Becoming a senior engineer.

A practical handbook for full-stack developers

INSIDE	QTY
CHAPTERS	12
CONCEPTS	134
CHECKLISTS	8
TEMPLATES	4
BUILD PLAN	1

feitura.

Becoming a senior engineer.

A practical handbook for full-stack developers

feitura.com

First edition · 2026

Becoming a Senior Engineer

A Practical Handbook for Full-Stack Developers

First edition, 2026 · Version 1.1

Published by Feitura · feitura.com

Typeset in Archivo and IBM Plex Mono.

Contents

00

INTRODUCTION

How to use this handbook

IN THIS INTRODUCTION PAGE

What this is

This book is a self-contained handbook for becoming a stronger senior full-stack developer. It is not a collection of links. It is designed to be readable without internet access and useful in any project you happen to be working on.

The content is original synthesis based on common senior engineering, leadership, architecture, testing, operations, and project-management practices. It is intentionally practical: every chapter should help you make better decisions, write better software, communicate better, and create evidence that you are operating at senior level.

The central idea

A senior developer is not merely someone who knows more syntax. A senior developer reliably improves outcomes in uncertain situations.

That means you need to develop four capabilities together:

1. Technical depth: You understand code, systems, data, testing, performance, and operations.
2. Product judgment: You can connect engineering choices to user and business outcomes.
3. Delivery skill: You can break work down, reduce risk, and finish valuable increments.
4. Leadership behavior: You improve the people and system around you without needing authority first.

This handbook trains those capabilities in parallel.

How to study

Use a weekly cycle:

1. Read one chapter or part of a chapter.
2. Extract three ideas you can apply immediately.
3. Apply one idea in code or project planning.

4. Write a short note explaining what changed in your thinking.
5. Review the week using the evidence log template in Part V.

Do not try to consume everything quickly. Seniority comes from integration, not exposure. The point is not to read many pages. The point is to change how you design, build, test, communicate, and decide.

Recommended weekly time budget

For 8 to 12 hours per week:

- 2 hours reading and note-making.
- 4 hours building or refactoring something.
- 2 hours testing, debugging, or performance work.
- 1 hour writing a design note, ADR, or review summary.
- 1 hour weekly review.

If you have less time, keep the same shape but reduce volume. Never remove the writing and review parts completely; they are where judgment becomes visible.

Evidence-based learning

Every month should produce evidence. Evidence is what separates vague confidence from senior-level proof.

Examples of evidence:

- A feature shipped with tests.
- A migration and schema explained clearly.
- A design document with trade-offs.
- A bug report with root cause and prevention.
- A refactor that reduced complexity.
- A performance investigation with before/after measurements.

- A code review that improved maintainability.
- A project plan that identified risks early.

Keep these artifacts. They become your senior portfolio.

How to read a chapter

For each chapter, answer:

1. What problem does this chapter help me solve?
2. What do I already do well here?
3. What do I avoid because it feels uncomfortable?
4. What is one concrete behavior I will practice this week?
5. What evidence will prove I practiced it?

Pair the handbook with a real project

Reading without practice fades. Pick one project where you can deliberately apply each chapter as you go. Any non-trivial app works: an order management tool, a project tracker, an inventory dashboard, a billing flow, a content site, an internal back-office. The Capstone in Part IV of this book, a complete build plan, provides one ready-made example you can build end to end.

Use the project to practice:

- Domain modeling with real entities and relationships.
- API design with clear resource boundaries.
- Frontend craft with real workflows.
- Testing strategy through unit, feature, and end-to-end tests.
- Operations through Docker, migrations, logs, and setup docs.
- Leadership through decision records and planning notes.

The project does not need to become huge. It needs to become a place where you practice senior habits deliberately.

The senior standard

For every topic, aim for this standard:

- I can explain the idea simply.
- I can apply it in code.
- I can identify trade-offs.
- I can teach it to someone else.
- I can produce evidence that I used it well.

When you can do those five things consistently across architecture, testing, delivery, communication, and leadership, you are no longer just collecting knowledge. You are building senior judgment.

CONCEPTS IN THIS INTRODUCTION

PART I

Thinking like a senior

01

CHAPTER 01

Senior engineer mindset

IN THIS CHAPTER PAGE

The shift from output to outcomes

Earlier in a career, success often means completing assigned tasks correctly. Senior work is different. The senior question is not only, “Did I build the thing?” It is also:

- Did this solve the right problem?
- Did it reduce or increase future risk?
- Can the team understand and maintain it?
- Did I make the trade-offs visible?
- Did I leave the system easier to change?

A senior developer still writes code. But the code is part of a larger responsibility: creating reliable outcomes through technical judgment.

Seniority is judgment under constraint

Real projects rarely have perfect information, perfect time, or perfect requirements. Seniority is the ability to make good decisions anyway.

Common constraints:

- The deadline is fixed but scope is unclear.
- The codebase has legacy behavior nobody fully understands.
- The ideal architecture is too expensive for the current stage.
- The business wants speed, but the system already has quality problems.
- The team disagrees about approach.

A senior developer does not wait for all uncertainty to disappear. They reduce uncertainty intentionally.

Useful senior questions:

- What is the smallest thing we can build to learn?
- What decision is reversible?
- What decision is expensive to reverse?

- What must be true for this approach to work?
- What could fail silently?
- What are we optimizing for right now?

The three horizons

Senior developers think across three horizons at once.

Horizon 1: Today

Can we ship the current work safely? Are tests passing? Are users blocked? Is there a production issue? Is the next step clear?

Horizon 2: This month

Are we accumulating debt that will slow us down? Are patterns emerging? Is the team aligned? Are we building the right foundation for upcoming features?

Horizon 3: This quarter

Will the architecture still support expected growth? Are we creating knowledge silos? Are we investing in tooling, documentation, and reliability before pain becomes crisis?

A common mistake is living only in Horizon 1. That creates short-term speed and long-term drag. Another mistake is living only in Horizon 3. That creates impressive diagrams and little delivery. Senior judgment balances all three.

Ownership without control

You will often be responsible for outcomes without having full control over every variable. That is normal.

Ownership means:

- You notice problems early.
- You communicate risks clearly.

- You propose options instead of only reporting obstacles.
- You follow through.
- You make hidden work visible.

Ownership does not mean doing everything yourself. In fact, doing everything yourself can be a failure mode. A senior developer improves the system of work so others can contribute.

Taste in engineering

Good senior engineers develop technical taste. Taste is not personal preference dressed up as authority. It is pattern recognition built from experience.

Examples of good taste:

- Choosing boring technology when reliability matters more than novelty.
- Avoiding an abstraction until duplication proves it is needed.
- Naming things based on domain meaning, not implementation detail.
- Writing tests around behavior rather than internal mechanics.
- Designing APIs that are hard to misuse.
- Keeping deployment and rollback in mind before production.

Taste improves when you review your own decisions. Ask: what did this choice make easier, what did it make harder, and what surprised me later?

The senior failure modes

Watch for these traps:

The hero trap

You solve everything personally, quickly, and silently. It feels productive, but the team becomes dependent on you. Senior work should increase team capability, not just individual throughput.

The architecture astronaut trap

You create complexity for future possibilities that may never arrive. Senior architecture is not maximum abstraction. It is appropriate structure for current and plausible future needs.

The cynic trap

You have seen many failures, so you become dismissive. Experience should create discernment, not contempt. The best senior engineers can be skeptical and constructive at the same time.

The local optimization trap

You optimize your ticket but harm the whole system. Senior developers care about the flow of value across the whole team and product.

A practical senior decision framework

When facing a decision, write short answers to:

1. Problem: What problem are we solving?
2. Context: What constraints matter?
3. Options: What are two or three viable approaches?
4. Trade-offs: What does each option cost?
5. Risk: What could go wrong?
6. Reversibility: Can we undo this later?
7. Decision: What are we choosing now?
8. Review: When will we revisit it?

This can become a small ADR. It does not need to be fancy. The value is in making reasoning explicit.

PRACTICE EXERCISE

Pick one technical decision from your current project. Write a one-page note:

- The decision.
- Why it was needed.
- Options considered.
- Why you chose the current option.
- What would make you change your mind.

This exercise trains the central senior muscle: explicit judgment.

CONCEPTS IN THIS CHAPTER