Almirah Framework. User Manual

Overview

Almirah is an open-source Application Lifecycle Management (ALM) framework built around one idea: all project documents – requirements, architecture, test protocols, decisions, and risks – are plain Markdown files kept in version control, and everything else is generated from them. The `almirah` command-line tool parses those files and renders a set of interlinked static HTML pages with full traceability: every reference becomes a hyperlink, reverse links are computed automatically, and traceability, coverage, and implementation matrices are built for you.

There is no special server, no database in the common sense, and no proprietary format. Documents are edited in any text editor, reviewed as pull or merge requests of your code review tool, and document history is the git history. This makes the framework a natural fit for teams that already live in a source-code workflow – and for regulated environments where document control and traceability evidence are mandatory.

This manual walks the whole journey: installing the tool, creating a first project, authoring documents in the Almirah Markdown dialect, tracing specifications to tests and source code, and managing decisions and risks. The two closing chapters are references for the command line and the project configuration file.

Table of Contents

Overview *1*

Installation *4*

Getting Started *4*

Project Structure *5*

Writing Documents *6*

Supported Markdown 6

Controlled Paragraphs 7

Up-links 7

Controlled Tables 7

Authoring Rules and Limitations 8

Traceability and Coverage *8*

How Linking Works 8

Traceability Matrices 8

Coverage Matrices 9

Document Statistics 9

Broken Links 9

Linking Source Code *9*

Test Management *10*

Protocols and Runs 10

Executing a Run 10

Combining Protocols 11

Decision Records *11*

Decision Record Types and Naming 11

Anatomy of a Record 12

Affected Documents, Resolved and Proposed 12

The Overview Page 12

Change Management with No Extra Tool 13

Risk Management *13*

Anatomy of a Risk Record 13

Configuring a Register 13

One Mechanism, Many Methodologies 14

The Risks Overview 14

The Rendered Site *15*

Worked Examples *16*
Command Line Reference *16*
Project File Reference *17*
License Information *18*
Document History *18*

Installation

Almirah is distributed as a Ruby gem and requires Ruby 3.0 or newer. Check the installed Ruby version first:

```
$ ruby -v
ruby 3.2.0 (2022-12-25 revision a528908271) [arm64-darwin22]
```

If your Ruby is older, install a current one with a version manager of your choice (rbenv, rvm, asdf) or from ruby-lang.org. Then install the gem:

```
$ gem install Almirah
```

> *Note: the gem name starts with a capital letter – "Almirah", not "almirah". The capital letter is essential for installation.*

After installation, the `almirah` command is available in the terminal. Run `almirah help` to list the available commands.

To upgrade to the latest release later, run `gem update Almirah`.

Getting Started

The quickest way to see the framework in action is to let it scaffold a small example project:

```
$ almirah create my_project
```

The command creates a miniature of a real project – two specifications, three test protocols, two executed test runs, and one decision record:

```
my_project/
├── decisions/       adr-001-start-project-decision.md
├── specifications/
│   ├── req/         req.md
│   └── arch/        arch.md
└── tests/
    ├── protocols/   tp-001  tp-002  tq-001
    └── runs/        001  010
```

Now process the project:

```
$ almirah please my_project
Project file not found: project.yml
parsing specifications ..... 2 ok
parsing test protocols ..... 3 ok
parsing decisions .......... 1 ok
parsing risk records ....... 0 ok
traceability matrices ...... 1 ok
coverage matrices .......... 2 ok
decision links ............. 1 ok
risk links ................. 0 ok
rendering HTML ............. my_project/build/index.html
```

The “Project file not found” warning is harmless here: without a `project.yml` Almirah falls back to defaults. You will add a configuration file once you want to select input specifications, configure risk registers, or link source-code repositories – see the Project File Reference chapter.

Open the result in a browser:

```
$ open my_project/build/index.html
```

The index page lists every document with its statistics and provides full-text search (in a web-server environment only). Every requirement reference in every document is now a hyperlink, and traceability and coverage matrices are rendered next to the documents.

To process a project with an executed test run, pass the run identifier:

```
$ almirah please my_project --run 010
```

The coverage matrices then reflect the results of that exact run. Test runs are explained in the Test Management chapter.

Project Structure

An Almirah project is a folder with a fixed layout. Every part is optional – a project may hold only specifications, or only decisions – but the folder names are fixed:

Folder	Content
specifications/	One sub-folder per document, each holding a Markdown file of the same name (for example specifications/srs/srs.md) plus an optional img/ folder for pictures
tests/protocols/	Reusable, non-executed test protocols, one folder per protocol (tp-001/tp-001.md)
tests/runs/	Executed test runs: numbered folders holding copies of protocols filled with results (001/tp-001/tp-001.md)
decisions/	Decision records – architecture decisions, issues, enhancement proposals – with an optional img/ folder
risks/	Risk registers: one sub-folder per register, one Markdown file per risk record, with an optional img/ folder
project.yml	Optional configuration file at the project root
build/	Generated output – never edit it by hand, it is fully rewritten on each build

The `project.yml` file declares which specifications are the design inputs, which external source-code repositories participate in tracing, how risk registers are computed, and appearance settings. The Project File Reference chapter documents every key.

Writing Documents

Supported Markdown

Almirah ships its own Markdown parser, tuned for specification documents. It understands the following elements:

Element	Syntax	Notes
Frontmatter	YAML block between two lines of three hyphens	Sets the document title and metadata: title, revision, date, author
Document title	`% Title` on the first line	Pandoc-style alternative to frontmatter
Headings	# to ######	Automatically numbered; each heading gets a linkable anchor and appears in the navigation pane
Paragraphs	plain text	With inline bold, italic, code spans, and links
Controlled paragraphs	ID in square brackets at line start	The Almirah extension – see the next section
Images	`![caption](img/file.svg)`	The document-local img/ folder is copied into the html build; images open in a zoom overlay on click
Unordered lists	lines starting with a hyphen or asterisk	Nesting by indentation is supported
Ordered lists	lines starting with a number and a dot	Multi-digit numbering continues correctly
Tables	pipe syntax with a separator line	Grid-table border lines made of plus and minus signs are tolerated and ignored
Notes	lines starting with the greater-than sign	Rendered as a highlighted note block
Code blocks	fenced by lines of three consecutive backticks, with an optional language name	Syntax highlighting via Rouge
TODO markers	a line starting with the word TODO and a colon	Rendered as a visible TODO block

Text can link to other documents of the same project by relative path, to headings, and to external resources:

> *Example 1:* `[a link to another document of this project](./../srs/srs.md)`

> *Example 2:* `[an external link](https://almirah.site)`

Controlled Paragraphs

Plain Markdown has no notion of a requirement. Almirah adds one – a small extension that turns selected paragraphs and table rows into controlled items: uniquely identified, referencable, traceable units. Everything else stays ordinary Markdown.

A paragraph becomes controlled when its line starts with an ID in square brackets, followed by the text:

> *Example: "[SRS-001] The software shall measure the heart rate in the range from 30 to 250 bpm."*

In the rendered HTML, such a paragraph turns into an anchored row that any other document can reference. Conventions and rules:

- The ID prefix matches the owning document – items of *srs.md* carry IDs starting with *SRS*.
- Every ID must be unique within its document; duplicates are counted and reported in the document statistics.
- The ID must open its own top-level paragraph line. A bracketed ID inside a list item, a note block, a heading, or a table cell does not create a controlled item.

Up-links

A controlled item references a higher-level item with the greater-than notation – an up-link placed at the end of the *Controlled Paragraph*:

> *Example: "[SAD-010] The Measurement module computes the heart rate. >[SRS-001]"*

Several up-links are separated by commas:

> *Example: "[SRS-001] The software shall allow creating Controlled Items. >[ARCH-005], >[SYS-005]"*

During processing, every up-link resolves into a hyperlink, and the down-link – from the referenced requirement back to the item that references it – is generated automatically. You never maintain back-references by hand. The same pairs feed the traceability and coverage matrices described in the next chapter.

Controlled Tables

A Markdown table becomes a controlled table when its rows carry up-links in the last column. This is how test protocol steps reference the requirements they verify:

Test Step #	Test Step Description	Result	Req-ID
1	Some preparation step		
2	Verify the heart rate range	pass	>[SRS-001]

The same mechanism powers the “Affected Documents” table of decision records and risk records, where each row states a proposed requirement text and links it to the requirement it updates. If the referenced ID exists, the cell renders as a hyperlink; if it does not exist yet – a proposed requirement – it stays visibly unresolved, which is useful while the requirements work is still pending.

Controlled tables are recognized in test protocols, decision records, and risk records; in ordinary specifications a table is always a plain Markdown table.

Authoring Rules and Limitations

The parser is line-oriented and deliberately simple. A few rules keep documents rendering exactly as intended:

- Never begin an ordinary paragraph with an opening square bracket. Any paragraph whose line starts with a bracketed token is treated as a controlled-item ID – rephrase so the bracket is not the first character of the line.
- Inline code spans are delimited by single backticks only. Every backtick on a line is paired with the next one, so multi-backtick spans (the CommonMark way of embedding a literal backtick) are not supported. To mention the backtick character in prose, write the word “backtick”.
- Inline HTML is not interpreted. Tags are escaped and shown as literal text.
- Images work in their own paragraph only – the image syntax inside a table cell or a note block renders as literal text.
- Code fences are not opaque. Headings, controlled paragraphs, lists, and tables inside a fenced code block are still parsed as document structure. Avoid putting Markdown examples that contain headings or bracketed IDs at line starts inside fences – indent such lines with at least one space, or use note blocks for one-line examples instead.

Traceability and Coverage

Controlling traceability is the primary purpose of the framework: between specifications on one side, and between specifications, tests, decision records, risks, and source code on the other.

How Linking Works

Processing is two-pass. The first pass collects every controlled-item ID in the project; the second resolves all references against them. Each up-link you write becomes a hyperlink, and the opposite down-link is generated automatically. A requirement therefore always shows which design items realize it, which test steps verify it, and which decisions affect it – without any manual bookkeeping.

Traceability Matrices

For every pair of linked specification documents (say, requirements and architecture), Almirah renders a traceability matrix: each row pairs a higher-level item with the lower-

level items that trace to it. Items with no counterpart stand out immediately – that is how a requirement nobody designed for, or a design item justified by nothing, gets caught.

Coverage Matrices

Test protocols link their steps to requirements or any other specification via controlled tables. From these, Almirah renders a coverage matrix per specification: which requirements are verified by which test cases, and which are not covered at all. A partially covered document is visible at a glance. When the project is processed with a selected test run, the matrix carries the pass and fail results of that run.

Document Statistics

The index page shows, for every specification:

- Number of Controlled Items;
- Number of Items with Up-links;
- Number of Items with Down-links;
- Number of Items with Test Coverage;
- Duplicated Item IDs found;
- Last used Item ID.

The last two are practical daily helpers: duplicated IDs are authoring mistakes to fix, and the last used ID tells you which number to take next.

Broken Links

Dangling references do not fail the build – they are counted and named in the console output and highlighted in the rendered HTML:

```
decision links ............. 38 ok
broken links ............... 1
  srs: sys.md
rendering HTML ............. ./build/index.html
```

Treat a non-zero broken-links count as a review finding: either the target item was renamed or deleted, or the reference has a typo.

Linking Source Code

Implementation files join the traceability chain via ordinary comments. A comment carrying a REQ tag links the code that follows it up to a requirement:

```
<REQ> Computes the heart rate from the PPG signal. >[SRS-001] </REQ>
def compute_heart_rate(signal)
```

Supported languages for syntax highlight: C, C++, Python, Java, Ruby, JavaScript, TypeScript, Go, and Rust – the tag syntax is the same in each, placed inside whatever comment form the language uses.

The repositories to scan are declared in `project.yml`:

```
repositories:
  - name: Almirah.Code
    path: ../Almirah.Code/lib
```

Almirah collects the tagged locations into an implementation matrix, so every requirement shows the code that realizes it. The full chain then reads: requirement – design – code – test, with a hyperlink at every hop, in both directions.

> *Important: repository paths are resolved against the current working directory, not against the project argument. The intended layout is a flat one – the documentation project and the code repositories are sibling folders – and the build is run from inside the documentation project ("almirah please ."). If the paths do not resolve, source linking is silently skipped: the build succeeds, but the implementation matrix and the source-code links are missing.*
>
> *Performance note: the Almirah implementation is not optimized yet for the big codebases. Each source file in the linked repository needs to be processed and rendered in the case of <REQ> </REQ> pair is found. So, please be ready for the speed limitations and resource consumption while processing gigabytes of code source files.*

Test Management

Protocols and Runs

The framework separates test cases from test executions:

- Protocols under *tests/protocols/* are reusable, non-executed test case documents. Coverage computed from them answers the question "is there a test?" – not "did it pass?".
- Runs under *tests/runs/* are numbered folders holding copies of protocols filled in with actual results. Each run folder captures one execution campaign.

A protocol may contain a summary table (tester name, date, software version) and one or more test step tables with the columns: Test Step, Description, Expected Output, Actual Output, Result, Req-ID. The Req-ID column carries up-links to the requirements each step verifies, which makes the step table a controlled table.

You are free to come up with your own test protocol structure. The presence of a *Controlled Table* in a protocol is the only rule you need to follow.

Executing a Run

To execute tests, copy the protocols into a new numbered run folder, perform the steps, and fill in the Actual Output and Result columns with pass or fail, attaching evidence images where useful:

```
$ cp -r my_project/tests/protocols/tp-001 my_project/tests/runs/002/
```

Then process the project against that run:

```
$ almirah please my_project --run 002
```

The coverage matrices now reflect that exact run – with pass and fail results, and other fields populated in the test run folder.

Combining Protocols

For campaigns that are executed as one long session, a helper merges all protocols into a single document stored as build/combined.md:

```
$ almirah combine my_project
$ almirah combine my_project --run 002
```

The first form combines the protocol templates; the second combines the protocols of the selected run.

Decision Records

Almirah is able to track project work without a separate issue tracker: every architectural decision, work item, bug fix, or enhancement proposal is a Markdown file in the *decisions/* folder, versioned right next to the specifications it affects, and parsed, linked, and rendered like every other document.

Decision Record Types and Naming

The record type is encoded in the file name: a letter prefix, a sequence number, and a short descriptive slug.

Prefix	Type	Purpose
adr	ADR	Architecture Decision Record – a substantial design or process decision
issue	ISSUE	A bug or defect and the decision on how to fix it
enh	ENH	An enhancement proposal
wi	WI	A work item

> *Example: a file named "adr-170-introduce-decision-records.md" renders as record ADR-170.*

Every record should carry a YAML frontmatter title, which is what the rendered pages display.

> *Note: You are free to use your own decision records types system with any abbreviations (BUG, FIX, EPIC, UC). Almirah cares only abut the folders decision records are placed and file name structure (epic-324-my-favorite-feature.md)*

Anatomy of a Record

A record is structured by conventional sections proposed by Michael Nygard with some extensions:

- Status – a small table listing the record's history, with the current state marked by an asterisk in the first column. Typical statuses: Proposed, Accepted, In-Progress, Implemented; a record replaced by a later decision receives Superseded.
- Context and Decision – the problem and the decision taken, in prose, with images welcome: diagrams live in the record's img/ folder and are copied into the build.
- Scope – a work-breakdown table with owners, dependencies, and focused and safe effort estimates; Out of Scope states explicitly what the decision does not include.
- Consequences and Alternatives Considered – the classic ADR sections.
- Affected Documents – the traceable heart of the record: a *Controlled Table* where each row states a proposed requirement text and links it up to a specification item the record affects (creates or updates).
- Software Versions and References – which release the decision targets, and related links.

A status table is a simple markdown table as shown below:

	Date	**Status**
	14-05-2026	Proposed
	14-05-2026	Accepted
*	15-05-2026	In-Progress
	24-05-2026	Implemented

Future-dated rows may be written upfront for planning; the asterisk marks where the record currently stands.

Affected Documents, Resolved and Proposed

The Affected Documents table ties a decision into the traceability graph. A row referencing an existing requirement resolves into a hyperlink, and the requirement gains a *DR* (Decision Record) mark in the opposite direction – controlled items show the decision records that affect them, expanding to the full list on click. A row proposing a new requirement stays visibly unresolved until that requirement is actually written – so a half-implemented decision is impossible to mistake for a finished one.

The Overview Page

All records are collected into a rendered overview page, linked from the top navigation bar of every page. The overview lists each record's sequence number, type, current status, owners, target date, and title, and charts the project state: records by type,

records by status, and the velocity of implemented records over time. It is a project backlog generated from plain files.

Change Management with No Extra Tool

Because records are files, change management rides on source control: a new decision is a branch and a pull request, review happens in a regular code-review tool, and history is the git log. Nothing about the flow is Almirah-specific – which is precisely the point. The framework's own development is managed this way.

With git and code-review tool you can easily identify when particular decision was introduced, who reviwed and approved it for implementation, and what wording was proposed for a controlled specification items initially. This is a huge benifit of keeping decision records in a form of text files in source control.

Risk Management

Risk management follows the same pattern as everything else: risks are Markdown files in a *risks//* folder, and the register – with computed risk priority numbers, color bands, and links into the requirements – is generated from them.

Anatomy of a Risk Record

A risk record is a Markdown file whose sections carry the data:

- a Status history table – typical statuses are Identified, Analyzed, Mitigating, Accepted, and Closed, with the current one marked by an asterisk;
- the descriptive sections your methodology calls for – Threat, Failure Mode, Mitigation, and so on;
- one numeric section per risk-priority input, holding a single number:

```
# Severity
9
# Occurrence
4
# Detection
6
```

- an Affected Documents controlled table linking the proposed controls up to requirements, and a Monitoring section.

Diagrams – fault trees, attack trees – live in the record's img/ folder and render both on the record page and in the register overview.

Configuring a Register

A risk register is declared in `project.yml`: which folder it lives in, which record sections appear as register columns, and how the *Risk Priority Number* (RPN) is computed:

```
risks:
  - folder: fmea
    columns: [Failure Mode, Severity, Occurrence, Detection, Mitigation, A
ffected Documents, Status]
    rpn:
      - name: Initial
        inputs: [Severity, Occurrence, Detection]
        thresholds:
          acceptable: 50
          unacceptable: 200
      - name: Residual
        inputs: [Residual Severity, Residual Occurrence, Residual Detectio
n]
        thresholds:
          acceptable: 50
          unacceptable: 200
```

Each RPN group multiplies the values of its input sections. The thresholds drive the color coding: at or below the acceptable value is green, at or above the unacceptable value is red, and the band between is amber. A group with a single input – for example, a precomputed CVSS score – is surfaced unchanged, with no product computed. A record without residual scores simply leaves its Residual cell blank – an honest picture of what is mitigated and what is not.

One Mechanism, Many Methodologies

The RPN configuration is generic enough to express very different risk approaches:

- FMEA (IEC 60812) – Initial and Residual groups over Severity, Occurrence, and Detection.
- Fault Tree Analysis (IEC 61025) – criticality as Severity times Likelihood, one record per minimal cut set, fault trees as SVG images.
- Security risks with CVSS (ISO 27005) – a single-input group surfacing the CVSS base score, with thresholds on the CVSS severity boundaries.
- Project risks – the simplest form: one Probability times Impact group, no residual scoring.

Each of these is available as a live worked example – see the Worked Examples chapter.

The Risks Overview

When at least one risk record exists, the top navigation bar gains a Risks entry leading to the registries overview: the register tables with their color-coded RPN columns, links to the per-record pages, and the count of open risks – records whose status is not Closed. Your open-risk list is always one build away.

The register table is not authored anywhere – it is another view of the same records. A risk record is a plain Markdown document, and its top-level sections are the data: for every column of the register, Almirah takes the record's section whose heading matches the column name and renders that section's content as the record's cell. Writing a Mitigation

section in a record is what fills the Mitigation column of its register – one section, one column.

The `columns:` list of the register's entry in `project.yml` controls which of those sections are visible in the table, and in which order. Sections not listed remain on the record's own page but do not appear in the register – so a record may carry as much analysis prose as it needs while the register stays a compact worksheet. A few columns receive special treatment:

- the record ID and Title columns are implicit and always come first;
- the Status column is reserved: it is filled from the record's current lifecycle status – the row marked with an asterisk in the Status table – never from a section named Status;
- an Affected Documents column renders as the record's linked requirement IDs, each a clickable link (or a highlighted unresolved reference), while the proposed text stays on the record page;
- the computed RPN columns are appended after the configured columns.

A register with no entry in `project.yml` still renders, with the implicit columns plus Status only.

The Rendered Site

Everything Almirah generates lands in the build/ folder of the project as a self-contained static site. Highlights of the reading experience:

- The index page lists every specification and protocol with its statistics, and provides (in a web-server environment) full-text search over all indexed content: for a search term it returns the source document, a link to the containing section, and a text snippet.
- The top navigation bar of every page carries a Documents link (the home icon) back to the index, a Decision Records link when records exist, and a Risks link when risk records exist.
- The navigation pane on each document page shows the automatically numbered section tree; it scrolls independently, so even very long documents remain navigable.
- Controlled items render as anchored rows showing their up-links, down-links, test coverage, and decision-record marks.
- Images open in a zoom overlay on click.
- All author-supplied text is HTML-escaped on output – markup in documents cannot inject scripts or tags into the rendered pages.
- An optional font_size number in `project.yml` sets the base text size in pixels of every rendered page, scaling body text and the navigation pane while headings, the top bar, and the footer keep their sizes.

For teamwork, a simple bash script can be written to do the following steps

1. Pull the latest state of documents from the git repository.
2. Process the pulled repository with the Almirah gem.
3. Copy Almirah output (*build/* folder) to a project web-server *public/* folder.

Running this script every 1 minute (for example) allows you to have a controlled content server for a team as a single source of truth.

Worked Examples

Each demonstration is a small, self-contained Almirah project living on its own branch of the Almirah.TDS repository. You can browse the rendered results live on almirah.site, read the sources on GitHub, or check a branch out and rebuild it yourself:

```
$ git clone https://github.com/OleksandrIvanovDev/Almirah.TDS.git
$ cd Almirah.TDS
$ git checkout demo-specifications-one_di
$ almirah please .
$ open build/index.html
```

Branch	Shows
demo-specifications-one_di	A three-level specification chain (requirements, architecture, detailed design) for an imagined bedside patient monitor: up-links, generated down-links, traceability and coverage matrices, including one deliberately uncovered item
demo-decisions-aff_docs	A decision record with SVG diagrams and an Affected Documents table: how resolved and still-unresolved requirement references render
demo-risks-fmea	An FMEA worksheet with Initial and Residual RPN groups; some records deliberately unmitigated
demo-risks-fta	Fault Tree Analysis with fault trees as SVG images and criticality computed from Severity and Likelihood
demo-risks-cvss	A security risk register with single-input CVSS scoring across all three color bands
demo-risks-project	The simplest register: project risks scored as Probability times Impact

Every demo branch also carries a README describing its document set in detail.

Command Line Reference

Command	Purpose
almirah create	Scaffold a new project with example specifications, protocols, runs, and a decision record
almirah please	Process the project: parse, link, check, and render HTML into */build/*

almirah please --run	Process the project against the selected test run from *tests/runs//*
almirah combine	Merge all test protocols into *build/combined.md*
almirah combine --run	Merge the protocols of the selected test run
almirah help	List the available commands

When the project declares linked repositories, run the build from inside the project folder ("almirah please .") so the relative repository paths resolve – see the Linking Source Code chapter.

Project File Reference

The optional `project.yml` at the project root configures the build. All keys are optional; a missing file produces the default behavior with a console warning.

```
font_size: 12
specifications:
  input:
    - arch
repositories:
  - name: Almirah.Code
    path: ../Almirah.Code/lib
risks:
  - folder: fmea
    columns: [Failure Mode, Severity, Occurrence, Detection, Mitigation, A
ffected Documents, Status]
    rpn:
      - name: Initial
        inputs: [Severity, Occurrence, Detection]
        thresholds:
          acceptable: 50
          unacceptable: 200
```

Key	Meaning
font_size	Base text size in pixels for every rendered page; absent means the built-in default
specifications / input	List of specification folder names treated as design inputs – the top of the traceability tree. Almirah will threat all the specifications mentioned in this element as one-to-many traceability items (default is on specifications to one specification)
repositories	Source-code repositories to scan for REQ tags: a name and a path relative to the working directory of the build
risks	One entry per risk register folder under risks/
risks / folder	The register's folder name
risks / columns	The record sections rendered as register columns, in order; without it the register shows the implicit columns plus Status

risks / rpn	Named RPN groups: each multiplies the values of its input sections
risks / rpn / inputs	The record sections whose numeric values feed the group; a single input is surfaced unchanged
risks / rpn / thresholds	The acceptable and unacceptable bounds driving the green, amber, and red color bands

License Information

The Almirah framework – the gem and its source code – is open-source software released under the MIT License, copyright Oleksandr Ivanov. In practical terms, this means you may use the tool freely, including for commercial work, and may modify and redistribute it, provided the copyright notice and the license text are preserved; the software is provided "as is", without warranty of any kind. The full license text ships in the LICENSE file of the Almirah.Code repository.

The license applies to the tool only, not to what you produce with it. Your Markdown documents remain entirely yours, and the generated HTML in the build/ folder carries no license obligations – you may keep it private, publish it, or deliver it to a customer under any terms you choose.

The gem builds on the following open-source components:

Component	Purpose	License
Thor	Command-line interface	MIT
Rouge	Syntax highlighting of code blocks	MIT
Orama	Client-side full-text search bundled into the rendered pages	Apache 2.0

Document History

Version	Description	Date
1.0	Initial version created for Almirah version 0.4.6	09-Aug-2026