Month 6, Week 4: The Unabridged Learning Guide

Introduction: The Architect's Final Exam

For six months, you have been on a journey. You began as an apprentice, learning the fundamental tools of our craft: the command line, the Git repository, the grammar of JavaScript. You progressed to become a builder, architecting and constructing complete, secure, and data-driven applications with professional frameworks like NestJS. You learned the science of engineering, mastering the algorithms and data structures that separate performant systems from failed ones. You learned to think in systems, to deploy with confidence, and to automate your workflow.

This week, your final week, you are no longer a student. You are an architect.

This is your final exam, but it is not a test of memorization. It is a test of application. The **Capstone Project** is your opportunity to synthesize every skill you have acquired into a single, comprehensive, and professional-grade backend system. Working in a team, you will face the same challenges a senior engineer faces every day: clarifying requirements, designing a robust architecture, making technical trade-offs, and delivering a high-quality product.

This guide will serve as your blueprint for this final challenge. It will outline the project expectations, suggest ambitious ideas to test your limits, and provide a framework for the professional polish that will turn your project into the centerpiece of your portfolio. Your final presentation will not just be a demo; it will be a professional portfolio review, preparing you for the technical interviews and career that lie ahead.

You have the knowledge. You have the skills. It is time to build.

Table of Contents

Module 1: The Capstone Challenge

The Objective: From Zero to Production-Ready Your final task is to work in a team of three to design, build, test, document, and containerize a complete backend

system from scratch. This project will serve as the primary entry in your professional portfolio and will be the subject of your final review.

The "Definition of Done": A Checklist for Excellence A project is not "done" when it "works." It is done when it is professional. Your final submission will be graded against these criteria:

- [ ] **Correct & Functional:** The API works as designed and meets all the core functional requirements you have defined.
- [ ] **Well-Architected:** The project follows NestJS principles (a modular structure, clear separation of concerns between controllers and services, use of DI). The code is clean, readable, and consistent.
- [ ] **Secure:** Implements robust authentication (JWT) and authorization (Guards, Role-Based Access Control). All data is validated using DTOs and the `ValidationPipe`.
- [ ] **Data-Driven:** The application is connected to a persistent PostgreSQL database using TypeORM, with well-defined entities and relationships.
- [ ] **Tested:** The project has a comprehensive test suite.
  - **Unit Tests:** Services should have high test coverage, with dependencies mocked.
  - **Integration Tests:** Critical data pathways are tested against a real test database.
  - **E2E Tests:** The most important API endpoints are tested from the outside via Supertest.
- [ ] **Documented:** The project is easy for another developer to understand and run.
  - A professional `README.md` is mandatory.
  - API documentation (using a tool like Swagger, or simply clear documentation in the README) is mandatory.
- [ ] **Containerized:** The entire application stack (API, database, cache) can be started with a single `docker-compose up` command. A production-ready, multi-stage `Dockerfile` is mandatory.
- [ ] **Automated:** A `ci.yml` file for GitHub Actions is present, which automatically runs the linter and the full test suite on every push and pull request to the `main` branch.

Module 2: Project Ideation - Choosing Your Challenge

Project 1: The E-commerce Platform API

- **Core Challenge:** A classic but comprehensive project that requires a solid understanding of relational data.
- **Potential Data Model:**
  - `Users` (id, name, email, password, role)
  - `Products` (id, name, description, price, stock_quantity)
  - `Orders` (id, userId, status, total, created_at)
  - `OrderItems` (id, orderId, productId, quantity, price) - a "join table" for the many-to-many relationship.
- **Key API Endpoints:**

  - `POST /auth/register`, `POST /auth/login`
  - `GET /products`, `GET /products/:id`
  - `POST /products` (Admin only)
  - `GET /cart`, `POST /cart/items`, `DELETE /cart/items/:itemId`
  - `POST /orders` (Checkout)
  - `GET /orders`, `GET /orders/:id` (User can only see their own)

Project 2: The Real-Time Bidding API

- **Core Challenge:** This project focuses on managing state over time and handling potential race conditions.
- **Potential Data Model:**
  - `Users`
  - `Items` (id, name, description, starting_price, auction_end_time, owner_id)
  - `Bids` (id, itemId, userId, amount, created_at)
- **Key API Endpoints:**
  - `POST /items` (Create a new auction item)
  - `GET /items`, `GET /items/:id`
  - `POST /items/:id/bids` (Place a new bid)
- **Architectural Challenges:**
  - The `POST /items/:id/bids` endpoint is a critical section. You must ensure that a new bid is higher than the current highest bid. This might require a database transaction to prevent race conditions.
  - How do you handle the end of an auction? A background job scheduler (or a simple cron job) is needed to run every minute, find auctions that have ended, and notify the winner. This is a great place to use a message queue.

Project 3: The Food Delivery Service API

- **Core Challenge:** This project is about managing complex relationships and different user roles with distinct permissions.
- **Potential Data Model:**
  - `Users` (with a role: `customer`, `restaurant_owner`, `courier`)
  - `Restaurants` (id, name, address, owner_id)
  - `MenuItems` (id, name, price, description, restaurant_id)
  - `Orders` (id, customer_id, restaurant_id, courier_id, status, total)
- **Key API Endpoints:**
  - `POST /restaurants` (Owner only)
  - `POST /restaurants/:id/menu-items` (Owner of that restaurant only)
  - `GET /restaurants` (Public)
  - `POST /orders` (Customer only)
  - `PATCH /orders/:id/accept` (Courier only)
  - `PATCH /orders/:id/status` (Owner or Courier, depending on the status change)
- **Architectural Challenges:**
  - This project requires a robust Role-Based Access Control (RBAC) system. You will need to create multiple guards or a very flexible `RolesGuard` to handle the complex permissions.

- The order status is a state machine. You need to ensure that status transitions are valid (e.g., an order cannot go from `delivered` back to `preparing`).
- A message queue could be used to broadcast new, available orders to nearby couriers.

Module 3: The Architect's Blueprint & Professional Polish

The Design Document: Your Master Plan This is your first deliverable. Before coding, your team must agree on the architecture. This document forces you to think through the entire system and will save you countless hours of rework. It must be clear, concise, and agreed upon by all team members.

The Professional `README.md` This is the most important piece of documentation you will write. It is the first thing a future employer or collaborator will see. It must be excellent. Use Markdown to its full potential: headings, lists, code blocks, and tables.

Preparing for the Portfolio Review Your final presentation will be a 15-minute demo and architectural review. You will be expected to: 1. **Demonstrate the API:** Use a tool like Postman to demonstrate your key API endpoints working live. 2. **Explain Your Architecture:** Walk through your high-level design. Why did you choose the database schema you did? How do your services interact? 3. **Showcase Your Code Quality:** Briefly show a controller, a service, and a test file. Explain your logic. Be prepared to defend your decisions. 4. **Discuss Your CI/CD Pipeline:** Show your GitHub Actions workflow and explain what it does. 5. **Reflect on the Challenges:** What was the hardest part? What would you do differently next time? What trade-offs did you make? This is where you demonstrate a senior mindset.

Final Guidance & Next Steps

This capstone project is the most important work you will do in this program. It is your final exam and the beginning of your professional portfolio.

Treat your teammates not as classmates, but as fellow engineers. Use Git and GitHub professionally. Create pull requests for all your work and perform code reviews for each other. Communicate clearly and constantly.

You have all the tools. You have the knowledge. You have the architectural foundation.

Now, go build. ‘