Day-to-Day Tasks for Building a Scalable LMS Backend with NestJS (2025 Edition)

Introduction

This updated plan builds a production-ready LMS API for a junior NestJS developer. It includes:

- User roles (student, teacher, admin).
- JWT auth + role-based access.
- Courses, lessons, quizzes, progress tracking.
- **New:** Redis caching for performance.
- **New:** RabbitMQ for async tasks (e.g., emails on enrollment).
- **New:** Prisma ORM (modern, type-safe), Docker Compose, rate limiting, logging, basic GraphQL.
- **New:** Email notifications via Nodemailer + RabbitMQ.

Total Estimated Time: 130-150 hours (5 weeks).
Tech Stack: NestJS, TypeScript, PostgreSQL + Prisma, Redis, RabbitMQ, JWT, Swagger, Throttler, Winston, Nodemailer.
Assumptions: Basic Node/TS knowledge. Use Docker for services. Commit daily to Git.

Install global tools: `npm i -g @nestjs/cli prisma`.
Start with `nest new lms-backend --package-manager npm`.

Week 1: Setup, Docker, Auth & Users (Foundations)

Day	Tasks	Estimated Time (Hours)
1	- Create project & folder structure. - Set up ESLint/Prettier. - Install deps: Prisma, @nestjs/jwt, passport-jwt, class-validator, @nestjs/throttler, winston, etc. - Create docker-compose.yml (Postgres, Redis, RabbitMQ).	6
2	- Configure Prisma (schema.prisma with User model). - Set up ConfigModule + .env. - Init database & run first migration. - Test app with Docker.	6

Day	Tasks	Estimated Time (Hours)
3	- Create auth module: register/login with bcrypt. - JWT strategy & guards. - Role enum & role guard.	7
4	- Users module: CRUD with DTOs. - Protect routes (e.g., admins only delete). - Add rate limiting globally.	6
5	- Test auth/users with Postman. - Add Helmet & CORS. - Set up Winston logging.	6

Week 2: Courses & Enrollments (Core CRUD)

Day	Tasks	Estimated Time (Hours)
6	- Courses module: entity, CRUD, teacher-only create. - Enrollment logic (many-to-many relation in Prisma).	6
7	- Add course listing with pagination. - Implement enrollment endpoint + basic error handling.	7
8	- Lessons module: CRUD within course. - Quizzes module: CRUD + basic scoring.	7
9	- Progress module: track completion/scores.	6
10	- Manual testing + global validation pipe.	5

Week 3: Content & Progress + Swagger/GraphQL (Polish Basics)

Day	Tasks	Estimated Time (Hours)
11	- Add Swagger docs fully.	5
12	- Optional: Add GraphQL module (Apollo) for alternative queries.	6
13	- Implement quiz submission endpoint.	6
14	- Progress updates + GET progress.	6
15	- Testing endpoints + basic unit tests.	6

Week 4: Advanced Features & Testing

Day	Tasks	Estimated Time (Hours)
16	- Add email service (Nodemailer) stub. - Global exception filter.	5
17	- CQRS basics (commands for writes, queries for reads).	6
18	- Write unit/integration tests (Jest).	7
19	- Test coverage + bug fixes.	6
20	- Docker optimization + README docs. - Final review.	5

Week 5: Redis Caching + RabbitMQ Async (Scaling)

Day	Tasks	Estimated Time (Hours)
21	- Integrate Redis with @nestjs/cache-manager. - Cache course lists, user progress (e.g., @Cacheable on services).	6
22	- Implement cache invalidation (e.g., on updates).	5
23	- Set up RabbitMQ with @nestjs/microservices. - Create queues for emails (e.g., enrollment notification).	7
24	- Add producer/consumer: send email on enrollment/quiz completion. - Use @nestjs/bull as alternative if needed.	7
25	- Test async flows + full app testing. - Deploy prep (env for prod).	6

Tips for Success

- **Daily Routine:** Review, code, test, commit.
- **Resources:** NestJS docs, Prisma docs, Redis/RabbitMQ tutorials.
- **Extensions:** WebSockets for live progress, file uploads (S3), CI/CD.
- **Challenges:** Docker networking, cache invalidation—use NestJS Discord.
- **Why These Additions?** Redis reduces DB load; RabbitMQ enables async scalability; Prisma simplifies ORM; Docker standardizes dev.

Total Hours: ~140. Adjust as needed. This makes your LMS backend scalable and production-ready—great job!