtime(showtimes: list, showtime_id: str, updates: dict) ->
dict: ...
```

### 4.2 Seat Map Handling

- Generate seat grids per screen (e.g., rows A-H, seats 1-12) with zone pricing (standard, premium).
- Display seat maps with occupancy indicators.
- Reserve seats and mark them as sold, holding temporary locks during booking flow.

**Required Functions (`seating.py`):**

```
def initialize_seat_map(screen_config: dict) -> dict: ...
def render_seat_map(seat_map: dict) -> str: ...
def is_seat_available(seat_map: dict, seat_code: str) -> bool: ...
def reserve_seat(seat_map: dict, seat_code: str) -> dict: ...
def release_seat(seat_map: dict, seat_code: str) -> dict: ...
```

### 4.3 Booking Workflow

- Customers select showtime, choose seats, and provide contact details.
- Calculate ticket costs with pricing tiers, taxes, and optional discounts.
- Generate ticket confirmation with unique booking ID and QR-like code (text-based).
- Support cancellations and refunds according to rules.

**Required Functions (`bookings.py`):**

```
def create_booking(showtimes: list, seat_maps: dict, booking_data: dict) -
> dict: ...
def cancel_booking(bookings: list, booking_id: str, seat_maps: dict) ->
bool: ...
def calculate_booking_total(seats: list[str], pricing: dict, tax_rate:
float, discounts: list[dict]) -> dict: ...
def list_customer_bookings(bookings: list, email: str) -> list: ...
def generate_ticket(booking: dict, directory: str) -> str: ...
```

### 4.4 Data Persistence & Backups

- Persist showtimes, seat maps, and bookings after each operation.
- Maintain backups in `backups/` with timestamped files.
- Provide recovery workflow if data files become corrupted.

**Required Functions (`storage.py`):**

```
def load_state(base_dir: str) -> tuple[list, dict, list]: ...
def save_state(base_dir: str, showtimes: list, seat_maps: dict, bookings:
```

```
list) -> None: ...
def backup_state(base_dir: str, backup_dir: str) -> list[str]: ...
def validate_showtime(showtime: dict) -> bool: ...
```

### 4.5 Reporting & Analytics

- Summaries of tickets sold per showtime, occupancy rates, and revenue.
- Identify best-selling movies and peak days.
- Export daily or weekly performance reports.

**Required Functions (`reports.py`):**

```
def occupancy_report(showtimes: list, seat_maps: dict, bookings: list) ->
dict: ...
def revenue_summary(bookings: list, period: tuple[str, str]) -> dict: ...
def top_movies(bookings: list, showtimes: list, limit: int = 5) -> list:
...
def export_report(report: dict, filename: str) -> str: ...
```

## 5. User Experience Requirements

- Separate menus for customers and administrators.
- Seat selection displayed in a grid with legend (available, reserved, sold).
- Confirmation prompts before finalizing booking and cancellation actions.
- Show error messages for invalid seat codes or sold-out showtimes.

## 6. Validation & Business Rules

- Prevent double booking of seats; apply transactional logic when reserving multiple seats.
- Enforce cancellation policy (e.g., no cancellation within 30 minutes of showtime).
- Validate date/time formats (`YYYY-MM-DD HH:MM`).
- Ensure pricing tiers exist before computing totals.

## 7. Testing Expectations

- Unit tests for seat availability, booking total calculation, and cancellation path.
- Use temp files or mocks to avoid overwriting real data during tests.

## 8. Suggested Timeline

1. Week 1 – Movie and showtime data models.
2. Week 2 – Seat map generation and rendering.
3. Week 3 – Booking flow and persistence.
4. Week 4 – Reporting, backups, and UX refinement.
5. Week 5 – Testing and documentation.

## 9. Grading Rubric (100 pts)

- Booking Workflow & Seat Management: 30 pts
- Data Persistence & Recovery: 15 pts
- Pricing & Discount Logic: 15 pts
- Reporting & Analytics: 10 pts
- Input Validation & Reliability: 10 pts
- Documentation & UX: 10 pts
- Testing & Code Quality: 10 pts

## 10. Submission Checklist

- [ ] Sample showtimes and seat map provided.
- [ ] README demonstrates full booking and cancellation flow.
- [ ] Automated tests included and passing.
- [ ] Weekly commits show iterative development.