📌 WEEK 4 — ADVANCED BACKEND ENGINEERING

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

🎯 WEEK 4 ADVANCED OBJECTIVES

Interns will learn:

- Professional backend architecture
- Clean, layered API design (Controller → Service → Repository → Model)
- Database modeling with indexing + performance tuning
- Security + sanitization + rate limiting
- Error boundaries + global exception handling
- Logging, monitoring & request tracing
- Async job queues
- Real-world API documentation & Postman Collections

This week produces engineers who can work in real startups.

DAY 1 — NODE + PROJECT ARCHITECTURE

- ◆ **Learning Outcomes:**

  - Node internals
  - Layered architecture
  - Config management
  - Professional folder structure

- ◆ **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 - DATABASE MODELING + INDEXING + ADVANCED CRUD

◆ **Learning Outcomes:**

- Designing real schemas
- Mongoose hooks, indexes, virtual fields
- Repository pattern

◆ **Topics**

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

◆ **Exercise**

Build **User** & **Product** 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 repository pattern:

```
UserRepository.create()
UserRepository.findById()
UserRepository.findPaginated()
UserRepository.update()
UserRepository.delete()
```

- ◆ **Deliverables:**

  - /models/User.js
  - /models/Product.js
  - /repositories/user.repository.js
  - /repositories/product.repository.js
  - Index analysis screenshot from MongoDB Compass

DAY 3 — HIGH-PERFORMANCE REST API + ADV QUERY ENGINE

- ◆ **Learning Outcomes:**

  - Build complex, production APIs
  - Dynamic filters, sorting, soft delete
  - Error boundaries

- ◆ **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 — SECURITY, VALIDATION, RATE LIMITING, HARDENING

- ◆ **Learning Outcomes:**

  - Secure & sanitize APIs
  - Request validation
  - Rate limiting
  - Input 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 — JOB QUEUES + LOGGING + API DOCUMENTATION + CAPSTONE

- ◆ **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:

1. **Background job**
   - Job: email notification or report generation
   - Queue: BullMQ
   - Retry + backoff
   - Worker process & logs
2. **Request tracing**
   - Every request gets `X-Request-ID`
   - Logs grouped by request ID
3. **API Documentation**
   - Auto-generate using Swagger OR produce a Postman Collection
   - Include folder-level environment variables
4. **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:

✔ Architect scalable backend projects
✔ Build complex REST APIs
✔ Secure, validate, and harden systems
✔ Handle background processing
✔ Produce documentation
✔ Build production-ready backend foundations