VERY HARD JAVA ASSIGNMENT – ENTERPRISE PROJECT

Project: Hotel Booking & Management Platform (Backend-focused, Microservice-ready)

Objective

Build a production-grade backend application using Java and Spring Boot that supports hotel room booking, availability management, user authentication, payments (mock), reporting, and admin operations. The project must demonstrate layered architecture, RESTful APIs, persistence with MySQL, security with JWT, containerization with Docker, and CI basics. Focus on code quality, test coverage, documentation, and deployment readiness.

Core Requirements

Tech stack: Java 17+, Spring Boot 3.x, Spring Data JPA, Spring Security (JWT), MySQL (or PostgreSQL), Maven/Gradle.
Project modularity: separate packages (controller, service, repository, dto, model, config), clear layering and dependency injection.
Entities: User, Role, Hotel, Room, Booking, Payment, Review.
Authentication & Authorization: Register/Login endpoints, JWT-based auth, role-based access (ADMIN, USER).
Booking flow: Search rooms by hotel/date/availability; Create booking with availability check (atomic); Cancel booking (with refund rules).
Payment (Mocked): Simulate payments and store transaction records. Mark bookings as PAID on success.
Persistence: Use Spring Data JPA and provide DB migration scripts (Flyway or SQL scripts).
Validation & Error handling: Use DTOs, javax.validation annotations, and global exception handler.
API Documentation: Swagger/OpenAPI config.
Tests: Unit tests for services and integration tests for controllers (use H2 for tests).
Docker: Provide Dockerfile and docker-compose to run app + database locally.
Logging & Monitoring: Setup structured logs (SLF4J) and basic actuator endpoints.
README: Clear setup/run instructions, API endpoints list, sample requests.

Detailed Functional Specifications

Users & Roles: Users must register with email/password. ADMIN can manage hotels/rooms. USER can search and book rooms.
Hotel & Room: Hotel has id, name, address, rating. Room has id, hotelId, roomNumber, type (SINGLE/DOUBLE/SUITE), pricePerNight, amenities, totalQuantity, availableQuantity.
Booking: Booking has id, userId, roomId, checkIn, checkOut, numRooms, totalAmount, status (PENDING/CONFIRMED/CANCELLED/COMPLETED), createdAt.
Availability: When booking, atomically reduce availableQuantity for the room for the date range. Handle concurrency (optimistic locking or DB transactions).
Payment: POST /payments to simulate payment. Randomly fail 10% of requests. On success, mark booking as PAID and save Payment record.
Cancellation Policy: Allow full refund if cancelled >48 hours before check-in, 50% refund if within 48 hours. Simulate refund by creating a PAYMENT record with negative amount.
Reviews: Users can leave reviews for completed bookings (rating + comment).
Reports: ADMIN can export CSV or JSON report per hotel: bookings, revenue, occupancy rate for a date range.

Example API Endpoints (required)

POST /api/auth/register → register user
POST /api/auth/login → returns JWT
GET /api/hotels → list hotels (filter by location/rating)
POST /api/admin/hotels → create hotel (ADMIN only)
GET /api/hotels/{id}/rooms → list rooms for hotel with availability params

POST /api/bookings → create booking (requires JWT)
GET /api/bookings/{id} → booking details (owner or ADMIN)
POST /api/payments → simulate payment for booking
POST /api/bookings/{id}/cancel → cancel booking (with refund rules)
GET /api/admin/reports/hotel/{id}?from=&to;=&format;=csv/json → export report (ADMIN only)

Non-Functional Requirements

Code style: meaningful names, modular methods, small classes.
Security: store passwords hashed (BCrypt), validate tokens, secure endpoints.
Performance: handle concurrent bookings safely.
Tests: aim for 60%+ unit coverage and at least a few integration tests.
CI: Provide a GitHub Actions workflow to build, run tests, and produce a JAR.
Documentation: README with env variables, DB migrations, docker-compose usage, and sample curl/Postman calls.

Deliverables (submit as a ZIP or Git repo link)

1. Source code with clear package structure.
2. SQL migration scripts (or Flyway) and sample seed data.
3. Dockerfile and docker-compose.yml to run the application and DB.
4. Postman collection or sample curl commands for key flows.
5. Unit and integration tests.
6. README.md with build/run steps, environment variables, and API documentation.
7. Sample exported report and at least 3 sample receipts (text files).

Evaluation Criteria

Correctness of core booking and payment flows.
Code quality, architecture, and separation of concerns.
Error handling and validation.
Tests and reproducibility (docker-compose).
Documentation and ease of running the project.

Optional Bonus (will earn extra credit)

Integrate a lightweight email simulation to send booking confirmations (can be file-based).
Implement optimistic locking for room availability with versioning.
Add a basic front-end client (single-page using React or plain HTML/JS) to demonstrate APIs.
Deploy to Heroku/Cloud run and provide a public URL (extra).

Submission Instructions

Create a ZIP including the project (or provide Git repo URL).
Ensure docker-compose up works and the app bootstraps sample data.
Include instructions to run tests and generate reports.

Best Regards,
Nala Sravan Kumar
Full Stack Developer