internStudio

JAVA DEVELOPMENT INTERNSHIP

TASK GUIDE

Build robust, production-grade systems.
Java for real-world scale.

SIX ADVANCED PROJECTS — CHOOSE ANY TWO

01 **Enterprise Inventory & Order Management System**

02 **Library Management System with Role-Based Access**

03 **Banking & Ledger Simulation System**

04 **Hotel Reservation & Billing System**

05 **Multi-threaded Java Chat Server**

06 **Order-Processing Microservice with Message Queue**

INTERNSHIP	REQUIRED TASKS	FOCUS
Java Development	Complete any 2 projects	OOP · Spring Boot · Databases · Concurrency · APIs

INTERNSHIP INSTRUCTIONS

Your Internship Tasks

Select and complete **any 2 projects** from the task list. Each should be a properly layered, object-oriented Java application — not a single class with a main method.

Design	Model	Build	Test	Document	Deploy

EVERY PROJECT SHOULD INCLUDE

- [ ] A properly layered architecture (controller/service/repository)
- [ ] Sound object-oriented design
- [ ] A relational database with sensible relationships
- [ ] Input validation and meaningful exceptions
- [ ] Unit tests for core business logic
- [ ] Externalized configuration, not hard-coded values
- [ ] Consistent, documented API contracts
- [ ] Sensible logging

BEFORE YOU SUBMIT

- [ ] Complete any 2 projects from the list
- [ ] Push the project to GitHub
- [ ] Include a build file (Maven/Gradle)
- [ ] Add a README with setup and run instructions
- [ ] Add screenshots or sample requests/responses
- [ ] Submit details via the submission form
- [ ] Share your work on LinkedIn as required

> **Suggested technologies:** Java · Spring Boot · Spring Data JPA / Hibernate · PostgreSQL/MySQL · JUnit. Optional: Spring Security, Docker, RabbitMQ/Kafka.

Submit your completed work here

Once both projects are tested, documented and pushed to GitHub, submit your project details through the official submission form.

https://forms.gle/vxL5JE8cvRXXfDBv8

> **A note on quality.** Favor a clean, layered design over clever shortcuts. You should be able to explain your class design and why the layers are split the way they are.

01 Enterprise Inventory & Order Management System

Build a Spring Boot system that manages products, stock levels and customer orders for a mid-sized business.

REQUIREMENTS

- Layered architecture (controller/service/repository)
- Stock tracking with reservation on order
- Low-stock alerts
- REST API with proper DTOs
- Product and category management
- Order lifecycle (placed, fulfilled, cancelled)
- Role-based access (staff vs. manager)
- Unit and integration tests

Focus: Layered Spring Boot architecture, transactional stock management and role-based access.

02 Library Management System with Role-Based Access

Build a system for managing a library's catalog, member accounts and book lending, with different access per role.

REQUIREMENTS

- Spring Security-based authentication
- Catalog search and availability tracking
- Fine calculation for overdue returns
- Admin reporting on borrowing activity
- Role-based authorization (librarian vs. member)
- Borrowing and return workflow with due dates
- Reservation queue for unavailable books
- Unit tests for lending logic

Focus: Spring Security, role-based authorization and real lending/fine business logic.

03 Banking & Ledger Simulation System

Build a simplified banking system that manages accounts and processes transfers with correct double-entry bookkeeping.

REQUIREMENTS

- Account creation and balance management
- Concurrency-safe transfers (no lost updates)
- Overdraft and validation rules
- Statement generation for a date range
- Deposits, withdrawals and transfers between accounts
- Full transaction history per account
- Double-entry ledger consistency
- Unit tests covering concurrent transfer scenarios

Focus: Transactional correctness, concurrency control and financial domain modeling.

04 Hotel Reservation & Billing System

Build a system for managing room inventory, guest reservations and billing for a hotel.

REQUIREMENTS

- Room inventory and availability by date range
- Check-in / check-out workflow
- Invoice generation (PDF or formatted output)
- Cancellation and refund handling
- Reservation creation with conflict prevention
- Billing with itemized charges
- Admin dashboard for occupancy
- Unit tests for availability logic

Focus: Date-range availability logic, billing calculations and a real two-role system.

05 Multi-threaded Java Chat Server

Build a socket-based chat server that handles many concurrent clients using proper thread management.

REQUIREMENTS

- Custom socket server accepting multiple clients
- Chat rooms / channels
- Graceful handling of client disconnects
- Basic protocol for client-server messages
- Thread pool rather than one thread per client unmanaged
- Private messaging between users
- Message history persistence
- Load-tested with multiple simulated clients

Focus: Low-level networking, thread-pool concurrency and protocol design.

06 Order-Processing Microservice with Message Queue

Build a Spring Boot microservice that processes orders asynchronously via a message queue rather than a direct call chain.

REQUIREMENTS

- REST endpoint to submit an order
- A consumer service that processes orders asynchronously
- Dead-letter handling for failed messages
- Structured logging across the flow
- Order published to a message queue (RabbitMQ/Kafka)
- Idempotent processing (safe against duplicate delivery)
- Status tracking queryable by the client
- Integration tests covering the async path

Focus: Asynchronous, message-driven architecture and distributed-systems fundamentals.

FINAL STEP

Ready to Submit?

Verify everything below before you send your work in.

SUBMISSION

- [ ] GitHub repository
- [ ] README with setup and run steps
- [ ] Build file included (Maven/Gradle)
- [ ] Screenshots or sample requests included
- [ ] Submission form completed
- [ ] LinkedIn post where required

TESTING

- [ ] Unit tests for core business logic
- [ ] Tests pass on a clean checkout
- [ ] Edge cases considered

ARCHITECTURE

- [ ] Layers are properly separated
- [ ] Object-oriented design is sound
- [ ] No business logic leaking into controllers

DATA

- [ ] Database schema and relationships are sensible
- [ ] Migrations or schema scripts included
- [ ] Transactions used where correctness requires them

Important. Complete any 2 projects. You should be able to explain your class design, your data model, and any concurrency or transaction decisions you made.

Submission Form

Submit your repository, live link and screenshots here.

https://forms.gle/vxL5JE8cvRXXfDBv8

internStudio

LEARN · BUILD · GROW

www.internstudio.in

team.internstudio@gmail.com