📌 WEEK 4 — BACKEND SYSTEMS & PRODUCTION ENGINEERING

(Node.js + Express + MongoDB + API Architecture + Security + Scalability)

🎯 WEEK 4 ADVANCED OBJECTIVES

Interns will demonstrate **independent backend engineering capability** by:

- Designing **custom backend system architecture**
- Implementing **strict separation of concerns** with enforced contracts
- Applying **database optimization & query strategies**
- Building **defensive, production-safe APIs**
- Introducing **observability & async workflows**
- Producing **auditable documentation & artifacts**

This week evaluates **thinking + execution**, not memorization.

DAY 1 — BACKEND SYSTEM BOOTSTRAPPING & LIFECYCLE

🔹 **Learning Outcomes:**

- Node.js runtime lifecycle (not just event loop)
- Environment-driven bootstrapping
- Controlled startup & graceful shutdown
- Dependency orchestration

🔹 **Mandatory Folder Structure (No shortcuts)**

```
src/
 config/
 loaders/
 models/
```

routes/

controllers/

services/

repositories/

middlewares/

utils/

jobs/

logs/

🔹 **Topics to Learn**

- Event loop phases
- Node clustering
- Config loader + environment isolation
- Express advanced bootstrapping

🔹 **Exercise**

Build:

1. **App loader** (loads Express, middlewares, DB, routes in order)
2. **Config loader** supporting:

.env.local

.env.dev

.env.prod

3. **Startup logs** using Winston/Pino:

✔ Server started on port X

✔ Database connected

✔ Middlewares loaded

✔ Routes mounted: 23 endpoints

🔹 **Deliverables:**

- /src/loaders/app.js
- /src/loaders/db.js
- /src/utils/logger.js

- ARCHITECTURE.md

DAY 2 - DATA DESIGN & QUERY PERFORMANCE (NON-CRUD)

🔹 **Learning Outcomes:**

- Schema design for **read-heavy systems**
- Query cost analysis
- Data lifecycle rules

🔹 **Topics**

- Embedded vs Referenced schema
- TTL indexes
- Sparse + compound indexes
- Pagination strategies (skip/limit vs cursor)

🔹 **Exercise**

Build **Account** & **Order** schemas with:

- Pre-save hook (hash password or preprocess)
- Virtual fields (fullName or computed rating)
- Compound index: { status: 1, createdAt: -1 }
- Field validation & transformations

Implement the repository pattern:

`AccountRepository.create()`

`AccountRepository.findById()`

`AccountRepository.findPaginated()`

`AccountRepository.update()`

`AccountRepository.delete()`

🔹 **Deliverables:**

- /models/Account.js
- /models/Order.js
- /repositories/account.repository.js

- /repositories/order.repository.js
- Index analysis screenshot from MongoDB Compass

DAY 3 — QUERY PIPELINES & FAILURE-SAFE APIs

🔹 **Learning Outcomes:**

- API behavior under complex filters
- Dynamic filters, sorting, and Controlled soft-deletion
- Unified error contracts

🔹 **Topics**

- Controller → Service → Repository flow
- Complex filters:

`GET /products?search=phone&minPrice=100&maxPrice=500&sort=price:desc&tags=apple,samsung`

Soft deletes (flag + timestamp)
Advanced error handling:
  - Typed errors
  - Error codes
  - Centralized error middleware

🔹 **Exercise**

Build Product API with:

- Dynamic search engine (regex + OR/AND conditions)
- Filtering + sorting + pagination
- Soft delete with:

`DELETE /products/:id → marks deletedAt`

`GET /products?includeDeleted=true`

- Global error formats:

-
-

{ success: false, message, code, timestamp, path }

🔹 **Deliverables:**

- /controllers/product.controller.js
- /services/product.service.js
- /middlewares/error.middleware.js
- QUERY-ENGINE-DOC.md

DAY 4 — API DEFENSE & INPUT CONTROL

🔹 **Learning Outcomes:**

- Attack surface reduction & sanitize APIs
- Validation as first-class logic
- Rate limiting
- Input sanitization

Mandatory Protections

- Payload whitelisting
- Schema-level validation
- Request throttling
- Header hardening
- Query sanitization

🔹 **Topics:**

- Preventing:
  - NoSQL Injection
  - XSS
  - Parameter pollution
- JOI / Zod validation
- Helmet + CORS
- Rate limiting with express-rate-limit

🔹 **Exercise**

1. Build robust **validation schema** for User + Product.
2. Add global:
    - rate limiting
    - CORS policy
    - Helmet security headers
    - Payload size limits
3. Write **security test cases** (manual)

🔹 **Deliverables:**

- /middlewares/validate.js
- /middlewares/security.js (helmet, rate-limit, cors)
- SECURITY-REPORT.md (must show: vulnerabilities tested & results)

DAY 5 — ASYNC WORKERS, OBSERVABILITY & RELEASE READINESS

🔹 **Learning Outcomes:**

- Async background jobs
- Structured logging
- Postman documentation
- Production-ready backend thinking

🔹 **Topics:**

- Job queue design using:
    - Bull / BullMQ (Redis) (or in-memory fallback)
- Logging patterns (correlation IDs)
- Request tracing
- API documentation (Postman/Swagger)

🔹 **Exercise**

Implement:

Background job

- Job: email notification or report generation
- Queue: BullMQ
- Retry + backoff
- Worker process & logs

Request tracing

- Every request gets X-Request-ID
- Logs grouped by request ID

API Documentation

- Auto-generate using Swagger OR produce a Postman Collection
- Include folder-level environment variables

Deploy-ready folder prod/

- ecosystem.config.js (PM2)
- .env.example

🔹 **Deliverables:**

- /jobs/email.job.js
- /utils/tracing.js
- /logs/*.log
- Postman Collection Export
- DEPLOYMENT-NOTES.md

📌 WEEK-4 COMPLETION REQUIREMENTS

Skill Area	Requirement
Architecture	Layered structure + loaders
DB Modeling	Indexes + hooks + relations
API Engine	Pagination + sorting + filters
Security	All middlewares + vulnerability checks
Job Queues	Working background job

Documentation	Postman collection + diagrams

🎯 EXPECTED OUTCOME AFTER WEEK-4

Interns can now:

✔ Design backend systems independently

✔ Think beyond CRUD

✔ Anticipate production failures

✔ Secure APIs defensively

✔ Build observable services