Title & Introduction 🎯

Section	Content	Notes
Title	**Backend Ticket Booking System**	Clean, professional title.
Subtitle	AlignTurtle Backend Developer Internship Assignment	Acknowledge the context.
Candidate	**POLEBOINA KAVYA**	Candidate Name.
Contact	Email: kavyapoleboina834@gmail.com, Phone: +91 82896315	Include your contact info and GitHub link (to be added later).
Goal	Design and implement a robust, secure, and fully documented movie ticket backend.	State the objective clearly.

Technology Stack & Architecture ⚙️

The system is built upon a standard but powerful technology stack optimized for rapid, reliable backend development. We chose **Python 3.x** and the **Django** framework for its "batteries-included" nature, which accelerates development across the entire application layer. The API interface is handled entirely by the **Django REST Framework (DRF)**, providing clean, standardized JSON responses and robust serialization. For security, we opted for **JWT (JSON Web Tokens)** via djangorestframework-simplejwt. This provides a **stateless authentication** mechanism, which is critical for scalability, as it removes the need for persistent server-side session storage. All components interact with the chosen database ([Specify Database, e.g., PostgreSQL/SQLite]), and the entire API contract is documented using **Swagger** for ease of consumption by frontend developers.

Data Model & Relationships 🗄️

The database schema is comprised of three primary models that establish a logical flow from catalog to transaction: **Movie, Show, and Booking**. The **Movie** model serves as the simple film catalog. The **Show** model is central to capacity management, linking to a **Movie** and defining the specific screen_name, date_time, and crucially, the **total_seats** capacity. The **Booking** model is the transactional record, establishing a many-to-one relationship with both the authenticated User and the specific Show. It also defines the **seat_number** and tracks the transaction state using the **status** field (booked or cancelled). This structure ensures data integrity and allows for efficient querying of seat availability based on active bookings for any given show.

JWT Authentication Flow 🔑

Authentication is mandatory for all booking-related operations to ensure user accountability. The flow begins with the **POST /signup** and **POST /login** endpoints. The login endpoint, handled by JWT, returns a short-lived **Access Token** and a long-lived **Refresh Token**. The Access Token is the key to the system; users must send it in the **Authorization: Bearer <token>** header for any subsequent protected API call. The stateless nature of JWT means the server doesn't need to consult the database on every request, as the token itself is digitally signed and contains all necessary authentication information, significantly improving performance and horizontal scaling.

Core Business Logic (The Challenge) ✅

The technical success of this assignment hinges on the robust implementation of the booking logic within the **POST /shows/<id>/book/** endpoint. To guarantee data integrity and prevent system failures, the entire booking process is encapsulated within a **database transaction** (@transaction.atomic). This ensures the check for availability and the creation of the new booking are treated as a single, indivisible operation.

1. **Preventing Overbooking:** Inside the transaction, a query is executed to count the total number of **active** bookings (status='booked') for the target Show. If this count is equal to or exceeds the Show.total_seats, the transaction is rolled back, and a specific "Capacity Full" error is returned.
2. **Preventing Double Booking:** Simultaneously, a check ensures no existing active Booking record is found for the exact combination of show_id and the requested seat_number. If a match is found, an "Seat Already Taken" error is returned, preventing conflicts.
3. **Seat Cancellation:** The **POST /bookings/<id>/cancel/** endpoint handles seat release. Instead of deleting the record, it performs a non-destructive update, changing the Booking.status to cancelled. This action immediately frees up the seat for future bookings while preserving the historical transactional data. Furthermore, a strict **security ownership check** ensures that a user can only cancel bookings that belong to their account.

Documentation & Deliverables 📦

The final submission adheres strictly to all required deliverables. The complete codebase, including the Django project and the requirements.txt file, resides in a **GitHub Repository** (URL to be provided upon submission). The project includes a comprehensive **README.md** that serves as the primary technical guide, detailing environment setup, database migration steps, and crucial instructions on obtaining and utilizing the JWT Access Token. Finally, complete API documentation is provided through **Swagger** integration, accessible at the mandated **/swagger/** route. This documentation showcases all endpoint schemas, expected responses, and clearly highlights the required JWT authorization header for all protected routes.

Bonus Points & Conclusion 🌟

To exceed expectations, several bonus requirements were addressed. **Unit Tests** were written specifically for the complex booking and cancellation logic to guarantee the business rules are enforced under various conditions. **Robust Error Handling** was integrated using Python try/except blocks and DRF exception handling, ensuring the API consistently returns clear, specific, and professional error messages instead of generic 500 errors. Furthermore, **Input Validation** was implemented to reject illogical data, such as negative seat numbers or attempts to book seats outside the defined total_seats range. These additions demonstrate a focus on production-ready code quality, security, and exceptional developer experience.

"Thank you for the detailed information and for shortlisting me for this opportunity."