**Presentation & Explanation**

**System Architecture**

* **Overview**: The system is a monolithic RESTful API built using Django, hosted on a local development server (e.g., http://127.0.0.1:8000/api/).
* **Components**:
  + **Frontend (Optional)**: Planned integration with React or Vue.js for user interaction.
  + **Backend**: Django handles API logic, with Django REST Framework (DRF) for endpoints.
  + **Database**: SQLite for development, with PostgreSQL recommended for production.
  + **Authentication**: JWT-based using djangorestframework-simplejwt.
  + **Environment**: Managed via .env files with python-dotenv.
* **Flow**: Client requests (e.g., GET /api/movies/) are routed through Django URLs to DRF views, processed with business logic, and return JSON responses.

**Key Features and Workflows**

* **User Features**:
  + Browse movies (GET /api/movies/).
  + View showtimes (GET /api/movies/<movie\_id>/showtimes/).
  + Book seats (POST /api/bookings/ with {"showtime\_id": 1, "seat\_number": "1A"}).
  + Cancel bookings (DELETE /api/bookings/<booking\_id>/).
* **Admin Features**:
  + Add movies (POST /api/movies/ with {"title": "Inception", "genre": "Sci-Fi", "duration": 148}).
  + Remove movies (DELETE /api/movies/<movie\_id>/).
  + Manage showtimes (POST /api/movies/<movie\_id>/showtimes/).
* **Workflow**:
  + User authenticates with /api/token/, receives a JWT.
  + Uses JWT to access protected endpoints.
  + Admins perform CRUD operations on movies and showtimes.

**Design Decisions and Implementation Insights**

* **Framework Choice**: Django with DRF for rapid development and RESTful compliance.
* **Authentication**: JWT chosen for stateless, scalable security over session-based auth.
* **Views**: Implemented both class-based (GenericAPIView with mixins) and function-based views for flexibility; urls.py supports switching via comments.
* **Database**: SQLite for simplicity in development; PostgreSQL planned for production scalability.
* **Debugging**: django-debug-toolbar and ipdb added for development insights.
* **Scalability**: Current monolithic design; future microservices or frontend separation considered.
* **Challenges**: Python 3.13 compatibility with Django 4.2.11; mitigated with 3.12 fallback.