THE WINWINSP DEVELOPMENT METHOD SERIES

Volume I

GROWSHARELY
AI DEVELOPMENT HANDBOOK

A Real-World Journey of Building Software
with Architecture, Documentation
and Artificial Intelligence

By

Mohammad Belal Hossain (MBH)

with contributions from

ChatGPT & Cursor AI

Dhaka, Bangladesh • First Edition • July 2026

Building Software Through Principles, Documentation, and Human-AI Collaboration

THE WINWINSP DEVELOPMENT METHOD SERIES

Volume I

GROWSHARELY

AI DEVELOPMENT HANDBOOK

A Real-World Journey of Building Software

with Architecture, Documentation

and Artificial Intelligence

About This Book

This handbook is both:

- the story of building GrowSharely, and
- a practical guide to building software with artificial intelligence.

This book may be read as:

- A real-world case study
- A software architecture handbook
- A guide to human-AI collaboration
- A documentation methodology for modern software projects

First Edition | Published in Dhaka, Bangladesh | July 2026

Dedication

This book is dedicated to:

- Every software engineer who dreams of building something meaningful.
- Every small team that believes great products can be created with limited resources.
- Every learner who is willing to ask questions and keep improving.
- Every future builder who will create even better products by standing on the lessons recorded in these pages.

A Note from the Author

Dear Reader, This book was never part of the original plan. The original plan was simply to build a software product.

But somewhere during the journey, something unexpected happened. Every discussion, every bug, every design decision, and every lesson started becoming valuable on its own.

We realized that we were not only building software. We were also building:

- a methodology,
- a set of principles,
- a way of collaborating with AI,
- and a body of knowledge that could help future projects.

This handbook is our attempt to preserve that knowledge. I hope it inspires you to build, to learn, and to document your own journey.

— Mohammad Belal Hossain (MBH)

Table of Contents

Foreword

Software development is changing.

For decades, software was built entirely by human hands. Today, artificial intelligence has become a collaborator—helping developers design, document, test, and even think differently.

This handbook records one such journey.

GrowSharely was not merely a software project. It became an experiment in architecture, documentation, and human-AI collaboration.

The lessons captured in these pages are real. They were learned while building a real product with real business requirements and real users.

If this book inspires even one developer to think more deeply, document more carefully, and learn continuously, then it has fulfilled its purpose.

Mohammad Belal Hossain (MBH)
Dhaka, Bangladesh | July 2026

Preface

Why This Handbook Exists

Every software product begins with an idea.

Sometimes that idea becomes a prototype.

Sometimes it becomes a business.

And occasionally, it becomes a journey that teaches lessons far beyond the software itself.

GrowSharely was one of those journeys.

The original goal was simple: build a complete platform for managing real-estate investment projects and shareholders.

But as the work progressed, the project became something much larger.

It became:

- a software architecture laboratory,
- a documentation experiment,
- an AI-assisted development experience,
- and a learning journey.

This handbook exists because the lessons learned during that journey deserve to be preserved.

More Than a Software Project

Three participants worked together throughout the project.

Mohammad Belal Hossain (MBH): *Business owner, software engineer, and product visionary.*

ChatGPT: *Architect, mentor, reviewer, and discussion partner.*

Cursor AI: *Implementation partner and coding assistant.*

Together they demonstrated something important:

A small team, supported by artificial intelligence and disciplined development practices, can build remarkably sophisticated software.

Why This Book Was Written

Most technical books teach:

- programming languages,
- frameworks,
- design patterns,
- and tools.

Very few books explain:

- how real business decisions are made,
- how architecture evolves,
- how mistakes become principles,
- and how artificial intelligence can participate in professional software development.

This handbook attempts to fill that gap. Everything inside these pages comes from real discussions, real mistakes, and real solutions. Nothing here is theoretical.

Who Should Read This Book?

This handbook is written for:

Software Engineers: who want to become architects.

Architects: who enjoy learning from real projects.

Product Owners: who want to understand software decisions.

Students: who want to see how products evolve in the real world.

Small Teams: who want to use AI effectively.

How to Read This Book

The book is organized into four parts:

Part I – Principles: The beliefs and lessons that guided every important decision.

Part II – Workflow: The development process that evolved during the project.

Part III – The Architecture of GrowSharely: The actual product and the decisions behind it.

Part IV – Team Standards: The practices that kept the project maintainable and organized.

One Final Thought Before We Begin

Building software is not merely writing code. It is:

- understanding people,
- understanding businesses,
- making decisions,
- communicating clearly,
- and learning continuously.

Artificial intelligence can accelerate all of these activities.

But responsibility and judgment still belong to humans.

The human remains the architect.

AI becomes the partner.

PART I
PRINCIPLES

CHAPTER 1
MBH Principles

The Principles That Quietly Shaped GrowSharely

Some principles are written before a project begins.

Others are discovered while building.

The principles in this chapter belong to the second category.

They were not planned.

They emerged naturally through discussions, mistakes, reviews, and countless decisions made during the GrowSharely journey.

Looking back, these principles became the foundation of almost everything that followed.

PRINCIPLE 1: BUSINESS FIRST, CODE SECOND

Software exists to serve a business.

Whenever architecture and business appeared to disagree, the first question was always:

"How does the business actually work?"

Technology can be changed. Business reality cannot.

GrowSharely Example

Ownership and payment became two completely separate domains because the business understood them as two different things.

Lesson Learned

Never start with tables and database design.

Start with business questions.

PRINCIPLE 2: DOCUMENTATION IS PART OF THE PRODUCT

Documentation is not a luxury. It is not something you write when there is extra time.

Documentation is part of the product itself.

A system without documentation slowly becomes difficult to maintain, difficult to understand, and eventually difficult to trust.

GrowSharely Example

The `.winwinsp` folder became almost as important as the source code.

Lesson Learned

If something is important enough to discuss repeatedly, it is important enough to document.

PRINCIPLE 3: A FEATURE IS NOT COMPLETE UNTIL USERS CAN TEST IT

Developers often believe a feature is finished when the code compiles.

Business users think differently. For them, a feature is complete only when it solves a real problem and feels natural to use.

GrowSharely Example

Several modules were considered complete only after multiple rounds of user acceptance testing.

Lesson Learned

Budget time for UAT. It is not optional.

PRINCIPLE 4: NEVER RUSH TO PRODUCTION

A successful demonstration is not the same thing as a production-ready product.

Production involves real users, real money, and real consequences.

Patience protects quality.

Lesson Learned

> Staging environments and release candidates exist for a reason. Respect them.

PRINCIPLE 5: BUILD FOR MAINTAINABILITY

Software should not only work today.

It should remain understandable tomorrow.

A slightly slower development process is acceptable if it creates a system that is easier to maintain.

Lesson Learned

Future developers will thank you for clean architecture. Sometimes that future developer will be you.

PRINCIPLE 6: FINISH BEFORE OPTIMIZING

Premature optimization is one of the easiest ways to delay business value.

Build the right solution first. Optimize when the need becomes real.

Lesson Learned

> Finished software creates value.

Perfect but unfinished software does not.

PRINCIPLE 7: THINK LIKE THE USER

Every screen should answer a question.

Every report should solve a problem.

Every dashboard should provide clarity.

Lesson Learned

Good software begins by asking:

"What does the user want to know right now?"

PRINCIPLE 8: LEARNING NEVER STOPS

Every bug teaches something.

Every mistake reveals something.

Every discussion uncovers another idea.

The GrowSharely journey became an education in architecture, product thinking, and AI-assisted development.

Lesson Learned

The best engineers never stop learning.

PRINCIPLE 9: ARCHITECTURE SHOULD BE UNDERSTANDABLE

Complexity should only be introduced when absolutely necessary.

Simple systems are easier to explain, easier to maintain, and easier to trust.

Lesson Learned

If something cannot be explained clearly, it may still be too complicated.

PRINCIPLE 10: AI IS A PARTNER, NOT A REPLACEMENT

Artificial intelligence can:

- accelerate development,
- improve documentation,
- assist with architecture,
- and increase productivity.

But responsibility remains human.

Judgment remains human.

Ownership remains human.

LESSON LEARNED

The future is not:

Human versus AI

The future is:

Human and AI building together.

Chapter Reflection

When these principles first appeared, they looked like small observations.

Over time, they became something much more important.

They became the culture of the project.

And perhaps that is how all good principles are born—not in meeting rooms, but in the everyday work of building real products.

MBH Principle

> Build software that serves people, document what you learn, and never stop improving.

CHAPTER 2

Architect Principles

The Principles That Shaped GrowSharely

Every software project teaches lessons.

Some lessons are small. Some completely change the way you think about software.

The principles in this chapter were not invented in a classroom. They were discovered while solving real problems.

Sometimes a bug created a principle.

Sometimes a business discussion created one.

Sometimes a long evening of confusion suddenly produced a simple idea that changed the entire architecture.

This chapter contains the principles that became the backbone of GrowSharely.

Principle 1: Business Rules Before Database Design

Many developers begin by designing tables.

During GrowSharely we learned something different.

A database should represent the business. It should never define the business.

When business rules are unclear, database design usually becomes complicated.

When business rules become clear, the database often becomes surprisingly simple.

Lesson Learned

> Understand the business first. The tables will follow.

Principle 2: Architecture Is a Series of Small Decisions

People often imagine architecture as a huge design document.

In reality, architecture is thousands of small decisions.

Should this field be editable?

Who should approve this action?

Where should this business rule live?

Every small decision slowly shapes the entire product.

Lesson Learned

> Great architecture is built one decision at a time.

Principle 3: Ownership Is Not Payment

This may be the single most important principle discovered during GrowSharely.

A person may own shares and still have unpaid installments.

A person may pay money but not yet receive ownership.

These are different concepts.

By separating Ownership and Finance, the system became dramatically simpler.

Lesson Learned

> Different business concepts deserve different domains.

Principle 4: Dashboards Should Answer Questions

Users never open software because they want to admire screens.

They open software because they need answers.

A shareholder asks:

"How much do I own?"

A Project Admin asks:

"How is my project performing?"

An accountant asks:

"What needs my attention today?"

A good dashboard answers these questions immediately.

Lesson Learned

> Information should be useful before it becomes beautiful.

Principle 5: Store Facts, Compute Summaries

Facts should be stored.

Summaries should be calculated.

A collection is a fact.

A bank transaction is a fact.

A balance is a calculation.

A due amount is a calculation.

This principle protected GrowSharely from countless reporting problems.

Lesson Learned

> Store events. Compute answers.

Principle 6: Documentation Is Part of Architecture

Code explains what the system does.

Documentation explains why.

Both are necessary. Without documentation, architecture slowly disappears from memory.

Lesson Learned

> Documentation is not extra work. Documentation is part of the work.

Principle 7: Complexity Should Be Introduced Slowly

Every feature seems useful while discussing it.

But every feature adds maintenance cost.

The team repeatedly chose simplicity over unnecessary complexity.

Organizations were deferred.

Transfers were deferred.

Reservations were deferred.

Those decisions helped V1 reach completion.

Lesson Learned

> Simple products get finished.
>
> Complex products remain ideas.

Chapter Reflection

Looking back, most architectural improvements came from one simple question:

> "How does the business really work?"

That question solved more problems than any framework or design pattern.

Architect Principle

> Good architecture reflects business reality.

CHAPTER 3

Cursor Principles

Building an AI Development Partner

When Cursor first joined the project, it was simply a coding assistant.

Later it became:

- an implementer,
- a reviewer,
- a documentation assistant,
- and eventually a trusted development partner.

This transformation did not happen automatically.

It happened because the project learned how to work with AI.

Principle 1: Planning Before Coding

The quality of AI output depends heavily on preparation.

Whenever we started with implementation immediately, problems followed.

Whenever we started with planning, results improved.

The lesson became clear:

> Think first. Build second.

Principle 2: Documentation Is the Source of Truth

AI conversations become long.

Memory becomes imperfect.

Documentation became the permanent reference.

Whenever confusion appeared, documentation won.

Lesson Learned

> The written decision should always outrank memory.

Principle 3: Never Silently Revert Decisions

This principle was born from experience.

Approved decisions sometimes mysteriously returned to previous versions.

Menus came back.

Old code reappeared.

The team eventually established a simple rule:

Never undo an approved decision without discussion.

Lesson Learned

> Respect project memory.

Principle 4: Build in Small Milestones

Large changes create confusion.

Small milestones create momentum.

The project became easier to manage when development was divided into phases.

One milestone → One goal → One review → Then move forward.

Principle 5: Freeze Stable Foundations

Once a module became stable, it was frozen.

This reduced regressions and created confidence.

Not every part of a product should be continuously changing.

Lesson Learned

> Protect what already works.

Principle 6: AI Is a Junior Developer Without Guidance

AI can generate impressive solutions.

But it still requires:

- requirements,
- context,
- reviews,
- and decisions.

Without guidance, even good AI becomes inconsistent.

Lesson Learned

> Treat AI as a talented junior engineer.

Chapter Reflection

The project did not teach us how to use AI.

It taught us how to work with AI.

And there is a difference.

Cursor Principle

> AI performs best when humans provide clarity, context, and discipline.

CHAPTER 4

AI Governance Principles

Building Software Responsibly with Artificial Intelligence

Artificial intelligence can write code.

It can generate documentation.

It can suggest architecture.

It can even identify risks.

But one thing never changes.

Responsibility remains human.

This chapter exists because good governance turns AI into leverage.

Without governance, AI can become chaos.

Principle 1: AI Proposes, Humans Decide

AI should make suggestions.

Humans should make decisions.

Every major GrowSharely decision followed this rule.

This simple principle prevented many mistakes.

Principle 2: Human Approval Is Mandatory

Planning → Review → Approval → Implementation → Testing → Release.

This became the rhythm of the project.

Lesson Learned

Approval creates accountability.

Principle 3: Trust but Verify

AI accelerates development.

Verification protects quality.

Every important output deserves review.

Tests.

UAT.

Documentation.

Human judgment.

All remain essential.

Principle 4: Production Requires Higher Standards

A feature may work perfectly in development.

Production is different.

Real users.

Real money.

Real consequences.

The standards become higher.

Principle 5: AI Should Augment Learning

Perhaps the greatest surprise of this project was that AI became a mentor.

Architecture discussions became lessons.

Questions became discoveries.

The team itself became stronger.

Lesson Learned

The best use of AI is not replacing thinking.

It is improving thinking.

Chapter Reflection

The future of software development is not:

Human versus AI.

The future is:

Human and AI working together with clear responsibilities.

Governance Principle

Artificial intelligence accelerates development, but responsibility always remains human.

PART II
WORKFLOW

CHAPTER 5
Git Workflow

Managing a Product from Idea to Release

Git is often introduced as a version control system.

That description is technically correct.

But for GrowSharely, Git became much more than that.

Git became:

- memory,
- accountability,
- confidence,
- and protection.

Why Git Matters

Every project eventually needs to answer difficult questions.

Who changed this?

When did it change?

Why was it changed?

Without Git, those questions become difficult.

With Git, the answers are usually only a few commands away.

The Project Philosophy

The project adopted a simple rule:

> Commit often.
>
> Release carefully.

Small Commits Are Powerful

Large commits are difficult to understand.

Small commits tell stories.

A good commit usually represents one meaningful idea.

Examples:

- Profile verification completed.
- Portfolio dashboard redesigned.
- Financial approval workflow added.

Months later, these commits become a valuable history of the project.

Documentation Is a Real Change

One important lesson emerged during GrowSharely.

Documentation changes deserve commits too.

Because documentation is part of the product.

A decision that exists only inside someone's memory is fragile.

A documented decision becomes permanent knowledge.

Release Candidates

The project introduced:

> **v1.0.0-rc1**

This version had one purpose:

> Business testing.
>
> Not production.

The release candidate created a safe environment where users could validate the product without the pressure of a production launch.

Production Releases

Production releases require discipline. Before a release:

- tests pass,
- documentation is complete,
- migrations are verified,
- backups exist,
- users have signed off.

Only then should a product go live.

One of the Most Important Lessons

A rollback plan should exist before deployment. Not after.

Because confidence comes from knowing that recovery is possible.

The Human Rule

One important principle remained throughout the project:

AI can write code.

Humans make commits.

Humans create releases.

Humans approve production.

Responsibility stays where it belongs.

Chapter Reflection

Git is often viewed as a technical tool.

But perhaps its greatest value is emotional.

It allows developers to change things without fear. Because no matter what happens, history remembers.

Git Principle

> If a change is important enough to remember, it is important enough to commit.

CHAPTER 6

Cursor Workflow

Learning to Build Software with an AI Development Partner

When GrowSharely began, Cursor was introduced simply as a coding assistant.

The expectation was modest.

It would write code faster.

Perhaps it would save a little time.

But over the following weeks, something unexpected happened.

Cursor became much more than an editor with AI capabilities.

It became:

- an implementation partner,
- a documentation assistant,
- a reviewer,
- and occasionally, a student learning alongside the team.

This chapter explains how that relationship evolved and how the team learned to work effectively with an AI development partner.

The First Lesson

AI Works Best with Clear Instructions

In the early days, prompts were often vague.

The results were equally vague.

The team slowly discovered an important truth:

The quality of the answer depends heavily on the quality of the question.

When requirements became clearer:

- implementations improved,
- mistakes decreased,
- and development accelerated.

Lesson Learned

> Never assume the AI understands your intentions.
>
> Explain your requirements clearly.

The Planning Habit

One of the best practices that emerged during the project was simple:

Never start with code. Start with planning.

Almost every successful milestone followed the same pattern:

> Business Requirement
>
> ↓
>
> Architecture Discussion
>
> ↓
>
> Planning Document
>
> ↓
>
> Implementation
>
> ↓
>
> Testing
>
> ↓
>
> UAT

This habit dramatically reduced confusion.

Milestones Changed Everything

Large projects can easily become overwhelming.

The solution was to divide the project into small milestones.

Each milestone answered one question:

> "What business problem are we solving right now?"

The milestones became manageable.

The progress became measurable.

The project became easier to understand.

The Importance of Context

AI performs better when it understands the project.

This is why the `.winwinsp` folder became so valuable.

Instead of explaining the same things repeatedly, the team created:

- principles,
- decisions,
- architecture documents,
- milestone plans,
- and business rules.

Over time, the project developed its own memory.

Lesson Learned

> Documentation gives AI context. Context improves results.

Review Before Acceptance

The team never assumed that generated code was automatically correct.

Every implementation was reviewed.

Questions were always asked:

- Does it match the business rule?
- Does it respect previous decisions?
- Does it break existing features?
- Does it introduce unnecessary complexity?

Sometimes the answer was yes.

Those moments became learning opportunities.

The Funny Discoveries

The journey was not always serious.

There were many amusing moments.

The mysterious `.winwinsp` folder that seemed to disappear.

The famous "Editor Window" confusion.

The occasions when Cursor confidently generated something entirely different from what was requested.

Those moments became reminders that AI is powerful but not magical.

The Most Important Discovery

Eventually the team realized something surprising.

AI was not replacing software development.

It was changing the way software development happens.

Developers spend less time typing.

They spend more time:

- thinking,
- reviewing,
- deciding,
- and understanding the business.

Chapter Reflection

A good developer knows how to write code.

A great developer knows how to collaborate.

In modern software development, that collaboration increasingly includes artificial intelligence.

The key is not learning how to command AI. The key is learning how to work with it.

Cursor Principle

> AI becomes truly valuable when it understands your product, your decisions, and your way of thinking.

CHAPTER 7

AI Agent Mode

Letting AI Work Independently Without Losing Control

As confidence in AI grew, the team began experimenting with a different approach.

Instead of asking AI to solve one small task at a time, larger responsibilities were delegated.

This became known as **Agent Mode**.

An agent receives a goal.

Then it plans, works, and produces results with minimal supervision.

It is an incredibly powerful approach.

It is also one that requires discipline.

What Is Agent Mode?

Traditional prompting looks like this:

> Write this function.

Agent Mode looks like this:

> Implement this milestone and report back with the results.

The second approach gives AI more freedom. That freedom can dramatically improve productivity. It can also create unexpected problems.

The Benefits

Agent Mode helped:

- complete repetitive work,
- create documentation,
- build test cases,
- update multiple files,
- and implement large milestones.

The amount of work completed in a short period of time was often surprising.

The Risks

Greater freedom creates greater responsibility. Sometimes the AI:

- changed more than expected,
- reverted previous decisions,
- modified unrelated files,
- or misunderstood the business requirement.

These experiences taught the team an important lesson.

Trust Requires Verification

Agent Mode should never eliminate review.

The process became:

Plan

↓

Approve

↓

Execute

↓

Review

↓

Accept

The review step remained essential.

Small Tasks First

The team discovered another useful practice.

Do not begin with large agent tasks.

Begin with small ones.

Learn how the AI behaves.

Build trust gradually.

Then expand responsibilities.

Documentation Protects Agent Mode

Documentation became even more important when using agents.

Because the AI could reference:

- principles,
- ADRs,
- milestones,
- and business rules.

The chances of misunderstanding became much lower.

Human Responsibility

Agent Mode never removed accountability.

The final decision always remained human.

No code entered production without review.

No business rule changed without approval.

No milestone was accepted without testing.

Chapter Reflection

Agent Mode is powerful because it allows AI to work more independently.

It is safe because humans remain responsible for the outcome.

Freedom and responsibility must always exist together.

Agent Principle

> Delegate work to AI, but never delegate responsibility.

CHAPTER 8

Approval Process

Why Great Products Depend on Deliberate Decisions

Every successful project develops a rhythm. For GrowSharely, that rhythm became:

Discuss → Approve → Build → Review → Accept

This process may appear slow. In reality, it saved enormous amounts of time.

Why Approval Matters

Many software problems begin with assumptions. Someone assumes a requirement. Someone assumes a business rule. Someone assumes a decision.

Approval removes assumptions. It creates clarity.

The Approval Chain

The project eventually adopted a simple process:

Idea

↓

Discussion

↓

Decision

↓

Approval

↓

Implementation

Only approved ideas moved forward.

Business Decisions Need Business Approval

Some decisions belong to developers.

Some belong to business owners.

The team learned to distinguish between the two. Examples:

Developer Decisions

- Folder structure
- Naming conventions
- Test organization

Business Decisions

- Ownership rules
- Financial approval workflows
- Share allocation rules
- User permissions

The wrong people should never make business decisions.

Written Approval Is Better Than Memory

Verbal decisions are easily forgotten.

Written decisions become permanent.

This is why ADRs became so important.

Every important decision eventually found its way into documentation.

Approval Builds Confidence

A team works more confidently when decisions are clear.

There is less confusion.

Less rework.

Less unnecessary debate.

The project moves faster because people know the direction.

The Cost of Skipping Approval

Several difficult moments in the project shared one common characteristic:

The approval process had been skipped.

A requirement was assumed.

A business rule was unclear.

A decision was never documented.

The lesson became obvious.

Chapter Reflection

Approval may seem like an extra step.

In reality, it protects the project.

It creates alignment.

And alignment is one of the most valuable assets a team can possess.

Approval Principle

> Slow down long enough to make the right decision, and you will move faster afterwards.

CHAPTER 9

Review Process

The Discipline That Protects Software Quality

Every software team loves building new features.

Fewer people enjoy reviewing them.

Yet review is one of the most valuable activities in software development.

Review catches mistakes.

Review protects quality.

Review creates learning.

What Review Really Means

Review is not criticism.

Review is collaboration.

Its purpose is simple:

> Make the product better.

Code Review

Code review became an essential practice during GrowSharely.

The questions were straightforward:

- Is the implementation correct?
- Does it respect previous decisions?
- Is the code understandable?
- Does it create unnecessary complexity?

Sometimes small reviews prevented major problems.

Architecture Review

Some decisions affect the entire system. Those decisions deserve additional attention.

Architecture reviews often asked questions such as:

- Does this belong in the correct domain?
- Is this future-proof?
- Does it introduce coupling?
- Is there a simpler solution?

These discussions often improved the product dramatically.

Documentation Review

Documentation was treated as a first-class citizen. The team reviewed:

- milestone plans,
- principles,
- process documents,
- and ADRs.

Because incorrect documentation can be just as dangerous as incorrect code.

Business Review

Perhaps the most important review was performed by business users.

A feature is not successful because developers like it. A feature is successful because users can use it confidently.

This realization shaped the entire project.

Review Creates Learning

One unexpected benefit of review was education.

Developers learned from discussions.

AI improved through feedback.

The product became better because the team continuously reflected on its decisions.

The Human Habit

Eventually the team adopted a simple rule:

Nothing important is complete until it has been reviewed.

This rule protected:

- code,
- architecture,
- documentation,
- and business requirements.

Chapter Reflection

Review is not a delay.

Review is an investment.

It is one of the reasons why mature products remain stable while rushed products become difficult to maintain.

The time spent reviewing is rarely wasted.

Review Principle

> Building creates software. Reviewing creates quality.

End of Part II

At this point in the story, you now understand:

- the principles that guided the project,
- the workflows that made the project possible,
- and the relationship between humans and artificial intelligence.

The next part of this handbook becomes even more interesting. We now move from **how the team worked** to **what the team actually built**.

PART III

The Architecture of GrowSharely

Part I taught us the principles that guided the journey.

Part II showed us the workflows that helped transform ideas into working software.

Now we arrive at the heart of the story. This part of the book answers a different question:

What exactly did we build, and why did we build it this way?

GrowSharely did not emerge from a technical experiment.

It emerged from real business needs.

Real investors.

Real projects.

Real money.

Real accountability.

And because the business was real, the architecture had to become equally real.

The chapters that follow explain how a simple idea gradually became a complete platform.

CHAPTER 10

Product Vision

Why GrowSharely Was Built

Every product begins with a problem.

GrowSharely was no different.

The original problem was surprisingly simple.

People wanted to invest in projects together, but managing those investments was difficult.

Information was scattered.

Payments were hard to track.

Ownership records were difficult to maintain.

Communication between stakeholders often depended on spreadsheets, messaging applications, and memory.

As the complexity increased, so did the risks.

The idea behind GrowSharely was born from a simple question:

> **What if all of these activities could be managed from one place?**

That question eventually became a software product.

The Real Problem

Before writing a single line of code, it was important to understand the business.

The problem was not:

> **"How do we build a web application?"**

The real problem was:

> **"How do we help people confidently manage shared investment projects?"**

Those two questions lead to completely different products.

The first question creates software.

The second creates solutions.

The Vision

The vision of GrowSharely became clear:

> Build a platform that allows people to create projects, manage shareholders, record ownership, track finances, and communicate effectively—all from one system.

The product should be:

- simple enough for non-technical users,
- powerful enough for administrators,
- flexible enough for future growth,
- and structured enough to maintain trust.

The Four Pillars

As discussions progressed, four major pillars emerged.

Project Management

Every investment belongs to a project. Projects needed:

- information,
- settings,
- users,
- coordinators,
- and business rules.

Projects became the center of the platform.

Ownership Management

Who owns what?

This simple question created an entire domain. The system needed to manage:

- shares,
- shareholders,
- commitments,
- nominees,
- and ownership history.

Financial Management

Money needed to be handled carefully. The platform required:

- collections,
- bank accounts,
- expenses,
- approvals,
- and reporting.

The financial domain eventually became one of the largest parts of the system.

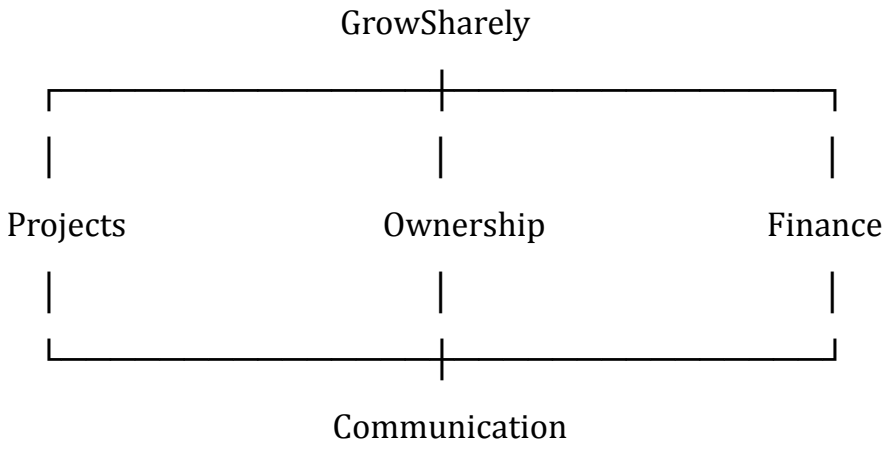
Communication

People need information. Projects need announcements. Users need support. The platform therefore introduced:

- notices,
- tickets,
- and notifications.

Because a platform is not complete if people cannot communicate effectively.

The Product Vision Diagram



Although the product contains many modules, everything eventually connects back to these four pillars.

The Human Side of the Vision

Software projects often become overly technical.

During GrowSharely, the team tried to remember something important:

People are not interested in modules.

They are interested in answers.

A shareholder asks:

> **"How much do I own?"**

An accountant asks:

> **"How much has been collected?"**

A Project Admin asks:

> **"How is my project performing?"**

The architecture had to answer these questions clearly.

Building for Real Users

One of the most important decisions made during the project was this:

> Build software that ordinary users can understand.

This decision influenced:

- dashboards,
- workflows,
- menus,
- reports,
- and even terminology.

Technical perfection means little if users feel lost.

The Long-Term Vision

The first version of GrowSharely focused on core business functionality. But the vision was always larger. Future possibilities included:

- organizations,
- mobile applications,
- APIs,

- communication channels,
- automation,
- and multi-tenant deployment.

The architecture was intentionally designed so that future growth would remain possible.

A Surprising Discovery

At one point, the project stopped feeling like a software application. It started feeling like a platform. That distinction matters.

Applications solve individual problems. Platforms create ecosystems. GrowSharely was slowly becoming a platform.

Chapter Reflection

Every successful product begins with understanding a real problem.

Technology comes later. **Frameworks** come later. **Databases** come later. **The vision comes first.**

And the clearer the vision becomes, the easier it becomes to make good decisions.

Product Principle

> Technology should never be the starting point of a product. The starting point is always a real human problem.

CHAPTER 11

Business Rules and Domain Principles

The Invisible Engine Behind GrowSharely

When people look at software, they usually see screens.

Buttons.

Menus.

Dashboards.

But underneath every successful product lies something much more important:

Business rules.

Business rules determine:

- what is allowed,
- what is prohibited,
- who can do what,
- and how information should behave.

Without business rules, software becomes unpredictable.

With good business rules, software becomes trustworthy.

What Is a Business Rule?

A business rule is simply a statement that defines how the business operates.

Examples:

- A shareholder can own multiple shares.
- Ownership does not automatically mean payment.
- A Project Accountant cannot approve his own transaction.
- A nominee does not receive additional permissions.

Simple statements. Powerful consequences.

Why Business Rules Matter

During development, many technical discussions eventually returned to the same question:

> **"What does the business want?"**

That question often resolved architectural disagreements.

The lesson became obvious:

Business rules should lead the architecture.

Not the other way around.

Domain Principle 1: Ownership Is Not Payment

This principle became one of the most important discoveries of the entire project.

A shareholder may:

- own shares,
- owe money,
- make partial payments,
- or complete payments.

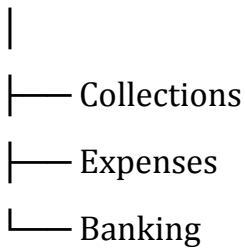
These concepts are related, but they are not the same.

This single realization led to two separate domains:

Ownership Domain

```
│
├── Shares
├── Shareholders
└── Nominees
```

Financial Domain



Separating these domains made the entire system easier to understand.

Domain Principle 2: Facts Are Stored. Answers Are Calculated.

Balances are not stored.

They are calculated.

Due amounts are not stored.

They are calculated.

Outstanding amounts are not stored.

They are calculated.

This principle protected the system from inconsistencies.

Domain Principle 3: Business Actions Must Be Auditable

People make mistakes. Software must be able to answer:

- who changed something,
- when it changed,
- and why it changed.

Audit trails became a first-class citizen in GrowSharely.

Domain Principle 4: Verification Creates Trust

Profiles require verification.

Nominees require verification.

Financial transactions require approval.

Trust is not assumed.

Trust is earned through workflow.

Domain Principle 5: Approvals Protect the Business

Some activities are too important to happen immediately. Examples:

- collections,
- expenses,
- withdrawals.

The approval process introduces a second pair of eyes.

Sometimes that second pair of eyes prevents very expensive mistakes.

Domain Principle 6: Information Should Exist Only Once

Duplicate information eventually creates confusion.

Therefore:

- balances are computed,
- ownership is stored once,
- summaries are derived,
- and reports reuse the same services.

This principle greatly reduced errors.

Domain Principle 7: Future Features Should Not Complicate Today

Many ideas were discussed:

- transfers,
- reservations,
- organizations,
- automation.

Some were postponed.

Not because they were bad ideas.

But because they were unnecessary for Version One.

This discipline helped the project reach completion.

Chapter Reflection

Business rules are rarely visible. Users never see them. Yet they quietly determine whether software becomes reliable or confusing. In many ways, business rules are the invisible engine of every successful product.

Domain Principle

> Software becomes trustworthy when it behaves exactly as the business expects.

CHAPTER 12

Roles, Permissions and Security Model

Building Trust Through Controlled Access

One of the earliest questions in the GrowSharely journey was surprisingly simple:

> "Who should be allowed to do what?"

At first glance, the answer appeared easy.

Create users.

Assign roles.

Done.

But as the project grew, the answer became more complicated.

A Project Admin should manage projects.

An Accountant should manage finances.

A Shareholder should see only his own information.

A Super Admin should oversee the entire platform.

And suddenly, a simple question became one of the most important architectural decisions in the entire system.

Why Permissions Matter

Permissions are not merely technical restrictions.

They protect:

- money,
- ownership,
- privacy,
- and trust.

Imagine if a shareholder could edit financial transactions.

Or if an accountant could transfer ownership.

Or if anyone could see another person's payment history.

The system would quickly become unreliable.

Good permissions create confidence.

The Four Core Roles

GrowSharely Version One introduced four major roles.

Each role serves a specific purpose.

Super Admin

The Super Admin is responsible for the platform itself.

This role can:

- create and manage projects,
- manage platform users,
- view platform summaries,
- configure system settings,
- monitor overall health.

However, one important principle was introduced:

> Being a Super Admin does not automatically grant operational access to every project.

This distinction became extremely important.

Project Admin

The Project Admin manages a specific project.

Responsibilities include:

- project settings,
- shareholders,
- profile verification,
- nominee verification,

- financial approvals,
- notices,
- tickets,
- and operational oversight.

The Project Admin is the business owner inside the project.

Project Accountant

The Accountant manages financial activities.

Responsibilities include:

- collections,
- deposits,
- withdrawals,
- expenses,
- reports.

However, an accountant cannot approve his own work.

This separation protects the business.

Shareholder

The Shareholder has the simplest role.

This user can:

- view ownership,
- view collections,
- view statements,
- receive notices,
- create tickets.

The shareholder never sees information belonging to others.

Privacy became a fundamental rule.

The Permission Model

The project eventually adopted two different concepts.

Platform Access

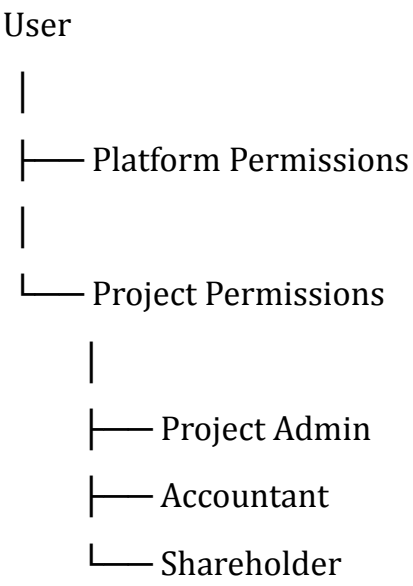
This determines whether a person may administer the platform.

Operational Access

This determines whether a person may work inside a specific project.

This distinction solved numerous problems.

The Security Diagram



Simple.

Easy to understand. Easy to maintain.

Authorization Is Business Logic

One of the most important discoveries of the project was this:

Authorization is not merely security.

Authorization is business logic.

Examples:

A Project Accountant cannot approve his own collection.

A nominee does not receive permissions.

A shareholder cannot see another shareholder's data.

These are business decisions. Not merely technical restrictions.

The Principle of Least Privilege

Every user should receive:

Exactly the permissions they need, and nothing more.

This principle guided almost every authorization decision.

Verification as a Security Layer

Permissions alone are not enough. Some actions require additional trust. For example:

- profile verification,
- nominee verification,
- financial approvals.

Verification became another layer of protection.

The Human Side of Security

Security is often discussed in technical terms.

Encryption.

Policies.

Permissions.

But security ultimately protects people.

It protects:

- trust,
- privacy,
- and confidence.

A secure system allows users to feel safe.

Chapter Reflection

Permissions are invisible when they work properly.

Users rarely think about them. But they quietly protect every important activity in the platform.

In many ways, security is one of the greatest acts of kindness a software product can offer its users.

Security Principle

> Give users everything they need, but nothing they do not.

CHAPTER 13

Modules and System Architecture

Understanding the Shape of GrowSharely

As the project evolved, it slowly became clear that GrowSharely was not a single application.

It was a collection of connected domains.

Each domain solved a different problem.

Together, they created one platform.

Understanding these modules became essential for understanding the architecture itself.

The Core Modules

The system eventually stabilized around eight major modules.

```
GrowSharely
│
├── Authentication
├── Projects
├── People
├── Ownership
├── Finance
├── Communications
├── Dashboards
└── Reporting
```

Each module had clear responsibilities. This separation made the system easier to understand and maintain.

Authentication Module

This module manages:

- login,
- registration,
- roles,
- permissions,
- access control.

Everything begins here.

Without authentication, the rest of the platform cannot function.

Project Module

Projects became the heart of the platform. Projects contain:

- shareholders,
- finances,
- communications,
- ownership records.

Almost every major activity eventually connects back to a project.

People Module

The People module manages:

- users,
- profiles,
- verification,
- project assignments.

This module became one of the most heavily used sections of the platform.

Ownership Module

The Ownership domain manages:

- shares,
- allocations,
- commitments,
- nominees.

This module answers one fundamental question: **Who owns what?**

Financial Module

The Financial domain manages:

- collections,
- bank accounts,
- expenses,
- installments,
- statements,
- balances.

This module answers another question: **What happened to the money?**

Communications Module

The Communications domain manages:

- notices,
- tickets,
- notifications.

Software is not complete if users cannot communicate.

Dashboard Module

Dashboards transform raw data into answers. They answer questions such as:

- How many shares do I own?
- What is my bank balance?
- What needs approval today?

Reporting Module

Reports convert information into documents. Examples:

- receipts,
- statements,
- ledgers,
- summaries.

Reports became extremely important during business testing.

Domain Boundaries

One of the greatest architectural achievements of GrowSharely was keeping domains separated.

For example:

> Ownership never updates financial tables.
>
> Finance never updates ownership tables.

Instead, each domain communicates through services and summaries.

The Architecture Diagram

```
Projects
 │
 ├── People
 ├── Ownership
 ├── Finance
 ├── Communication
 └── Reporting
```

This structure proved remarkably stable.

Why Modularity Matters

Modules provide several benefits:

- easier maintenance,
- simpler testing,
- safer development,
- clearer responsibilities.

When problems occur, they remain isolated.

When features are added, the impact remains controlled.

Chapter Reflection

Large software products become manageable only when they are divided into understandable pieces.

Modules provide those pieces.

They give shape to complexity.

And they make future growth possible.

Architecture Principle

> Complexity becomes manageable when responsibilities are clearly separated.

CHAPTER 14

Development Decisions and the ADR Journey

Recording Why Decisions Were Made

Most software projects document code.

Few document decisions.

This is unfortunate because decisions often outlive the code itself.

During GrowSharely, the team discovered the enormous value of documenting architectural decisions.

This practice eventually became one of the most important habits of the project.

What Is an ADR?

ADR stands for:

Architecture Decision Record

An ADR captures:

- the problem,
- the context,
- the decision,
- and the consequences.

Simple.

Powerful.

Permanent.

Why ADRs Matter

Without documentation, teams forget.

People leave.

Memories fade.

Requirements change.

An ADR becomes the memory of the project.

Whenever confusion appeared, the team often asked:

"Do we already have an ADR for this?"

Very often, the answer was yes.

The First Major Decision

One of the earliest ADRs separated:

Ownership and Finance.

This decision influenced almost every future module.

The Decision Process

The project gradually adopted a standard approach.

Problem

↓

Discussion

↓

Decision

↓

Documentation

↓

Implementation

This process reduced misunderstandings dramatically.

ADRs Prevent Repetition

Many questions appear repeatedly during long projects.

Should we support reservations?

Should Super Admin have project access?

Should balances be stored?

The answer no longer required another discussion. The ADR already existed.

ADRs Improve AI Collaboration

An unexpected benefit emerged.

Artificial intelligence became significantly more effective when it could read previous decisions.

Instead of guessing, the AI could understand:

- why a decision was made,
- what alternatives were rejected,
- and what principles should remain protected.

This dramatically improved consistency.

Important Decisions from GrowSharely

Some of the most important decisions included:

- Ownership and Finance separation.
- Computed balances.
- Manual share allocation.
- Approval workflows.
- Controlled permissions.
- Versioned nominee verification.
- Reporting strategy.
- Product completion principles.

Each decision quietly shaped the system.

The Cost of Missing Documentation

There were moments when decisions were remembered differently by different people.

These moments created confusion.

Eventually, a simple lesson emerged:

> **If a decision matters, write it down.**

A Living History

ADRs became much more than documents.

They became:

- history,
- lessons,
- and guidance for future projects.

In many ways, they became the memory of GrowSharely.

Chapter Reflection

Code explains what the system does.

Documentation explains why.

And among all forms of documentation, perhaps none are more valuable than decisions.

Because today's decisions become tomorrow's lessons.

Decision Principle

> The memory of a project should never depend on human memory alone.

End of Part III

At this point, you understand:

- why GrowSharely exists,
- how it works,
- how it is protected,
- and why its architecture evolved the way it did.

The final part of this handbook moves beyond software architecture.

It focuses on something equally important:

How a team works together, learns together, and builds lasting standards.

PART IV
TEAM STANDARDS

We now enter the final part of the book. This section should feel more personal because it is no longer only about software—it is about people, teamwork, learning, and the journey itself.

By the time GrowSharely reached this stage, the project had become much more than code.

It had become:

- a system of principles,
- a way of working,
- a method of learning,
- and a story of collaboration between humans and artificial intelligence.

The final chapters of this handbook attempt to preserve those lessons.

Because software eventually changes.

Frameworks change.

Technologies change.

But good principles and good habits remain valuable for many years.

CHAPTER 15

The `.winwinsp` System and Project Memory

Building ***a Second Brain for Software*** Development

One of the most important discoveries during the GrowSharely journey was simple:

> **Human memory is not enough.**

Projects become large.

Conversations become long.

Requirements evolve.

Decisions are forgotten.

Eventually, the team realized they needed something more reliable than memory.

That realization gave birth to:

The `.winwinsp` System.

What Is `.winwinsp` ?

At first glance, `.winwinsp` looks like a folder.

In reality, it became much more.

It became:

- project memory,
- architectural history,
- business documentation,
- learning notes,
- and an instruction manual for artificial intelligence.

It became the project's second brain.

Why Was It Needed?

Consider this simple question:

> **Why was a certain business rule implemented six months ago?**

Without documentation, the answer becomes:

"I think it was because..."

That answer is dangerous.

With documentation, the answer becomes:

"Here is the exact decision and the reason behind it."

The difference is enormous.

The Structure

The folder gradually evolved into something like this:

```
.winwinsp
│
├── architecture
├── business
├── docs
├── marketing
├── milestones
├── processes
├── decisions
├── guides
└── ai
```

Each section had a clear purpose.

Nothing was random.

Everything had a home.

Documentation as a Development Tool

Most teams write documentation after development.

GrowSharely slowly adopted a different approach.

Documentation became part of development itself.

The process often looked like this:

Discussion

↓

Documentation

↓

Implementation

↓

Testing

This habit reduced confusion dramatically.

Helping Future Developers

A good project should be understandable by someone who joins years later.

Documentation creates that possibility.

Without it, future developers spend their time rediscovering old decisions.

With it, they can move forward immediately.

Helping Artificial Intelligence

One of the surprising benefits of `.winwinsp` was its impact on AI.

The more context the AI received:

- the better its suggestions became,
- the more consistent its implementation became,
- and the fewer mistakes it made.

Documentation improved both humans and machines.

Lessons Learned

Several lessons emerged:

> **If a decision matters, document it.**
>
> **If a process repeats, document it.**
>
> **If a question keeps returning, document it.**

Eventually, documentation becomes an investment that continuously pays dividends.

Chapter Reflection

Many software projects have source code.

Very few have memory.

The `.winwinsp` system became one of the most valuable assets of GrowSharely because it preserved knowledge.

And preserved knowledge creates continuity.

Project Memory Principle

> Projects grow faster when they stop depending entirely on human memory.

CHAPTER 16

AI Collaboration and Pair Programming

Working Alongside Artificial Intelligence

Many people ask a simple question:

> "Can AI replace developers?"

The GrowSharely experience suggested a different question:

> "How can developers and AI work together effectively?"

This chapter is the answer to that question.

The Beginning

At first, AI was used for small tasks:

- generating code,
- fixing errors,
- writing documentation.

Over time, the relationship changed.

AI became:

- a reviewer,
- an architect,
- a mentor,
- and occasionally, a creative partner.

The New Form of Pair Programming

Traditional pair programming has two people.

One writes.

One reviews.

AI introduced a new variation.

Human → Thinks

AI → Generates

Human → Reviews

AI → Improves

Human → Approves

The cycle repeats.

Both participants contribute.

The Human Responsibilities

Throughout the project, certain responsibilities always remained human.

Humans:

- define goals,
- understand business,
- make decisions,
- accept risks,
- approve releases.

These responsibilities never changed.

The AI Responsibilities

AI proved exceptionally good at:

- implementation,
- explanations,
- documentation,
- summaries,
- repetitive work.

The combination became remarkably effective.

The Biggest Mistake

The team occasionally made one mistake.

Assuming AI was always correct.

This assumption inevitably caused problems.

The lesson became obvious.

Trust AI.

But verify its work.

AI as a Mentor

Perhaps the greatest surprise was educational.

Architecture discussions became lessons.

Problems became opportunities to learn.

The project itself became a classroom.

In many ways, AI became a teacher.

AI as a Student

Interestingly, the reverse was also true.

The AI learned from:

- documentation,
- principles,
- feedback,
- and corrections.

Over time, the collaboration improved.

The Future

The future of software development will likely involve:

Humans and AI working together.

The best developers will not necessarily be those who type the fastest.

They may be those who:

- ask the best questions,
- give the clearest instructions,
- and make the wisest decisions.

Chapter Reflection

Artificial intelligence does not remove the need for developers.

It changes the nature of their work.

And perhaps that is one of the most exciting developments in modern software engineering.

Collaboration Principle

> AI can generate solutions. Humans give those solutions meaning.

CHAPTER 17

Lessons Learned from Building GrowSharely

The Things We Did Not Expect to Learn

Every project teaches technical lessons. GrowSharely taught much more.

Some of the most important lessons had nothing to do with programming.

They were lessons about:

- people,
- communication,
- patience,
- and discipline.

Lesson 1: Simplicity Wins

Many ideas seemed exciting.

Many features looked useful.

Yet some of the best decisions were the features that were postponed.

The team repeatedly chose simplicity.

That decision helped the product reach completion.

Lesson 2: Business Understanding Is More Valuable Than Technical Knowledge

The greatest architectural breakthroughs often came from understanding the business.

Not from learning another framework.

Not from writing more code.

The business remained the teacher.

Lesson 3: Documentation Saves Time

Writing documentation may feel slow.

Rebuilding lost knowledge is much slower.

The `.winwinsp` folder repeatedly proved its value.

Lesson 4: Testing Builds Confidence

The project eventually passed hundreds of automated tests.

Those tests became a source of confidence.

Because change became less frightening.

Lesson 5: Completion Is Difficult

Starting is easy.

Finishing is difficult.

Many projects never reach completion.

One of the greatest achievements of GrowSharely was simply that it continued moving forward.

Lesson 6: Good Software Requires Patience

Architecture requires thought.

Testing requires discipline.

Documentation requires effort.

Rushing usually creates problems.

Patience often creates quality.

Lesson 7: Learning Never Stops

Perhaps the most beautiful lesson was this:

> **Every discussion became an opportunity to learn.**
>
> **Every bug became a teacher.**
>
> **Every problem became an invitation to improve.**

The Funny Moments: Some lessons were humorous.

> The disappearing `.winwinsp` folder.
>
> The mysterious editor window.
>
> The occasions when AI confidently misunderstood the requirement.

These moments became part of the story. And stories make projects memorable.

The Biggest Discovery: Software development is not merely a technical activity.

It is:

- communication,
- decision-making,
- teamwork,
- and continuous learning.

The code is only one part of the story.

Chapter Reflection

Looking back, the project produced something much larger than software.

It produced knowledge.

And knowledge may ultimately become the most valuable product of all.

Lesson Principle

> Every project builds software, but great projects also build wisdom.

CHAPTER 18

The Future Beyond GrowSharely

Every Ending Is the Beginning of Something New

When Version One was nearing completion, an interesting question emerged:

> What happens next?

The answer was simple.

GrowSharely was never the final destination.

It was the beginning.

The Future of the Product

Many possibilities remain:

- organizations,
- mobile applications,
- APIs,
- automation,
- communications,
- advanced reporting.

The architecture was intentionally designed to support future growth.

The Future of the Team

The lessons learned during GrowSharely can be applied to future projects.

The principles remain useful.

The workflows remain valuable.

The documentation approach remains powerful.

The Future of AI Development

Artificial intelligence will continue to improve.

The tools will change.

The capabilities will expand.

But one principle is unlikely to change:

Humans will remain responsible for understanding the problem.

The Future of This Handbook

This book should never become static.

It should evolve.

New lessons should be added.

New discoveries should be recorded.

Future editions may tell new stories.

A Dream Beyond GrowSharely

One of the most exciting possibilities is the creation of:

The Win-Win SaaS Starter Kit

A reusable foundation built from everything learned during this project.

In many ways, GrowSharely became the laboratory for something even larger.

To Future Builders

If you are reading this book years from now, remember this:

You do not need a large team.

You do not need unlimited resources.

You do not need perfect conditions.

You need:

- curiosity,
- discipline,
- patience,
- and the willingness to learn.

Final Reflection

The GrowSharely journey taught one simple truth:

Software is ultimately about people.

People create ideas.

People solve problems.

People make decisions.

Technology simply helps us achieve those goals.

One Final Conversation

Dear Reader, If this handbook has taught you anything, I hope it is this:

Build carefully.	Think deeply.
Document generously.	Learn continuously.

And never be afraid to ask questions. Because every great product begins with a question. And every great architect remains a student.

Final Principle

> Build software that serves people, preserve what you learn, and leave the next builder a better path than the one you found.

APPENDIX A

Troubleshooting Handbook

Lessons Learned the Hard Way

Software projects are full of surprises.

Some surprises are exciting.

Others consume an entire afternoon, several cups of tea, and occasionally a little bit of patience.

This appendix contains some of the memorable problems encountered during the GrowSharely journey and the lessons they taught us.

The purpose is simple:

> Save the next builder some time.

Case 1: Composer Cannot Find PHP

Problem

Composer suddenly stopped working after a server upgrade.

Error

PHP executable not found.

Cause

The Ubuntu upgrade removed or changed the active PHP version.

Solution

Install or reconfigure the PHP version.

```
sudo apt install php

sudo update-alternatives --config php

php -v
```

Lesson Learned

Never assume the environment remained unchanged after an upgrade.

Verify first.

Case 2: Apt Update Suddenly Fails

Problem

Ubuntu repositories stopped updating.

Error

Missing Signed-By in sources list.

Cause

Repository configuration changed after upgrading Ubuntu.

Solution

Review:

/etc/apt/sources.list

/etc/apt/sources.list.d/

Correct the repository definitions.

Lesson Learned

Operating system upgrades deserve their own testing plan.

Case 3: The Database Was Correct but the Business Was Wrong

Problem

The schema looked perfect.

The application still behaved incorrectly.

Cause

The business rules were misunderstood.

Solution

Return to the business discussion.

Forget the tables.

Understand the process.

Lesson Learned

> Business understanding is more valuable than database design.

Case 4: The Feature Worked but Users Were Confused

Problem

Everything technically worked.

Users still struggled.

Cause

The workflow made sense to developers but not to users.

Solution

Redesign the interface.

Improve the language.

Add guidance.

Lesson Learned

> Usability matters.

Case 5: The Bug That Was Actually Documentation

Problem

The team remembered the requirement differently.

Cause

The decision was never documented.

Solution

Create an ADR.

Document the rule.

Lesson Learned

> Undocumented decisions eventually become bugs.

Troubleshooting Principle

> Verify first. Assume nothing.

APPENDIX B

The Cursor Learning Journey

Funny Moments, Surprises, and Discoveries

Every long project creates stories.

Some are technical.

Some are educational.

Some are simply funny.

This appendix preserves the moments that made the GrowSharely journey memorable.

The Mystery of the Missing .winwinsp Folder

At one point it seemed as though the `.winwinsp` folder had disappeared.

Questions immediately appeared.

Was it deleted?

Was it ignored?

Was it in another branch?

Eventually the answer was much simpler.

The folder was there all along.

The lesson was unforgettable.

Lesson Learned

> Before solving a difficult problem, make sure the problem actually exists.

The Editor Window Mystery

A discussion lasted much longer than expected because everyone was talking about different windows.

The browser.

The editor.

The file explorer.

Eventually everyone realized they were looking at different things.

Lesson Learned

> Precise communication saves enormous amounts of time.

The Case of the Confident AI

Sometimes AI generated beautiful solutions.

To entirely different problems.

The confidence was impressive.

The correctness was not.

Lesson Learned

> Confidence and correctness are not the same thing.
>
> Always review.

The Unexpected Architect

At the beginning of the project, AI was expected to generate code.

Eventually it became:

- an architect,
- a reviewer,
- a mentor,
- and occasionally, a teacher.

That transformation surprised everyone.

Lesson Learned

The capabilities of a tool often exceed our initial expectations.

The Endless Principles

One principle became two.

Two became ten.

Ten became dozens.

Eventually the project accumulated an entire philosophy.

Lesson Learned

Principles are usually discovered, not invented.

The Biggest Surprise

The team expected to build software.

Instead they built:

- software,
- documentation,
- principles,
- workflows,
- and a handbook.

Lesson Learned

The most valuable output of a project is sometimes knowledge.

Journey Principle

Every project leaves behind stories. Preserve them.

APPENDIX C

Architect Principles Index

This appendix provides a quick reference to the most important principles discovered during the GrowSharely journey.

MBH Principles

- Business First, Code Second
- Documentation Is Part of the Product
- Finish Before Optimizing
- Build for Maintainability
- Think Like the User
- AI Is a Partner, Not a Replacement

Architect Principles

- Business Rules Before Database Design
- Ownership Is Not Payment
- Store Facts, Compute Summaries
- Dashboards Should Answer Questions
- Complexity Should Be Introduced Slowly
- Documentation Is Architecture

Cursor Principles

- Plan Before Coding
- Never Silently Revert Decisions
- Freeze Stable Foundations
- Build in Small Milestones
- Documentation Is the Source of Truth

Governance Principles

- AI Proposes, Humans Decide
- Human Approval Is Mandatory
- Trust but Verify
- Production Requires Higher Standards

Final Principle

> Build software that people can trust.

APPENDIX D

Glossary

ADR: Architecture Decision Record.

A document that explains why an important decision was made.

UAT: User Acceptance Testing.

Business users validate that software works as expected.

RC: Release Candidate.

A version intended for final testing before production release.

Domain

A business area with its own responsibilities and rules.

Milestone

A small, manageable unit of work with a specific goal.

Audit Trail

A historical record showing who performed an action and when.

Read Model

A calculated summary derived from stored facts.

Approval Workflow

A process requiring another person to review and approve an action.

Principle

A rule or lesson that guides future decisions.

About the Author

Mohammad Belal Hossain (MBH) is a software engineer, entrepreneur, educator, author, and lifelong learner from Bangladesh.

With more than two decades of experience in software development and business technology solutions, he has worked on a wide range of applications, including business management systems, SaaS platforms, educational solutions, and real-estate technology products.

He is the founder of GrowSharely and the Managing Director of Win-Win Service Provider, where he continues to explore practical applications of software architecture and artificial intelligence in solving real business problems.

Beyond software development, he remains actively involved in education and writing and believes that learning, teaching, and sharing knowledge are lifelong responsibilities.

This handbook reflects his journey of building software through disciplined engineering, continuous learning, and meaningful collaboration between humans and artificial intelligence.

His guiding belief is simple:

"Technology should solve real problems, remain understandable to ordinary people, and leave future builders a better path to follow."

Acknowledgements

This book would not exist without the countless discussions, experiments, mistakes, and discoveries that shaped the GrowSharely journey.

Special appreciation goes to:

- The GrowSharely users who inspired the business requirements.
- Cursor AI for becoming an implementation partner.
- ChatGPT for acting as an architect, reviewer, mentor, and writing companion.

And finally, To every future builder who reads this book:

May your projects teach you as much as this one taught us.

About the AI Contributors

This handbook was prepared with the assistance of artificial intelligence tools including ChatGPT and Cursor AI.

Cursor AI for becoming an implementation partner.

ChatGPT for acting as an architect, reviewer, mentor, and writing companion.

These tools assisted with:

- Architecture discussions
- Documentation
- Review
- Writing assistance
- Learning and mentoring

All final decisions, reviews, and approvals remained the responsibility of the human author.

Final Words

If this handbook leaves you with only one idea, let it be this:

> **Great software is not built by writing code faster.**

It is built by:

- understanding people,
- making thoughtful decisions,
- documenting what matters,
- and continuously learning.

The journey never truly ends. One project simply prepares you for the next.

Alhamdulillaah.

Version 1.0 is complete.

"Every project builds software. Great projects also build wisdom."
— Mohammad Belal Hossain (MBH)
