

MDU / DOCUMENTATION

MDU

Plain Markdown, beautifully published

MDU	4
Setup	5
.NET Tool Installation	5
Portable Installation	5
Docker	6
Usage	6
tree	8
lint	8
Lint Rules Reference	9
Rule 2	12
Rule 3	12
Rule 5	13
fix	14
render	15
serve	16
page	17
pdf	18
Publish	19
GitHub	20
GitLab	20
Custom	21
Features	21
Callouts	22
Code Blocks	24
Diagrams	25
Footnotes	34
Formatting	34
Hero Banner	37

Icons and Emojis	39
Images	42
Inventory System	43
Keyboard Keys	43
Keyboard Key Reference	44
Math	50
Media Links	51
Navigation	52
Online Store	54
OpenAPI Spec	54
OpenAPI Spec example	56
Order Service	57
PDF Link	58
Repository Imports	58
Swagger Petstore example	59
Swagger Petstore OpenAPI 3.0 example	105
Tables	153
Architecture	154
Components	156
Processing Pipeline	159
Rendering	163
Other Tools	165

MDU

Plain Markdown, beautifully published

Lint, inspect, and render your Markdown files into a fast static documentation site - no lock-in, no build plugins.

[Get started](#)

[Browse features](#)

MDU is a .NET tool that turns a folder of ordinary Markdown files into a documentation site. Your pages stay plain Markdown that reads well on GitHub; MDU adds navigation, search, diagrams, and a polished theme when it renders them.

Get started

- [Setup](#) - Choose a .NET tool or portable installation
- [Usage](#) - Run the commands and preview your site
- [Publish](#) - Host the generated site on GitHub, GitLab, or any web server

Commands

Command	Does
mdu tree	Shows the page structure
mdu lint	Finds broken links and structure problems
mdu fix	Migrates MkDocs syntax and applies automatic fixes
mdu serve	Previews the site with live reload
mdu render	Generates the static site
mdu pdf	Combines all pages into one PDF
mdu page	Moves pages and updates every link to them

Write and organize content

- [Features](#) - Callouts, diagrams, math, icons, keyboard keys, and more
- [Navigation](#) - Order pages, add top tabs, and hide entries
- [Repository Imports](#) - Combine Markdown pages from multiple GitHub repositories

About MDU

- [Architecture](#) - How MDU reads, checks, and renders a documentation folder
- [Other tools](#) - Documentation site generators worth knowing

Setup

Install MDU globally as a .NET tool:

```
dotnettoolinstall--globalmdu
```

This installation requires the [.NET 10 SDK](#). For [repository-local installation](#), the [Docker image](#), or a [portable executable](#), see the detailed guides.

.NET Tool Installation

MDU is published on NuGet as the `mdu` [.NET tool](#) and runs as the `mdu` command.

Requirements

Install the [.NET 10 SDK](#) for your operating system. It includes the runtime that MDU needs.

Install globally

A global tool is available from any folder on your machine:

```
dotnettoolinstall--globalmdumdu--help
```

Update it later with:

```
dotnettoolupdate--globalmdu
```

Install in a repository

A local tool is pinned in the repository's tool manifest, so everyone on the team and CI can use the same version. Create the manifest once, then install MDU:

```
dotnetnewtool-manifestdotnettoolinstall--localmdudotnetmdu--help
```

Commit `.config/dotnet-tools.json`. Others can install the pinned version with:

```
dotnettoolrestore
```

Next steps

See the [Usage guide](#) to lint, preview, and render your documentation.

Portable Installation

Each [GitHub release](#) includes self-contained executables that do not require the .NET SDK or runtime.

Download the archive for your system, extract it, and run `mdu` or `mdu.exe`. To use the command from any folder, put the extracted executable on your `PATH`.

Operating system	Architecture	Notes
Windows	x64	Run <code>mdu.exe</code> .
Linux	x64	Requires glibc 2.35 or newer. Alpine Linux and other musl-based distributions are not supported.
macOS	Intel (x64) or Apple silicon (ARM64)	First launch may need an internet connection so Gatekeeper can verify the notarization ticket.

For example, run the executable from its extracted folder:

```
./mdu --help
```

On Windows, use `mdu.exe --help` in PowerShell or Command Prompt.

Next steps

See the [Usage guide](#) to lint, preview, and render your documentation.

Docker

The release workflow publishes [capjan/mdu on Docker Hub](#) for Linux amd64 and arm64. It tags each release with its version and updates `latest`.

Run MDU in the current folder by mounting it at `/work`:

```
dockerrun --rm --volume "$PWD:/work" --workdir /work --user "$(id -u):$(id -g)" capjan/mdulint
```

Render the site with:

```
dockerrun --rm --volume "$PWD:/work" --workdir /work --user "$(id -u):$(id -g)" capjan/mdurender
```

The compact container includes the Native AOT `mdu` executable, its Alpine runtime dependencies, Git, CA certificates, and its license notices. [mdu.import](#) and Git-derived last-changed dates are available when the mounted documentation folder has the required repository data. Public imports work without credentials; private imports need Git credentials configured inside the container. `mdu serve` listens on the container's loopback interface; use `mdu render` to generate a site from Docker.

Usage

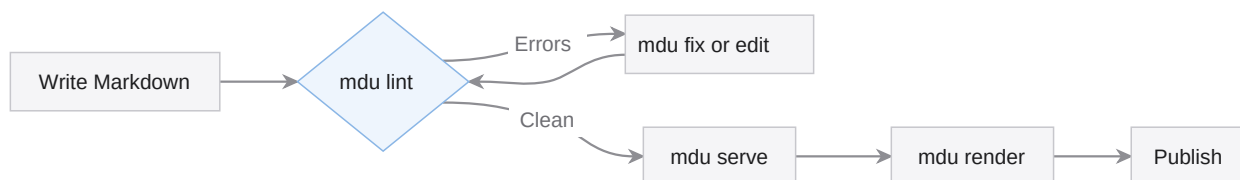
Quick start

Run the commands from your project folder. MDU finds the `docs` folder on its own.

```
mdutree# show the page structuremdulint# report broken links and structure problemsmduserve# preview the site with live reloadmdurender# write the static site to ./site
```

A typical workflow checks the pages, previews them while you write, and renders the site when you are

ready to publish. Click a step to open its command page.



`mdu serve` opens the preview in your browser and updates it whenever you save a page or a linked image. Press `Ctrl+C` to stop it. Rendering never changes your Markdown files.

A lint error names the file, line, rule, and problem:

```
/project/docs/Overview.md:7: MDU004 LocalLinkExists: Local link target './intro.md' does not exist.  
Linter detected 1 error.
```

Commands

Command	Syntax	Purpose
tree	<code>mdu tree [path]</code>	Print the detected page structure. Alias: <code>print</code>
lint	<code>mdu lint [path]</code>	Check pages and local links against the rules. Alias: <code>l</code>
fix	<code>mdu fix [path]</code>	Migrate MkDocs syntax and apply automatic fixes
render	<code>mdu render [path] [outpath]</code>	Write a static HTML site. Alias: <code>r</code>
serve	<code>mdu serve [path] [outpath]</code>	Serve a local preview that updates on changes
pdf	<code>mdu pdf [path] [outpath]</code>	Write one PDF file
page move	<code>mdu page move <source> <target></code>	Move a page or folder and update every link to it. Alias: <code>page mv</code>
page links	<code>mdu page links <file></code>	List the links that point to a file

Paths and file search

	Default
Source folder	<code>./docs</code> . Pass the path explicitly if it does not exist.
Output of render and serve	<code>./site</code> , with <code>documentation.pdf</code> when a page contains a PDF Link
Output of pdf	<code>./documentation.pdf</code>
Pages	Files matching <code>*.md</code>

```
mdurender ./ProjectDocs ./public/docsmduserve - -port8080
```

Common options

`tree`, `lint`, `fix`, `render`, `serve`, and `pdf` share these options:

Option	Effect
<code>-p, --pattern <pattern></code>	Search for files matching the pattern. Default: <code>*.md</code> .
<code>--hidden</code>	Include hidden files. Default: <code>on</code> ; <code>--hidden false</code> leaves them out.
<code>-?, -h, --help</code>	Show help for the command.

Command pages list the additional options. `mdu <command> --help` always shows the current help.

tree

`mdu tree` prints the titles of the detected Markdown pages as an indented tree. A folder's `README.md` is the parent page; the other pages in the folder and in its subfolders appear beneath it.

Usage

```
mdutree[PATH][options]
```

The alias `mdu print` runs the same command. See [Paths and file search](#) for the default source folder and [common options](#) for `--pattern` and `--hidden`.

Example

```
$ mdu tree ./ProjectDocs/Usage
Usage
  fix
  render
  tree
  pdf
  page
  serve
  lint
    Rule 5
    Rule 2
    Rule 3
    Lint Rules Reference
```

Each line is a page title, taken from the `title` front matter or the level-one heading. Pages appear in the order they are found on disk; the rendered navigation sorts them with [navigation.order](#).

Use another pattern when your pages have a different extension:

```
mdutree ./ProjectDocs --pattern "*.markdown"
```

lint

`mdu lint` checks the Markdown pages under a folder and reports each problem with its file, line, rule code, and message. It exits with code `1` when it finds errors and `0` otherwise; warnings do not change the exit code.

Usage

```
mdulint[PATH][options]
```

The alias `mdu l` runs the same command. See [Paths and file search](#) for the default source folder and [common options](#) for `--pattern` and `--hidden`.

Option	Effect
--------	--------

<code>--pretty</code>	Group results by file and use paths relative to the lint root.
<code>--color</code>	Colorize errors and warnings.
<code>--format <format></code>	text (default), json for scripts and tools, or github for annotations in GitHub Actions. <code>--pretty</code> and <code>--color</code> apply to text only.

Rules

Each diagnostic has a stable code and a descriptive name, for example `MDU004LocalLinkExists`:

```
/project/docs/Overview.md:7: MDU004 LocalLinkExists: Local link target './intro.md' does not exist.
```

- [Lint Rules Reference](#) - All codes, severities, checks, and fixes
- Guides for [Rule 2](#), [Rule 3](#), and [Rule 5](#)

The linter also lists orphaned pages: pages hidden with `navigation.visible: false` that no visible page links to. This list is informational and does not affect the exit code.

Tip

`mdu_fix` migrates MkDocs Material syntax and fixes `MDU006` automatically, including the links to moved pages. Run `mdu lint` again afterward.

Output for tools and CI

`--format json` prints one JSON document and nothing else, so scripts can read it directly. `file` is relative to the lint root and `path` is the full path. `rule` is `null` for rules whose messages carry only a code.

```
{
  "root": "/project/docs",
  "errors": 1,
  "warnings": 0,
  "diagnostics": [
    {
      "file": "Overview.md",
      "path": "/project/docs/Overview.md",
      "line": 7,
      "severity": "error",
      "code": "MDU004",
      "rule": "LocalLinkExists",
      "message": "Local link target './intro.md' does not exist."
    }
  ],
  "orphanedPages": []
}
```

`--format github` writes [workflow commands](#) that GitHub Actions shows as annotations on the affected lines of a pull request. Paths are relative to the working directory, so run the command from the repository root:

```
.github/workflows/docs.yml
```

```
-name: Lint documentation
run: mdu lint --format github
```

The exit code is the same for every format.

Examples

```
mdulint# the default docs foldermdulint./ProjectDocs# another foldermdulint./ProjectDocs--pattern "*.markdown"mdulint--pretty--color# readable output in a terminal
```

Lint Rules Reference

`mdu lint` reports diagnostics with a stable rule code and name. Errors make the command exit with code 1; warnings do not.

Rule summary

Code	Name	Severity	Check	More
MDU001	HeadingRequired	Error	Every Markdown page has a level-one heading that defines its title.	-
MDU002	DirectoryTitleMatch	Error	A section directory name matches the title of its README.md.	Rule 2
MDU003	LinkCasing	Error	The casing in a local link target matches the file-system entry.	Rule 3
MDU004	LocalLinkExists	Error	A local link or image target exists.	-
MDU005	DuplicatePageRepresentation	Error	A page is not represented by both a Markdown file and a same-named directory README.md.	Rule 5
MDU006	ReadmeWithoutSubpages	Warning	A section README.md is its directory's only Markdown page.	Rule 5
MDU007	MaterialAdmonition	Error	Material admonitions must be converted to GitHub Alerts.	fix
MDU008	MaterialTabs	Error	Material tabs must be converted to headings.	fix
MDU009	MaterialCards	Error	Material card grids must be converted to Markdown link lists.	fix
MDU010	UnsupportedMermaid	Error	A Mermaid diagram uses syntax MDU cannot render.	-
MDU011	CodeAnnotations	Error	Material code-annotation markers are unsupported.	fix
MDU012	MkDocsIndexPage	Warning	Rename the MkDocs index.md page to README.md.	fix
MDU013	MissingSectionReadme	Warning	A section with child pages needs a README.md page.	fix
MDU014	MkDocsPagesFile	Error	Migrate MkDocs .pages settings to Markdown front matter.	fix
MDU015	LegacySidebarMetadata	Error	Replace retired sidebar.* front matter with navigation.*.	-
MDU016	InvalidNavigationPlacement	Error	navigation must be a mapping and placement must be sidebar or top.	-
MDU017	NavigationPlacementLevel	Error	placement: top is only valid on first-level pages or section indexes.	-
MDU018	ExpandedTopNavigationRoot	Error	A top navigation root is always expanded; remove its ineffective navigation.expanded.	-

MDU001: HeadingRequired

Every Markdown page needs a level-one heading (`# Page title`) after any YAML front matter. Add a heading that describes the page when this diagnostic appears.

MDU002: DirectoryTitleMatch

For a section index named `README.md`, the directory name must match the heading title. The comparison ignores case, punctuation, spaces, hyphens, and underscores. If the title contains a colon, only the text before the first colon is compared. Set `navigation.title` when a section intentionally uses a different

navigation label. `mdu fix` migrates this field only when it is explicitly present in a `.pages` file.

For example, `# Keyboard Keys` matches the directory `Keyboard -Keys`. See [Rule 2](#) for examples and guidance.

MDU003: LinkCasing

The path in a local link must use the same casing as its target on disk. This catches links that may work on a case-insensitive file system but fail on a case-sensitive one. Query strings and fragments do not affect the path check. This also applies to local destinations in supported Mermaid, context, and PlantUML node links. Correct the target path to match the file or directory name. See [Rule 3](#) for an example.

MDU004: LocalLinkExists

Relative local links, image targets, and destinations in supported Mermaid, context, and PlantUML node links must point to an existing file or to a directory containing a `README.md`. Query strings and fragments are ignored when resolving the path. External URLs and fragment-only links are not checked by this rule.

Fix the destination or add the missing target. A casing mismatch for an existing target is reported as MDU003 instead.

MDU005: DuplicatePageRepresentation

A page cannot have both a Markdown file and a same-named directory containing `README.md`. For example, `Setup.md` and `Setup/README.md` represent the same page; keep one representation. See [Rule 5](#) for valid and invalid layouts.

MDU006: ReadmeWithoutSubpages

This warning appears when a section directory's `README.md` is its only Markdown page and there is no same-named Markdown file beside the directory. The section can be represented by a single file instead. Run `mdu fix` to move the `README.md` beside the directory and update affected links; see [fixing MDU006](#).

MkDocs Material compatibility

Rules MDU007- MDU014 identify MkDocs-specific syntax and structure that needs migration. Run `mdu fix` to convert the supported constructs. An MDU010 error means a Mermaid diagram uses [syntax that MDU cannot render](#); simplify the diagram, or it stays a code block in the rendered page.

Navigation front matter

MDU015 rejects the retired `sidebar.*` fields. Rename them to `navigation.*` directly; `mdu fix` does not convert these fields. MDU016 reports malformed navigation metadata or a `placement` value other than `sidebar` or `top`. MDU017 requires `placement: top` to be set on a first-level Markdown page or the `README.md/index.md` of a first-level folder. MDU018 rejects `navigation.expanded` on a top navigation root because that root is always open.

Informational output

The linter also lists pages marked `navigation.visible: false` that no visible page links to. These orphaned pages are informational; they do not have a rule code and do not affect the exit code.

[Back to lint](#)

Rule 2

Important

The name of a folder that contains a `README.md` must match the title of that `README.md`. Code: `MDU002 DirectoryTitleMatch`.

Why

A `README.md` is the page readers open when they enter a folder. When the folder name matches its title, readers know what the section covers while they browse the repository, and the navigation of the rendered site shows the same name.

Example

notes/README.md

```
#Notes
```

```
These are my notes.
```

Folder	Title	Result
notes/	# Notes	Matches
Keyboard-Keys/	# Keyboard Keys	Matches; case, spaces, hyphens, and underscores are ignored
Rule2/	# Rule 2: Folder names	Matches; only the text before the first colon counts
docs/	# Overview	Rename the folder or the title

Set `navigation.title` when a section should show a different label in the navigation.

Rule 3

Important

The casing of a local link must match the casing of its target on disk. Code: `MDU003 LinkCasing`.

Why

Windows and macOS usually ignore the case of file names, Linux and most web servers do not. A link with the wrong case works while you write, then breaks after you publish.

Example

Markdown

```
Return to the [overview](../README.md).
```

If the file is actually named `readme.md`, this link works on Windows but fails on a Linux web server. Change the link or rename the file so both use the same case.

Rule 5

Important

A page must have only one representation. Codes: `MDU005 DuplicatePageRepresentation` (error) and `MDU006 ReadmeWithoutSubpages` (warning).

Examples

Either layout is valid on its own:

```
docs/Setup.md
```

```
docs/Setup/README.md
```

Both at once are an error, `MDU005`, because they describe the same page:

```
docs/Setup.md
```

```
docs/Setup/README.md
```

A folder with only a README

A folder whose only Markdown page is its `README.md` is valid but unnecessary, because `Setup.md` would do the same. The linter reports the warning `MDU006`, which does not make `mdu lint` fail.

Tip

Run `mdu fix` to move the page to a same-named file beside the folder. It updates the links to the page and the relative links inside it.

A `README.md` with Markdown subpages is the parent page of its section and does not trigger the warning.

fix

`mdu fix` migrates MkDocs Material syntax to portable Markdown, normalizes the MkDocs page structure, and applies the linter's automatic fixes. It changes files in place, so review the changes before you publish.

Usage

```
mdufix[PATH][options]
```

See [Paths and file search](#) for the default source folder and [common options](#) for `--pattern` and `--hidden`. Then check the result:

```
mdufix./ProjectDocsmdulint./ProjectDocs
```

If there is nothing to fix, the command reports `No fixable issues found.` and changes nothing.

Tip

Commit or stash your work first. `git diff` then shows exactly what `mdu fix` changed.

Fixing MDU006

When a section folder contains only a `README.md`, `mdu fix` moves that page beside the folder: `docs/Ru le3/README.md` becomes `docs/Ru le3.md`.

- Links to the old page are updated, including links to the section folder and reference links.
- Relative links inside the moved page are adjusted, so they still reach the same pages and assets.
- Query strings and fragments are kept.
- If the folder still contains assets or other files, it stays in place.

Migrating MkDocs Material Markdown

The linter reports Material-specific syntax as [migration issues](#). `mdu fix` converts what it can:

MkDocs Material	Becomes
Admonitions and collapsible details	GitHub alerts; content stays visible, unknown types become NOTE
Content tabs	Headings
Card grids	Markdown link lists with their descriptions
{ .annotate } markers	Removed; code comments and numbered explanations stay
index.md	README.md
Section without a README.md	A new README.md named after the folder
.pages titles and order	navigation.title and navigation.order front matter; the .pages file is removed
Blog and tag indexes	Static Markdown link lists

MDU012 (`index.md`) and MDU013 (missing section `README.md`) are warnings, so ordinary Markdown

projects do not fail lint because of these naming conventions. Unsupported Material syntax and `.pages` files remain errors.

Good to know:

- `navigation.title` is only added when a `.pages` file sets a title. A normal MDU002 title mismatch is yours to resolve, or to keep with an explicit `navigation.title`.
- YAML is validated before files are moved or `.pages` files are removed.
- Retired `sidebar.*` front matter is not migrated; rename it to `navigation.*` yourself.
- Mermaid blocks with syntax that MDU cannot render stay unchanged and are reported as errors. `mdu fix` does not rewrite diagrams.

render

`mdu render` converts the Markdown pages into a static HTML site and exits. It writes the pages and linked local assets to the output folder, `./site` by default. If any page contains `[Download the PDF]` (`mdu:pdf`), MDU also generates a combined `documentation.pdf` there and resolves the link to that file. It renders like any other link. Your Markdown files stay unchanged.

Usage

```
mdurender [PATH[OUTPATH]] [options]
```

The alias `mdu r` runs the same command. See [Paths and file search](#) for the default folders and [common options](#) for `--pattern` and `--hidden`.

Option	Effect
<code>--editPageRoot <url></code>	GitHub edit URL of the documentation folder. Adds an edit icon beside each page's level-one heading, or an Edit this page link in the footer when there is none, and a repository button with the owner/repository name to the header.

Examples

```
mdurendermdurender./ProjectDocs./public/docsmdurender./docs./site--pattern"*.markdown"--editPageRoot"https://github.com/example/project/edit/main/docs"
```

Site configuration

An optional `mdu.yml` in the documentation folder sets the site name, light and dark logos and favicons, and theme colors. MDU copies the referenced files into the output folder.

`docs/mdu.yml`

```
site name: Documentation
logo:
  light: assets/logo-light.svg
  dark: assets/logo-dark.svg
favicon:
  light: assets/favicon-light.svg
  dark: assets/favicon-dark.svg
colors:
```

```
light:
  background: "#ffffff" text: "#1d1d1f" primary: "#0066cc" dark:
  background: "#1d1d1f" text: "#f5f5f7" primary: "#2997ff"
```

The background color also applies to the header, navigation, sidebar, and dialogs. Secondary text, lines, and code surfaces are mixed from the text and background colors, and hover backgrounds from the primary color, so they match a custom palette.

Page front matter

YAML front matter at the top of a page controls its title, metadata, and navigation entry:

```
---title: Installing MDU
description: Install the MDU .NET tool globally or per repository.
icon: material-download
navigation:
  title: Setup
  order: 10---
```

Key	Effect
title	Browser title and fallback navigation title
description	Page meta description
icon	Icon shortcode shown beside the level-one heading, such as material-download
navigation.title	Label in the sidebar, tabs, and breadcrumbs
navigation.order	Sort order among sibling pages
navigation.visible	false hides the entry; the page stays reachable by its URL
navigation.expanded	true opens the section in the sidebar initially
navigation.placement	top makes a first-level page or section a top navigation tab
pdf.visible	false leaves the page out of mdu pdf

Both the nested form shown above and the compact form `navigation.order: 10` work. [Navigation](#) explains the navigation keys in detail.

serve

`mdu serve` renders the site, starts a local preview server, and opens it in your browser. It generates `documentation.pdf` only if a page contains `[Download the PDF] (mdu:pdf)`. While it runs, it watches the documentation folder and the folders containing referenced OpenAPI files: when you save a page, a linked image, or a referenced OpenAPI file, the affected output is refreshed and the browser reloads it. Markdown changes refresh the PDF when a PDF Link is present; adding the first link creates the PDF, and removing the last link removes it. Press `Ctrl+C` to stop.

Usage

```
mduserve[PATH][OUTPATH]][options]
```

See [Paths and file search](#) for the default folders and [common options](#) for `--pattern` and `--hidden`.

Option	Effect
--------	--------

<code>--port <port></code>	Port of the preview server. Default: 4173.
<code>--no-browser</code>	Start the server without opening a browser.
<code>--editPageRoot <url></code>	Add edit links and a repository button, as with <code>mdu render</code> .

Examples

```
mduservermduserve./ProjectDocs./site--port8080mduserve--no-browser
```

Note

The output folder must differ from the documentation folder and must not contain it. Use `mdu render` to create the site you publish; `serve` is meant for local preview.

[How live reload works](#) describes what `serve` renders again after a change.

page

`mdu page` works on the documentation folder as a whole: `page move` moves or renames a page or folder and updates every link to it, and `page links` shows which pages link to a file.

Both commands work in the [default documentation folder](#); choose another one with `--root`. Links in pages outside that folder are not updated.

Move pages and folders

```
mdupagemove<SOURCE><TARGET> [--dry-run] [--root<PATH>]
```

The alias `mdu page mv` runs the same command. `SOURCE` is a Markdown page or a folder. `TARGET` is the new path, or an existing folder to move into; a trailing `/` also moves into a folder and creates it. When a page target has no `.md` extension, MDU adds the extension of the source.

MDU then updates every relative link in the documentation folder:

- Links to the moved page, including links with an anchor or a query such as `Setup.md#install`.
- Links inside the moved page to other pages, images, and files, so they still work from the new place.
- Reference definitions such as `[setup]: ./Setup.md`.
- For a moved folder, links to every file inside it, including images.

A link that still points to the right file is left as it is: pages of a moved folder that link to each other keep their links, and a link to a folder such as `Guide/` stays a folder link. Links in code, links to web addresses, and absolute paths never change. A rename that only changes the case of a name, such as `Setup.md` to `setup.md`, updates the links too, because web servers usually tell case apart.

```
$ mdu page move docs/Setup.md docs/Start/
Moved page: Setup.md -> Start/Setup.md
Updated 5 links in 3 pages:
  Guide/README.md (1)
  README.md (2)
  Start/Setup.md (2)
Run 'mdu lint' to check the new page structure.
```

Add `--dry-run` to see the same report without changing any file.

MDU refuses a move when:

- the target already exists, lies outside the documentation folder, or is inside the moved folder itself;
- the source or target path goes through a symbolic link;
- a symbolic link inside a moved folder would point to a different file afterward, for example a relative `../Shared` in a folder that moves one level deeper, or an absolute link to a file inside the moved folder.

A folder moves together with its symbolic links, and renaming a folder in place keeps their relative targets working. Pages reached through symbolic links are never rewritten.

Tip

A new place can change the navigation: a folder needs a `README.md` page, and the [title of a section page](#) should match its folder name. Run `mdu lint` after a move.

Git recognizes moved pages as renames, so `git log --follow` still shows a page's history. Commit the moved files together with the pages whose links changed.

Find links to a file

```
mdupagelinks<FILE> [--root <PATH>]
```

`page links` lists every link that points to a page, image, or other file, with the page and line of the link. A folder stands for its `README.md`. Use it before you delete or rename a file, or to find the pages that use an image.

```
$ mdu page links docs/Setup.md
Guide/README.md:3: [Setup](../Setup.md)
README.md:3: [Setup](Setup.md#install)
README.md:5: [Reference][setup]
README.md:7: [setup]: ../Setup.md
4 links in 2 pages point to Setup.md.
```

Options

Option	Commands	Effect
<code>--root <PATH></code>	move, links	Documentation folder whose pages are updated or searched. Default: <code>./docs</code> .
<code>--dry-run</code>	move	Show what would change without changing any file.
<code>--hidden</code>	move, links	Include hidden files and folders. Default: on.

pdf

`mdu pdf` combines the Markdown pages of a folder into one PDF file. It uses the same Markdown pipeline as `mdu render`, so code highlighting, icons, keyboard keys, math, callouts, and supported diagrams appear in the PDF too. MDU writes the file itself, without a PDF library, browser, or network service.

For automatic PDF generation when a page contains a PDF Link, and for placing links to it inside rendered pages, see the [PDF Link feature](#).

Usage

```
mdupdf[PATH[OUTPATH]][options]
```

Argument	Meaning	Default
PATH	A documentation folder or one Markdown file	./docs
OUTPATH	The PDF file to write	./documentation.pdf, or the Markdown file's name with .pdf

See [common options](#) for `--pattern` and `--hidden`.

```
mdupdfmdupdf./ProjectDocs./build/manual.pdfmdupdf./ProjectDocs/Features/Math.md./build/math.pdf
```

Layout

- Pages are A4 and follow the order of the documentation navigation.
- A folder export starts with a title page and a linked table of contents; a single file has neither.
- Headings stay with the start of the following block when a page break is needed.
- Links between included pages, links on diagram nodes, and external links stay clickable.

Exclude pages

To leave a page out of a folder export and its table of contents, set `pdf.visible` in its front matter:

```
---pdf.visible: false---
```

The nested form `pdf: { visible: false }` works as well.

Limitations

The PDF contains only what MDU can draw without downloading anything:

- Images appear as their alt text; remote image URLs stay clickable.
- Embedded videos appear as their link.
- Emoji shortcodes stay text, because the built-in PDF fonts have no emoji glyphs.
- The interactive icon search and the animated hero exist only in HTML; the hero's title, text, and button labels appear as plain text.

Publish

`mdu render` writes a folder of static files that any web server can host. Pick the guide for your platform:

- [GitHub](#) - Publish with GitHub Pages and the MDU action
- [GitLab](#) - Publish with GitLab Pages and GitLab CI/CD
- [Custom](#) - Render the site and upload it anywhere

Tip

Fetch the full git history in CI so that pages can show their [last changed date](#).

GitHub

The [mdu GitHub Pages Action](#) renders your documentation and publishes it to GitHub Pages.

1. In your repository, open **Settings ->Pages** and set **Build and deployment ->Source** to **GitHub Actions**.
2. Add this workflow. It runs whenever files under `docs/` change on `main`:

`.github/workflows/publish.yml`

```
name: Publish docs

on:
  push:
    branches: [main]
    paths:
      - 'docs/**'
    permissions:
      contents: read
      pages: write
      id-token: write

jobs:
  publish:
    runs-on: ubuntu-latest
    environment:
      name: github-pages
    steps:
      - uses: capjan/markdown-util-action@v1
```

[Action inputs and details](#)

GitLab

GitLab CI/CD can render your documentation and publish it with GitLab Pages. Add this `.gitlab-ci.yml` to the root of your repository; it publishes when files under `docs/` change on the default branch:

`.gitlab-ci.yml`

```
pages:
  stage: deploy
  image: mcr.microsoft.com/dotnet/sdk:10.0script:
    - dotnet tool install mdu --tool-path .tools
    - .tools/mdu render docs public
  pages:
    publish: public
  rules:
    -if: '$CI PIPELINE SOURCE == "push" && $CI COMMIT BRANCH == $CI DEFAULT BRANCH'changes:
      - docs/**/*
```

GitLab publishes the generated `public` folder; since GitLab 17.10 it is included as a job artifact automatically. Make sure a GitLab Runner is enabled for the project.

Note

If your source folder has another name, update both `docs` and `docs/**/*` to match.

[GitLab Pages documentation](#)

Custom

[Install MDU](#), then render your documentation:

```
mdurender
```

By default, MDU reads the only `docs` folder and writes to `site`. To use different folders, pass the input and output paths after `render`, for example `mdu render ./content ./public/docs`. The generated site contains plain HTML, CSS, and JavaScript and needs no server-side code.

If the sources are on GitHub, pass the edit URL of your docs folder with `--editPageRoot`. Every page then gets an edit link, and the header shows a repository button:

```
mdurender --editPageRoot "https://github.com/example/project/edit/main/docs"
```

Features

Everything on these pages is written in ordinary Markdown. Each page shows the source and the rendered result side by side, so you can copy what you need.

Write

- [Formatting](#) - Headings, emphasis, lists, links, and collapsible sections
- [Code Blocks](#) - Syntax highlighting, titles, and copy buttons
- [Tables](#) - Column alignment and formatting inside cells
- [Callouts](#) - Notes, tips, and warnings with GitHub's alert syntax
- [Footnotes](#) - Numbered notes with hover previews
- [Keyboard Keys](#) - Shortcuts such as `Cmd+S` drawn as keycaps

Illustrate

- [Diagrams](#) - Mermaid, PlantUML, and context diagrams as static SVG
- [Math](#) - GitHub-compatible formulas as static SVG
- [Icons and Emojis](#) - Thousands of icons, colored logos, and emoji shortcodes
- [Images](#) - Local and remote images with a lightbox
- [Media Links](#) - Direct MP4 playback and YouTube and Vimeo players that load only with consent
- [Hero Banner](#) - An animated beach scene for landing pages

Organize

- [Navigation](#) - Sidebar, top tabs, breadcrumbs, ordering, and the page footer

Compose

- [Repository Imports](#) - Combine Markdown pages from multiple GitHub repositories

Document APIs

- [OpenAPI Spec](#) - Add a static API reference from a local OpenAPI specification

Publish

- [PDF Link](#) - Place a link to the generated PDF anywhere in your pages
-

Callouts

Callouts, also called admonitions or alerts, highlight information that readers should not miss. MDU uses the syntax of [GitHub alerts](#), so your files look right on GitHub and in the rendered site.

Syntax

Start a block quote with the alert type on its own line, then write the content on the following lines:

Markdown

```
>[!NOTE]>Useful information that users should know, even when skimming content.
```

MDU supports all five GitHub alert types:

Markdown

```
>[!NOTE]>Useful `information` that users should know, even when skimming content.  
>[!TIP]>`Helpfuladvice` for doing things better or more easily.  
>[!IMPORTANT]>`Kevinformation` users need to know to achieve their goal.  
>[!WARNING]>`Urgentinfo` that needs immediate user attention to avoid problems.  
>[!CAUTION]>Advises about `risks` or `negative` outcomes of certain actions.
```

Examples

This is the Markdown above, rendered. Switch the theme in the header to see the light and dark colors.

Note

Useful information that users should know, even when skimming content.

Tip

Helpful advice for doing things better or more easily.

Important

Key information users need to know to achieve their goal.

Warning

Urgent info that needs immediate user attention to avoid problems.

Caution

Advises about risks or negative outcomes of certain actions.

A callout can contain any Markdown, such as several paragraphs, lists, links, and code:

Tip

Preview your documentation while you write:

```
mduserve
```

The [Usage](#) section describes all commands.

Rules

MDU follows the same rules as GitHub:

- The marker must be the first line of a block quote and stand alone on that line. Text after the marker, such as `> [!NOTE] Text`, renders a regular block quote.
- The type is not case-sensitive: `[!NOTE]`, `[!Note]`, and `[!note]` are equivalent.
- Only `NOTE`, `TIP`, `IMPORTANT`, `WARNING`, and `CAUTION` are alert types. Any other type renders a regular block quote.
- The marker must be followed by content. A block quote that contains only the marker remains a regular block quote.
- Callouts cannot be nested inside lists or other block quotes.

Styling

MDU uses [Markdig's alert blocks](#) and renders the same HTML structure as GitHub: a `div` with the classes `markdown-alert` and `markdown-alert-<type>`, for example `markdown-alert-tip`, followed by a `markdown-alert-title` paragraph. The title is shown without an icon, and the colors adapt to the light and dark themes.

Code Blocks

Fenced code blocks show code or commands exactly as written. A language identifier after the opening fence turns on syntax highlighting, and every block gets a copy button.

Syntax

Add a language identifier after the opening backticks:

Markdown

```
```csharp
var greeting = "Hello, world!";
Console.WriteLine(greeting);
```
```

Add `title="..."` after the language to display a title above the code:

Markdown

```
```istitle="index.is"
console.log('Hello World!')
```
```

GitHub still displays this as a JavaScript code block, but does not render the title because GitHub Markdown has no code block title syntax. MDU displays the title above the code.

MDU renders the titled code block with a header and syntax highlighting:

index.js

```
console.log('Hello World!')
```

Tip

Hover over a code block or focus it with the keyboard to show the copy button. It copies the code without the title.

If you omit the language identifier or use an unsupported one, the code is rendered without syntax highlighting:

```
This is plain code.
```

Supported languages

Use one of these language identifiers:

| Language | Identifiers |
|------------|---------------------|
| C# | cs, csharp |
| JavaScript | js, javascript, jsx |
| Swift | swift |
| TypeScript | ts, typescript |

| | |
|----------|----------------------|
| Python | py, python |
| Kotlin | kt, kotlin |
| JSON | json |
| SQL | sql |
| HTML | html, htm |
| CSS | css |
| Java | java |
| Rust | rs, rust |
| Go | go, golang |
| YAML | yaml, yml |
| Shell | bash, zsh, sh, shell |
| Markdown | md, markdown |

Markdown code blocks highlight headings, block markers, links, inline code, and emphasis while keeping the Markdown source visible as code. The highlighter uses lightweight language rules rather than full language parsers, so it colors selected syntax elements. Other languages remain readable as plain code.

Diagrams

Write diagrams as text; MDU turns supported code blocks into static SVG in the rendered page.

- Rendered at build time, without JavaScript
- Works offline
- Follows the light or dark theme
- Keeps diagram sources portable Markdown

Note

MDU uses its own renderers for subsets of Mermaid and PlantUML. The output can differ from those tools, and unsupported syntax is not rendered as a diagram.

| Type | Use for | Code block | Also renders with |
|-----------|------------------------------|------------|-------------------|
| Flowchart | Processes and decisions | mermaid | Mermaid, GitHub |
| Sequence | Interactions over time | mermaid | Mermaid, GitHub |
| Class | Types and relationships | mermaid | Mermaid, GitHub |
| Component | Architecture building blocks | plantuml | PlantUML |
| Context | A system in its environment | context | MDU only |

Flowcharts

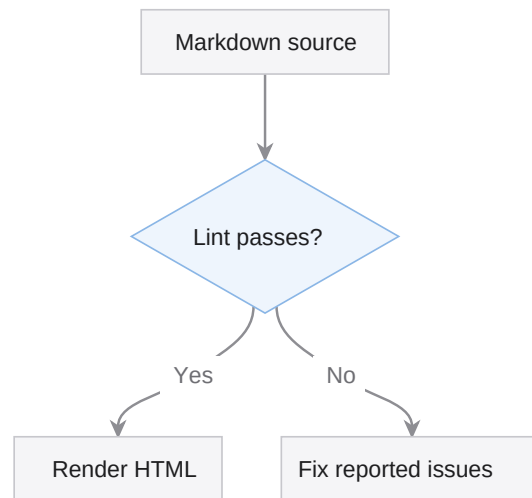
Use a `mermaid` code block for processes and decisions.

Markdown

```

```mermaid
flowchart TD
 accTitle: Documentation workflow
 Source[Markdown source] --> Lint{Lint passes?}
 Lint -->|Yes| Render[Render HTML]
 Lint -->|No| Fix[Fix reported issues]
```

```



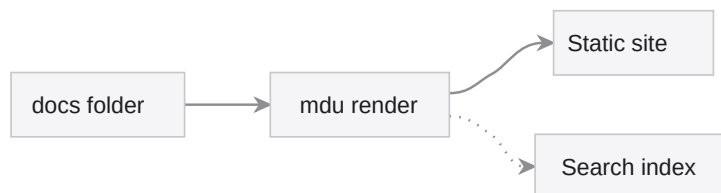
Use LR for a diagram that flows from left to right:

Markdown

```

```mermaid
flowchart LR
 accTitle: Rendered site
 Source[docs folder] --> Render[mdu render] --> Site[Static site]
 Render -. Search index .-> Search[Search index]
```

```



Flowchart syntax

| Element | Syntax | Meaning |
|----------------------|---|---|
| Header and direction | flowchart TD, graph LR | TD/TB top to bottom, BT bottom to top, LR left to right, RL right to left |
| Node shapes | A[Text], A(Text), A([Text]), A((Text)), A{Text}, A[[Text]], A{{Text}} | Rectangle, rounded rectangle, stadium, circle, diamond, subroutine, and hexagon |
| Namespaced IDs | target:App | Adds one namespace prefix to a node ID |
| Labels | A["Text [with brackets]"], A[First
Second] | Double quotes allow brackets in labels;
 starts a new line |
| Solid links | -->, ---, <-->, --0, --x | Arrow, plain line, arrows at both ends, circle end, cross end |
| Dotted links | .->, -.- | Dotted arrow or plain dotted line |
| Thick links | ==>, === | Thick arrow or plain thick line |
| Link labels | A --> Yes B or A -- Yes --> B | Adds explicit text to a link |

| | | |
|-------------------|--|--|
| Chains and groups | A --> B --> C, A & B --> C | Chains links or connects multiple source nodes to one target |
| Subgraphs | subgraph Services ["Services"] ... end | Groups nodes within the flowchart |
| Accessible title | accTitle: Text | Sets the accessible title |
| Comments | %% comment | Ignores the line |

Longer links made by adding dashes, dots, or equals signs span more layout ranks. MDU does not support every Mermaid flowchart shape or statement. Node links are covered under [Clickable nodes](#).

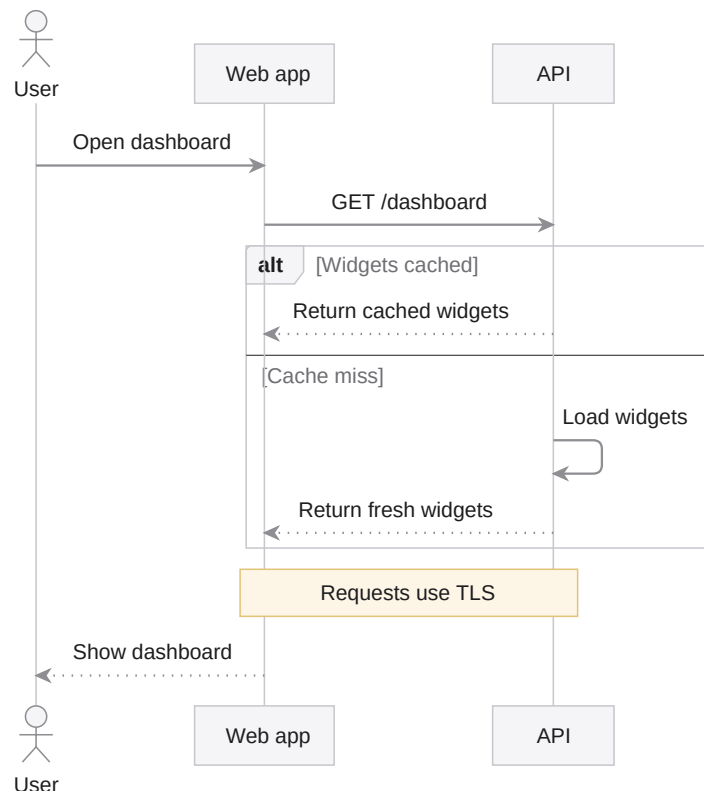
Sequence diagrams

Use a `mermaid` code block to show messages between participants over time.

Markdown

```

mermaid
sequenceDiagram
    accTitle: Dashboard request
    actor User
    participant Web as Web app
    participant API
    User->>Web: Open dashboard
    Web->>API: GET /dashboard
    alt Widgets cached
        API-->>Web: Return cached widgets
    else Cache miss
        API->>API: Load widgets
        API-->>Web: Return fresh widgets
    end
    Note over Web,API: Requests use TLS
    Web-->>User: Show dashboard
  
```



Sequence syntax

| Element | Syntax | Meaning |
|--------------|--|--|
| Participants | participant A, participant A as Label, actor A, actor A as Label | Declares a participant or stick-figure actor, optionally with a display label |
| Messages | ->, -->, ->>, -->>, -x, --x, -), --), <<->>, <<-->> | - draws a solid line and -- a dotted line; >> adds an arrowhead, x a cross end,) an open end, and <<...>> arrowheads at both ends |
| Line breaks | A->>B: First
second, Note over A: First
second | Starts a new line in a message or note |
| Activations | activate A, deactivate A; A->>+B: Text, A->>-B: Text | Starts or ends an activation; + activates the target and - deactivates the source |
| Notes | Note left of A: Text, Note right of A: Text, Note over A,B: Text | Places a note beside one participant or over one or two |
| Blocks | loop, alt/else, opt, par/and, critical/option, break, rect ... end | Groups messages in a control block |
| Numbering | autonumber, autonumber 10 5, autonumber off | Numbers messages, optionally setting the start and step, or turns numbering off |
| Titles | title Text, accTitle: Text | Sets the visible title or accessible title |
| Comments | %% comment | Ignores the line |

Blocks end with `end`. `rect` draws a highlighted area, but its color argument is ignored. `box`, `create`, `destroy`, and participant links are not supported.

Class diagrams

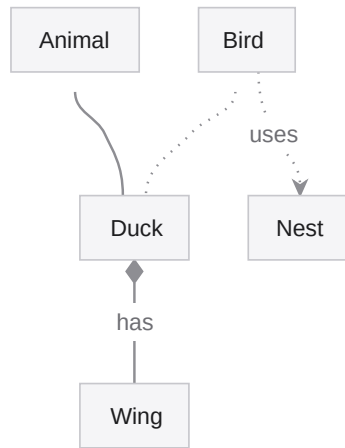
Use `classDiagram` or `classDiagram-v2` in a `mermaid` code block to show types and their relationships.

Markdown

```

```mermaid
classDiagram
class Animal
class Bird
class Duck
class Wing
class Nest
Animal <|-- Duck
Bird <|-- Duck
Duck *-- Wing : has
Bird ..> Nest : uses
```

```



Class syntax

| Element | Syntax | Meaning |
|---------------------|--|---|
| Header | classDiagram, classDiagram-v2 | Starts a class diagram |
| Class | class Name | Declares a class; a relationship also introduces referenced classes |
| Generalization | Base < -- Child, Child -- > Base | Hollow triangle at the superclass, which is placed above the subclass |
| Realization | Interface < .. Implementation, Implementation .. > Interface | Dotted hollow triangle at the interface, which is placed above the implementing class |
| Composition | Whole *-- Part, Part --* Whole | Filled diamond at the whole, which is placed above the part |
| Aggregation | Whole o-- Part, Part --o Whole | Hollow diamond at the whole, which is placed above the part |
| Dependency | A ..> B, B <.. A | Dotted arrow |
| Association | A -- B, A --> B, A <-- B, A .. B | Solid or dotted association, with an optional arrow |
| Relationship labels | A *-- B : has | Shows only explicit labels; MDU adds no default labels |

Class declarations, relationships, and optional relationship labels are supported. Attributes, methods, aliases, and display labels are not. Class diagrams have no title syntax.

Component diagrams

Use a `plantuml` code block for component diagrams. MDU renders a subset of PlantUML component syntax.

Markdown

```

```plantuml
@startuml
title Shop architecture
actor Customer
package "Shop" {
 [Web Shop] <<frontend>> as Web
 component [Order Service] as Orders
}
database "Orders DB" as DB
queue "Events" as Bus

```

```

Customer --> Web : browses
Web --> Orders : REST
Orders --> DB
Orders -right-> Bus : publishes
note right of DB : PostareSQL
@enduml
...

```



## Component syntax

Element	Syntax	Meaning
Frame	@startuml ... @enduml	Optional delimiters for the diagram source
Elements	[Name], component Name, interface Name, actor Name, database Name, node Name, cloud Name, folder Name, frame Name, package Name, rectangle Name, card Name, artifact Name, file Name, queue Name, storage Name	Declares components, interfaces, actors, or other supported elements
Interfaces	() "Name"	Declares an interface with a display name
Aliases	[Name] as Alias, component "Name" as Alias	Assigns an ID to an element with a display name
Containers	package "Shop" { ... }; also node, folder, frame, cloud, database, rectangle, and component	Groups elements; containers may be nested
Stereotypes	<<name>> after an element	Adds a stereotype
Link lines	-, --, .., ==, optionally with > or <	Solid, dotted, or bold line, with an optional arrow end
Link direction and length	->, -->, -up->, -down->, -left->, -right->	One dash defaults right, two or more default down; extra dashes lengthen the link and a direction word selects its direction
Link labels	A --> B : Text	Adds explicit text to a link
Link styles	A -[hidden]-> B, A -[dashed]-> B, A -[bold]-> B	Hides a link or draws it dotted or bold

Triangle ends	A .. > B, A < -- B	Adds a triangle end to the link
Node links	url of ID is [[details.md]]	Links an element; use its alias for ID when declared
Notes	note right of A : Text; note right of A ... end note; note "Text" as N ... end note	Adds an attached or standalone note
Line breaks	\n in labels or notes	Starts a new line
Layout	left to right direction, top to bottom direction	Sets the overall layout direction
Title	title Text	Adds a diagram title and accessible name
Comments	' comment, /' comment /	Ignores line comments and block comments

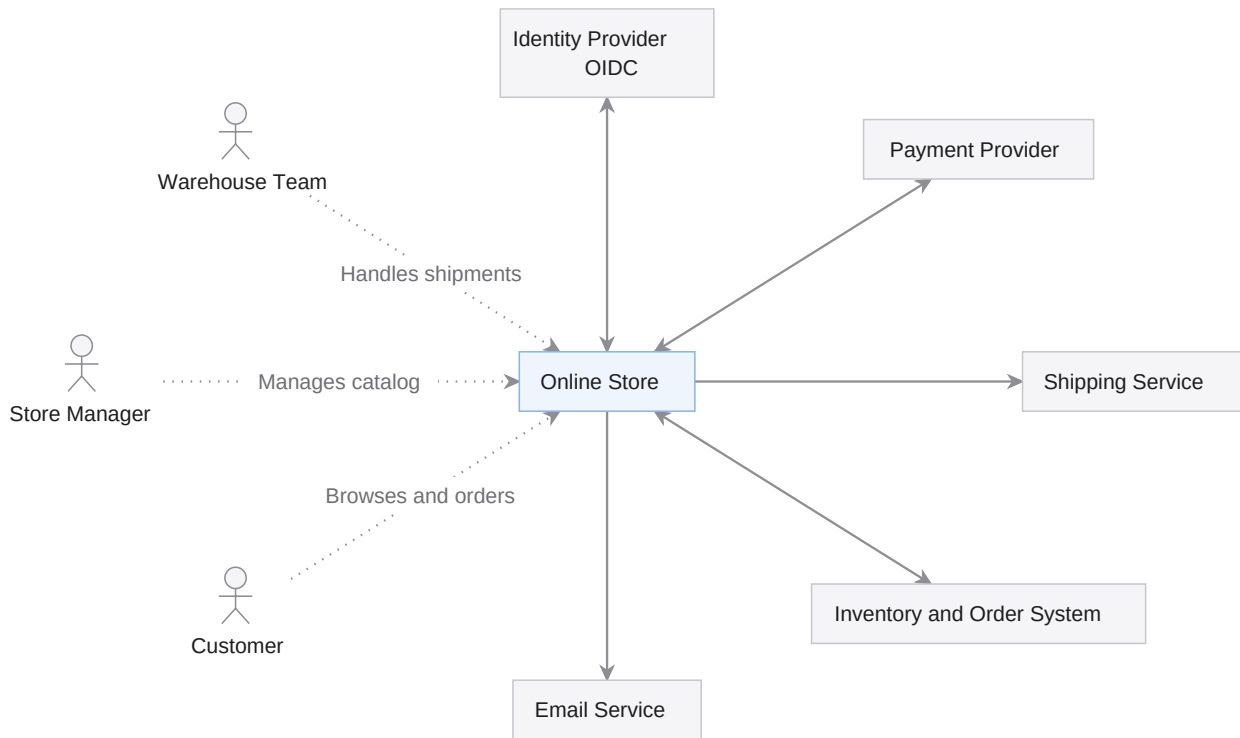
hide, show, scale, !theme, !pragma, allowmixing, and allow\_mixing are ignored. Other PlantUML diagram types, including sequence and class diagrams, remain code blocks.

## Context diagrams

Use an MDU `context` code block to show a system and the people and systems connected to it.

Markdown

```
```context
accTitle: Online store context
Store[Online Store]
Store <--> IdP[Identity Provider<br>OIDC]
Store <--> Payment[Payment Provider]
Store --> Shipping[Shipping Service]
Store <--> Inventory[Inventory and Order System]
Store --> Email[Email Service]
actor Customer
actor Manager as Store Manager
actor Warehouse as Warehouse Team
Customer -. Browses and orders .-> Store
Manager -. Manages catalog .-> Store
Warehouse -. Handles shipments .-> Store
```
```



## Context syntax

| Element  | Syntax                                                                            | Meaning                                              |
|----------|-----------------------------------------------------------------------------------|------------------------------------------------------|
| Header   | None                                                                              | A context block has no header line                   |
| Systems  | Flowchart node shapes such as A[Text], A(Text), A([Text]), A((Text)), and A{Text} | Uses the same shapes as a flowchart                  |
| People   | actor A or actor A as Label                                                       | Adds a stick-figure actor                            |
| Links    | Flowchart links such as -->, <-->, -.->, ==>, and A -- Text --> B                 | Uses the same lines, ends, and labels as a flowchart |
| Titles   | accTitle: System context                                                          | Sets the accessible title                            |
| Comments | %% comment                                                                        | Ignores the line                                     |

The center is the system every link touches. If more than one system qualifies, the first one in the source is the center. Other systems and actors are evenly spaced clockwise from the top in the order they first appear. Reorder their source lines to change their positions. Subgraphs are not supported.

## Clickable nodes

A Mermaid `click` statement links a flowchart or context node; PlantUML's `url of` statement links a component. The linked node is recognizable and can be opened from the rendered HTML.

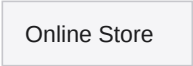
| Diagram   | Syntax                                   | Meaning                           |
|-----------|------------------------------------------|-----------------------------------|
| Flowchart | <code>click A href "details.md"</code>   | Links node A to a destination     |
| Context   | <code>click A href "details.md"</code>   | Links node A to a destination     |
| Component | <code>url of ID is [[details.md]]</code> | Links element ID to a destination |

Relative `.md` destinations become `.html` in the rendered site. `mdurl lint` checks destination casing and that the page exists. External URLs can also be used for node links; the optional Mermaid tooltip is ignored.

## Flowchart

### Markdown

```
```mermaid
flowchart LR
    accTitle: Store link
    Store[Online Store]
    click Store href "store.md"
```
```



```
graph LR
 A[Online Store] -- click --> B["href \"store.md\""]
```

## Context

### Markdown

```
```context
accTitle: Inventory link
Store[Online Store] --> Inventory[Inventory]
click Inventory href "inventory.md"
```
```

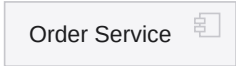


## Component

### Markdown

```
```plantuml
@startuml
title Order service link
component [Order Service] as Orders
url of Orders is ["order-service.md"]
@enduml
```
```

Order service link



```
graph TD
 A[Order Service] -- link --> B["order-service.md"]
```

## Styling and themes

Mermaid `style`, `classDef`, `class`, and `linkStyle` statements, PlantUML `skinparam` and explicit colors are ignored. The page stylesheet sets diagram colors so they follow the light or dark theme.

## Unsupported syntax

### Tip

MDU supports only a subset of Mermaid and PlantUML. If a Mermaid block uses unsupported syntax, it stays a code block and `mdu lint` reports `MDU010`. Unsupported PlantUML and context blocks also

remain code when rendered.

---

## Footnotes

MDU renders labeled footnote references and definitions as automatically numbered footnotes.

## Syntax

The text after `^` is a label that connects a reference to its definition. It is not the visible number: MDU assigns that number automatically. Reusing a label points to the same footnote.

Markdown

```
Tony's repulsor[^repulsor] overheats.
Shuri spots vibranium[^vibranium]. Tony checks the repulsor[^repulsor].

[^repulsor]: Iron Man's energy weapon.
[^vibranium]: Wakanda's energy-absorbing metal.
```

## Rendered example

Tony's repulsor<sup>1</sup> overheats. Shuri spots vibranium<sup>2</sup>. Tony checks the repulsor<sup>1</sup>.

## In the rendered site

- Hover over a reference, or focus it with the keyboard, to preview the footnote. The preview stays open while the pointer is over it.
- Footnotes are listed at the bottom of the page. The footnote you jumped to is emphasized; the others stay in the background.
- A return arrow appears on hover or focus. It jumps back to the text and outlines every reference to that footnote. Click elsewhere or press `Esc` to clear the outline.

- 
1. Iron Man's energy weapon.[Return](#)
  2. Wakanda's energy-absorbing metal.[Return](#)
- 

## Formatting

MDU uses Markdig to render CommonMark syntax and its enabled extensions. Markdown remains readable as plain text in the source file. The examples below show the syntax and the rendered result.

## Headings and paragraphs

Use one to six `#` characters for heading levels one through six. The rendered headings receive an automatic ID; headings from level two on also get a permalink icon that appears on hover or focus, and on

touch screens for the section you jumped to. Separate paragraphs with a blank line.

#### Markdown

```
##Asectionheading

This is one paragraph. A line break in the source
continues the paragraph.

This is a second paragraph.
```

## A section heading

This is one paragraph. A line break in the source continues the paragraph.

This is a second paragraph.

## Line breaks

A single newline in the Markdown source continues the same paragraph. To render a hard line break, end the line with two spaces or a backslash. The two-space form is preferred for readability: it keeps the break unobtrusive in the Markdown source. Press the space bar twice at the end of the line:

#### Markdown

```
This line ends here.
This line starts below it.
```

Rendered result:

This line ends here.  
This line starts below it.

The backslash is an alternative:

#### Markdown

```
This line ends here.\
This line starts below it.
```

Rendered result:

This line ends here.  
This line starts below it.

## Inline formatting

Use asterisks for emphasis and backticks for inline code. Markdig's emphasis extras also support strikethrough, subscript, superscript, and marked text:

#### Markdown

```
italic, **bold**, ***bolditalic***, ~~deleted~~, H~2~O, x^2^, ==marked==, and `inlinecode`.
```

*italic*, **bold**, ***bold italic***, ~~deleted~~, H<sub>2</sub>O, x<sup>2</sup>, marked, and `inline code`.

# Lists

Use `-`, `*`, or `+` for unordered lists, and a number followed by a period for ordered lists. Indent nested items by two spaces:

Markdown

```
- First item
- Second item
- Nested item

1. First step
2. Second step
```

- First item
  - Second item
    - Nested item
1. First step
  2. Second step

Task lists use `[ ]` for an open item and `[x]` for a completed item:

Markdown

```
-[x] Write the page
-[] Review the examples
```

- [x] Write the page
- [ ] Review the examples

# Links

Create a link with a label and a destination. An optional title appears when the reader hovers over the link. Plain URLs such as `https://github.com/xoofx/markdig` are also recognized automatically.

Markdown

```
[Markdig](https://github.com/xoofx/markdig"MarkdigonGitHub")
```

[Markdig](https://github.com/xoofx/markdig)

# Block quotes and separators

Start each quoted line with `>`; block quotes can contain other Markdown formatting, including lists:

Markdown

```
>A block quote can contain *emphasis*.
>>- It can also contain a list.
```

A block quote can contain *emphasis*.

- It can also contain a list.

Use a line containing three hyphens, asterisks, or underscores to add a horizontal separator:

```

```

## Collapsible sections

Wrap content in an HTML `<details>` element to show it only on demand; the `<summary>` is the visible title. Leave blank lines around the Markdown inside so it is rendered as Markdown. Add the `open` attribute to expand the section initially:

Markdown

```
<details>
<summary>Completed projects</summary>

- Project A
- Project B

</details>
```

Completed projects

- Project A
- Project B

## Related features

- [Footnotes](#) - Numbered notes with hover previews
- [Callouts](#) - Notes, tips, and warnings
- [Tables](#) - Pipe tables with alignment

# Hero Banner

## Overview

A hero banner opens a landing page with an animated scene: waves roll in, break into whitewater, and wash up the beach while your title and calls to action sit on top. The scene is rendered live in the browser with WebGL, follows the light and dark theme, and needs no images, fonts, or external scripts.

Hero banner

This banner uses the content width. Switch the theme to see the scene change from day to night.

See the options

## Syntax

Add a fenced code block with the language `hero` and describe the banner with YAML. Place it right below the page heading:

## Markdown

```
#Documentation```hero
title: Plain Markdown, beautifully published
text: Lint, inspect, and render your Markdown files into a fast static documentation site.
actions:
- text: Get started
 link: Setup/README.md
- text: Browse features
 link: ./Features/README.md
```
```

The source stays an ordinary code block, so the file remains readable on GitHub and in any other Markdown viewer.

Options

| Option | Values | Default | Description |
|---------|--------------------------|---------|---|
| title | Text | | Large heading of the banner. |
| text | Text | | Short introduction below the title. |
| actions | List | | Buttons with text, link, and an optional style of primary or secondary. The first button is primary unless you set a style. Links to .md files point to the rendered pages, and links to other sites open in a new tab. |
| time | auto, day, sunset, night | auto | Lighting of the scene. auto shows the day in the light theme and a moonlit night in the dark theme, and fades between them when the reader switches the theme. |
| width | full, content | full | full spans the whole window below the header, content places a rounded banner in the page text. |
| align | left, center | left | Position of the title, text, and buttons. |
| height | 240 to 1200 | 520 | Height in pixels. The banner never takes more than 85 % of the window height. |
| waves | 0 to 2.5 | 1 | Wave height, from a calm sea at 0 to heavy surf. |
| speed | 0 to 4 | 1 | Animation speed. 0 freezes the scene. |
| quality | auto, low, high | auto | Render resolution. auto lowers the resolution on devices that cannot keep up. |
| colors | water, shallow, sand | | Hex colors such as "#0a3d62" that replace the colors of deep water, shallow water, and sand. Quote them, because # starts a YAML comment. |

Example with every option:

Markdown

```
```hero
title: Release 3.0
text: Faster rendering, new diagrams, and a fresh look.
time: sunset
width: full
align: center
height: 600
waves: 1.4
speed: 0.8
quality: auto
actions:
- text: Read the changelog
```

```
link: Changelog.md
- text: Source code
 link: https://github.com/capian/mdu
 style: secondary
colors:
 water: "#0a3d62"
 shallow: "#1fa2a0"
 sand: "#e8d2a6"
```
```

Behavior

- The animation only runs while the banner is visible and the browser tab is active.
- Readers who prefer reduced motion see a still image of the scene.
- Without WebGL, the banner shows a static gradient in the colors of the scene.
- A page shows at most one full-width banner; further `hero` blocks on the same page use the content width.
- The PDF export shows the title, text, and button labels as plain text.

Icons and Emojis

MDU turns shortcodes such as `:rocket:` or `:material-account-circle:` into emojis and icons while it renders your Markdown. Seven open source icon sets and GitHub-compatible emoji shortcodes are built in. You do not need to configure anything, and the rendered pages load nothing from a CDN.

`:rocket:` 🚀

Search

Type part of a name, such as `github`, `arrow right`, or `heart`. Click a result to copy its shortcode. The filters limit the results to icons or emojis.

[The interactive icon search is available in the HTML version.]

Add the same search to your own documentation with an empty `icon-search` code block:

Markdown

```
``icon-search``
```

MDU writes the search script and its index to the `js` folder of the site only when a page contains an icon search.

Usage

Using emojis

The best way to add an emoji is usually to type the emoji itself. It needs no conversion, so your Markdown reads the same on GitHub, in your editor, and in every other tool. Every major operating system has an emoji picker:

| Operating system | Shortcut |
|----------------------------|--------------------------------------|
| macOS | CtrlCtrl+CmdCmd+?Space or ???Globe+E |
| Windows | ?Win+. or ?Win+; |
| Linux (GNOME and GTK apps) | CtrlCtrl+. or CtrlCtrl+; |

If you prefer to keep your files plain ASCII, use a shortcode instead. Emojis use the same shortcodes as GitHub, such as `:smile:`, `:+1:`, `:warning:`, or `:de::`

Markdown

```
Ship it :rocket:
```

Ship it :rocket:

MDU writes the Unicode character into the page, so both ways produce the same result: emojis look like the emojis of the reader's operating system, and GitHub shows them too.

Using icons

Icons use the name of their icon set as a prefix, followed by the icon name. The prefixes are the same as in [Material for MkDocs](#), so existing documentation keeps working.

Markdown

```
The repository is on :fontawesome-brands-github: GitHub.
```

The repository is on  GitHub.

| Icon set | Prefix | Example |
|---------------------------|----------------------|---------------------------------------|
| Material Design Icons | material- | :material-account-circle: |
| Font Awesome Free Brands | fontawesome-brands- | :fontawesome-brands-github: |
| Font Awesome Free Regular | fontawesome-regular- | :fontawesome-regular-face-laugh-wink: |
| Font Awesome Free Solid | fontawesome-solid- | :fontawesome-solid-rocket: |
| Octicons | octicons- | :octicons-heart-fill-24: |
| Simple Icons | simple- | :simple-dotnet: |
| Dashboard Icons | dashboard- | :dashboard-jellyfin: |

For example, `:material-account-circle:` shows the `account-circle` icon from Material Design Icons, and `:octicons-heart-fill-24:` shows the 24-pixel version of the filled Octicons heart.

Icons are embedded into the page as inline SVG. They scale with the font size and take the color of the surrounding text, so they fit headings, links, and the light and dark theme without extra work:

Tip

Icons in callouts take the color of the text around them.

Outside of MDU, for example on GitHub, icon shortcodes are shown as plain text.

Logos in color

Dashboard Icons are logos of apps and self-hosted services. Unlike the other sets, they keep their own colors. When a logo has a light and a dark variant, MDU adds both and shows the one that fits the current theme. Switch the theme in the header to see the GitHub logo change:

Markdown

```
:dashboard-github: GitHub
```

 GitHub

The variants also have their own shortcodes, for example `:dashboard-github-light:`, if a page should always show the same one.

MDU includes the Dashboard Icons whose SVG files are at most 20 KB and contain no embedded images, scripts, or links, which are about nine out of ten.

Where shortcodes work

Shortcodes work anywhere in text, for example in paragraphs, lists, tables, links, and callouts. They are not replaced inside code spans and code blocks, so you can still write about them, as this page does. A shortcode that MDU does not know stays as it is.

Icons before links

Put an icon directly before a link to lead the reader to it. When the pointer is over the link, the icon moves a little toward it:

Markdown

```
:material-arrow-right: [Search](#search)
```

→ [Search](#)

Use a separate line for each entry. Any icon works, not only arrows, for example `:octicons-arrow-right-24:` or `:material-chevron-right:`. The link must be the next element after the icon: text or spaces in between are fine, but a link inside emphasis, such as `**[Styling](#styling)**`, does not move the icon. When the reader's system asks for reduced motion, the icon stays in place.

Styling

Add an attribute list directly after a shortcode, without a space, to give the icon or emoji an ID, classes, or HTML attributes. For example, set a color with the `style` attribute:

Markdown

```
:material-check-circle:{ style="color: #16803c" } Complete
```

✔ Complete

HTML output

An icon is rendered as a `span` with the class `icon` that contains the SVG; the SVG is hidden from screen readers because icons are usually decorative. An emoji is rendered as a `span` with the class `emoji` and the shortcode as its title:

```
<span class="icon"><svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" aria-hidden="true" focusable="false">...</svg></span><span class="emoji" title=":rocket:">??</span>
```

Licenses

The icon sets are published by their authors under these licenses:

- Material Design Icons: [Apache License 2.0](#), by Pictogrammers
- Font Awesome Free: [CC BY 4.0](#), by Fonticons, Inc.
- Octicons: [MIT License](#), by GitHub
- Simple Icons: [CC0 1.0](#), by the Simple Icons contributors
- Dashboard Icons: [Apache License 2.0](#), Copyright (c) 2024 Bjorn Lammers, Meier Lukas, Thomas Camlong and Homarr Labs

MDU converts the icons for inline use: it reduces the Iconify packages to their SVG content and optimizes the Dashboard Icons with [SVGO](#). The `mdu` package contains the license texts and these notices in `THIRD-PARTY-NOTICES.md`.

Simple Icons and Dashboard Icons contain logos of products and companies. The logos remain trademarks of their owners, and the icon licenses do not grant any trademark rights. Use a logo to refer to its product, as documentation usually does, and check the brand's guidelines for other uses.

Images

MDU renders images with standard Markdown syntax. Local image files are copied into the site with their relative paths, and every image opens in a larger view when clicked.

Syntax

Add alternative text in square brackets and the image URL in parentheses. An optional title appears when a visitor hovers over the image:

Markdown

```
![Hand checking off items on a to-do list](https://images.unsplash.com/photo-1768055105012-4c53791b2877)
```

[Bild: Hand checking off items on a to-do list]
(<https://images.unsplash.com/photo-1768055105012-4c53791b2877>)

Photo by [Jakub ?erdzicki](#) on [unsplash](#)

The alternative text describes the image to people who use assistive technology, so write it for every

image.

Local images

Keep local image files inside the documentation folder and link to them with a relative path from the Markdown file. For example, a page can refer to an image in its own `images` subfolder as `images/page.png`.

`mdu render` and `mdu serve` copy linked local images and keep their relative paths. Supported extensions are `.bmp`, `.png`, `.gif`, `.jpg`, `.jpeg`, `.webp`, and `.svg`. `mdu lint` reports a missing image as `MDU004`. External image URLs are not copied; they load from their original location.

In the rendered site

- Images scale down to the page width and keep their proportions, so wide images fit small screens.
- Click an image to open a larger view. Close it with its close button, by clicking the backdrop, or with `Esc`.
- An image that links to another page keeps its link; an image that links to its own file opens the larger view.

Inventory System

The Inventory System tracks sellable quantities across the shop's warehouses. It is the source of truth for available stock and prevents two checkouts from claiming the same units.

Reservations and updates

- The store reads availability while customers browse the catalog.
- The [Order Service](#) creates a short-lived reservation during checkout.
- A successful order converts the reservation into a committed allocation; a cancelled or expired order releases it.
- Warehouse adjustments and completed shipments update the on-hand quantity and publish stock-change events.

The [clickable context example](#) links the Inventory node to this detail page.

Keyboard Keys

Add shortcuts right in a sentence. MDU turns them into keycaps.

Write `++cmd+s++` to show `Cmd+S`. Wrap the whole shortcut in `++` and join keys with `+`.

Save with `Cmd+S`, undo with `Cmd+Z`, and redo with `Cmd+Shift+Z`.

A few simple rules

- Type letters and numbers directly: `++ctrl+c++ ->Ctrl+C`.
- Key names are case-insensitive. Common modifiers include `cmd / command`, `opt / option`, `ctrl / control`, and `shift`.
- Use key names for punctuation: `period` for `.` and `plus` for `+`.
- Put a custom label in quotes: `++cmd+"Any key"++ ->Cmd+Any key`.

Text between `++` that is not a shortcut stays inserted and underlined, as in `++new text++`.

Need another key?

Modifier keys show their familiar symbols: `CmdOptionCtrlShift`. Arrow keys show only their arrows. See the [keyboard key reference](#) for all supported key names and aliases.

Keyboard Key Reference

Use the name in the first column inside `++ . . . ++`. Names are case-insensitive; aliases are listed separately.

Alphanumeric and space keys

| Name | Key | Aliases |
|------|-----|---------|
| 0 | 0 | - |
| 1 | 1 | - |
| 2 | 2 | - |
| 3 | 3 | - |
| 4 | 4 | - |
| 5 | 5 | - |
| 6 | 6 | - |
| 7 | 7 | - |
| 8 | 8 | - |
| 9 | 9 | - |
| a | A | - |
| b | B | - |
| c | C | - |
| d | D | - |
| e | E | - |
| f | F | - |
| g | G | - |
| h | H | - |

| | | |
|-------|--------|-----|
| i | I | - |
| j | J | - |
| k | K | - |
| l | L | - |
| m | M | - |
| n | N | - |
| o | O | - |
| p | P | - |
| q | Q | - |
| r | R | - |
| s | S | - |
| t | T | - |
| u | U | - |
| v | V | - |
| w | W | - |
| x | X | - |
| y | Y | - |
| z | Z | - |
| space | ?Space | spc |

Punctuation

| Name | Key | Aliases |
|---------------|-----|------------------|
| backslash | \ | - |
| bar | | pipe |
| brace-left | { | open-brace |
| brace-right | } | close-brace |
| bracket-left | [| open-bracket |
| bracket-right |] | close-bracket |
| colon | : | - |
| comma | , | - |
| double-quote | " | dblquote |
| equal | = | - |
| exclam | ! | exclamation |
| grave | ` | grave-accent |
| greater | > | greater-than, gt |
| less | < | less-than, lt |
| minus | - | hyphen |
| period | . | - |
| plus | + | - |

| | | |
|--------------|---|---------------|
| question | ? | question-mark |
| semicolon | ; | - |
| single-quote | ' | - |
| slash | / | - |
| tilde | ~ | - |
| underscore | _ | - |

Navigation keys

| Name | Key | Aliases |
|-------------|------------|----------------------|
| arrow-up | UpUp | up |
| arrow-down | DownDown | down |
| arrow-left | <-Left | left |
| arrow-right | ->Right | right |
| page-up | ?Page Up | prior, pg-up |
| page-down | ?Page Down | next, page-dn, pg-dn |
| home | ?Home | - |
| end | ?End | - |
| tab | ?Tab | tabulator |

Editing keys

| Name | Key | Aliases |
|-----------|--------------------|------------|
| backspace | BackspaceBackspace | back, bksp |
| delete | DeleteDel | del |
| insert | Ins | ins |

Action keys

| Name | Key | Aliases |
|--------------|--------------|---------------|
| break | Break | cancel |
| caps-lock | ?Caps Lock | capital, cplk |
| clear | Clear | clr |
| eject | ??Eject | - |
| enter | ?Enter | return |
| escape | ?Esc | esc |
| help | Help | - |
| print-screen | Print Screen | prtsc |
| scroll-lock | Scroll Lock | scroll |

Numeric keypad keys

| Name | Key | Aliases |
|---------------|-----------|--------------------|
| num0 | Num 0 | - |
| num1 | Num 1 | - |
| num2 | Num 2 | - |
| num3 | Num 3 | - |
| num4 | Num 4 | - |
| num5 | Num 5 | - |
| num6 | Num 6 | - |
| num7 | Num 7 | - |
| num8 | Num 8 | - |
| num9 | Num 9 | - |
| num-asterisk | Num * | multiply |
| num-clear | Num Clear | - |
| num-delete | Num Del | num-del |
| num-equal | Num = | - |
| num-lock | Num Lock | numlk, numlock |
| num-minus | Num - | subtract |
| num-plus | Num + | add |
| num-separator | Num . | decimal, separator |
| num-slash | Num / | divide |
| num-enter | Num Enter | - |

Modifier keys

| Name | Key | Aliases |
|---------------|------------------|-----------------------------|
| alt | Alt | - |
| left-alt | Left Alt | lalt |
| right-alt | Right Alt | ralt |
| alt-graph | AltGr | altgr |
| command | CmdCmd | cmd |
| left-command | CmdLeft Command | lcommand, lcmd, left-cmd |
| right-command | CmdRight Command | rcommand, rcmd, right-cmd |
| control | CtrlCtrl | ctrl |
| left-control | CtrlLeft Ctrl | lcontrol, lctrl, left-ctrl |
| right-control | CtrlRight Ctrl | rcontrol, rctrl, right-ctrl |
| function | Fn | fn |
| globe | ???Globe | - |
| meta | Meta | - |

| | | |
|---------------|------------------|---------------------------|
| left-meta | Left Meta | lmeta |
| right-meta | Right Meta | rmeta |
| option | OptOption | opt |
| left-option | OptLeft Option | loption, lopt, left-opt |
| right-option | OptRight Option | roption, ropt, right-opt |
| shift | ShiftShift | - |
| left-shift | ShiftLeft Shift | lshift |
| right-shift | ShiftRight Shift | rshift |
| super | Super | - |
| left-super | Left Super | lsuper |
| right-super | Right Super | rsuper |
| windows | ?Win | win |
| left-windows | ?Left Win | lwindows, left-win, lwin |
| right-windows | ?Right Win | rwindows, right-win, rwin |

Function keys

| Name | Key | Aliases |
|------|-----|---------|
| f1 | F1 | - |
| f2 | F2 | - |
| f3 | F3 | - |
| f4 | F4 | - |
| f5 | F5 | - |
| f6 | F6 | - |
| f7 | F7 | - |
| f8 | F8 | - |
| f9 | F9 | - |
| f10 | F10 | - |
| f11 | F11 | - |
| f12 | F12 | - |
| f13 | F13 | - |
| f14 | F14 | - |
| f15 | F15 | - |
| f16 | F16 | - |
| f17 | F17 | - |
| f18 | F18 | - |
| f19 | F19 | - |
| f20 | F20 | - |
| f21 | F21 | - |
| f22 | F22 | - |

| | | |
|-----|-----|---|
| f23 | F23 | - |
| f24 | F24 | - |

Extra keys

| Name | Key | Aliases |
|-------------------|-------------------|--------------|
| backtab | ?Back Tab | bktab |
| browser-back | Browser Back | - |
| browser-favorites | Browser Favorites | favorites |
| browser-forward | Browser Forward | forward |
| browser-home | Browser Home | - |
| browser-refresh | Browser Refresh | refresh |
| browser-search | Browser Search | search |
| browser-stop | Browser Stop | - |
| copy | Copy | - |
| context-menu | Menu | apps, menu |
| fingerprint | Fingerprint | - |
| mail | Mail | launch-mail |
| media | Media | launch-media |
| media-next-track | ??Next Track | next-track |
| media-pause | ??Pause | pause |
| media-play | ??Play | play |
| media-play-pause | ??Play/Pause | play-pause |
| media-prev-track | ??Previous Track | prev-track |
| media-stop | ??Stop | stop |
| power | ?Power | - |
| print | Print | - |
| reset | Reset | - |
| select | Select | - |
| sleep | Sleep | - |
| volume-down | Volume Down | vol-down |
| volume-mute | Mute | mute |
| volume-up | Volume Up | vol-up |
| zoom | Zoom | - |

Mouse keys

| Name | Key | Aliases |
|---------------|---------------|---------|
| left-button | Left Button | lbutton |
| middle-button | Middle Button | mbutton |
| right-button | Right Button | rbutton |

x-button1

X Button 1

xbutton1

x-button2

X Button 2

xbutton2

[Back to Keyboard Keys](#)

Math

MDU renders GitHub-compatible math syntax to static SVG while it builds the site. The pages contain the formulas directly, so browsers need no JavaScript math renderer or font service.

Inline math

Wrap an expression in single dollar signs:

Markdown

```
The area of a circle is $A = \pi r^2$.
```

The area of a circle is $A = \pi r^2$.

Display math

Use double dollar signs for a centered formula:

Markdown

```
$$  
\int_0^1 x^2 dx = \frac{1}{3}  
$$
```

$$\int_0^1 x^2 dx = \frac{1}{3}$$

Fenced math

GitHub's `math` code fence renders a display formula as well:

Markdown

```
```\math  
\sum_{i=1}^n i
```
```

$$\sum_{i=1}^n i$$

Compatibility

- MDU supports the three forms shown above: `$...$`, `$$...$$`, and fenced `math` blocks. GitHub's additional inline delimiter form is not supported.

- The .NET LaTeX renderer implements a subset of LaTeX math commands, not a complete LaTeX engine. If an expression cannot be rendered, the page shows its TeX source.
- Formulas become inline SVG paths with theme-aware colors. The renderer's licenses are listed in the packaged `THIRD-PARTY-NOTICES.md`.

References

- [GitHub: Writing mathematical expressions](#) explains GitHub's Markdown delimiters.
- [Overleaf: Mathematical expressions](#) is a reference for common LaTeX math syntax and symbols.

Media Links

Use an HTML `<video>` element to embed an MP4 directly. Put a YouTube or Vimeo link on a line of its own to show it in a responsive player; a link inside a sentence stays an ordinary link.

Direct MP4 video

This eight-second demo scene is attached to the [v4.6.1 GitHub release](#). The Pages workflow copies it into the documentation site, so the video renders from the central docs site without being stored in the Git repository.

For PDF output, put the opening `<video>` tag on its own line, as in the example below. A one-line `<video ...></video>` is treated as inline HTML and omitted from the PDF.

HTML

```
<videocontrolsplaysinlinepreload="none"width="960"height="540"poster="https://capiian.github.io/
mdu/assets/mdu-demoscene-preview.png"><sourcesrc="https://capiian.github.io/mdu/assets/mdu-demos
cene.mp4"type="video/mp4">
  Your browser does not support embedded videos.
</video>
```

[\[Watch video\]](#)

YouTube and Vimeo

YouTube

Markdown

```
[WatchthefirstversionoftenfingersonYouTube](https://youtu.be/hyeCMRQhVOI)
```

<https://www.youtube-nocookie.com/embed/hyeCMRQhVOI>

Vimeo

Markdown

```
[WatchtheVimeodemo](https://vimeo.com/76979871)
```

<https://player.vimeo.com/video/76979871?dnt=1>

Privacy by default

An embedded player contacts its provider as soon as it loads, even if nobody presses play. MDU therefore shows a placeholder first and loads the video only when the reader agrees. Depending on the selected player, the buttons read:

- **Load this video once from YouTube** or **Load this video once from Vimeo** loads only that video on this page.
- **Always allow YouTube on this site** or **Always allow Vimeo on this site** saves the choice in the browser's local storage, so later visits load that provider's videos directly.

The reader can review or revoke saved choices at any time under **Media settings** in the page header. Revoking removes the saved choice and unloads that provider's players on the current page.

Note

Once a video is loaded, the provider may process information about the visit, and revoking cannot undo what it already received. The placeholder links to the provider's privacy policy, and Vimeo players use the `dn t=1` option to limit new cookies.

Navigation

MDU builds the navigation from the folder structure. A folder's `README.md` represents the folder, and the other pages appear beneath it. You control the result with front matter:

| Key | Effect |
|-----------------------------------|---|
| <code>navigation.title</code> | Label in the sidebar, tabs, and breadcrumbs |
| <code>navigation.order</code> | Sort order among siblings |
| <code>navigation.visible</code> | <code>false</code> hides the entry |
| <code>navigation.expanded</code> | <code>true</code> opens a sidebar section initially |
| <code>navigation.placement</code> | <code>top</code> turns a first-level entry into a tab |

Breadcrumbs above each page show its path from the documentation root. Pages whose trail would only name the page itself, such as the landing page, show none.

Top navigation

Set `navigation.placement: top` on a page at the documentation root, or on the `README.md` or `index.md` of a first-level folder, to make it a tab:

```
docs/  
??? README.md  
??? Development/  
?   ??? README.md # navigation.placement: top  
?   ??? Setup.md  
??? Operations.md # navigation.placement: top
```

`docs/Development/README.md`

```
---navigation:
  placement: top
  title: Development
  order: 10---
```

How tabs behave:

- **Wide screens** show the tabs below the site header. The sidebar lists only the active tab's pages and is hidden when the tab has none.
- **Up to 1220 px** wide, the tabs fold into the full sidebar tree at their original place, and the selected section opens there. On narrow screens the sidebar becomes a drawer.
- **Breadcrumbs** always keep the full path from the documentation root; selecting a tab does not change it.
- The root `README.md` stays the landing page. Mark it `top` as well if it should also appear as a tab, such as *Dashboard*.

A top-level Markdown file can be a tab too; its sidebar section is simply empty.

Ordering entries

Smaller `navigation.order` values come first among siblings. Entries without an order follow and sort alphabetically by their navigation title; the same applies to tabs.

```
---navigation:
  order: 10---
```

The compact form `navigation.order: 10` works too. If both forms are present, the nested `navigation` value wins. `navigation.title` changes only the navigation label, not the page's title or heading.

Expanding a section

Set `navigation.expanded: true` on a page with child pages to open its sidebar section by default:

```
---navigation:
  expanded: true---
```

The section leading to the current page always opens, and sections the reader opens or closes stay that way while they browse. A selected top navigation tab is always expanded, so `navigation.expanded` on it is a lint error.

Hiding an entry

Set `navigation.visible: false` to omit an entry from the sidebar and tabs. The page stays reachable by its URL, and the child pages of a hidden folder page move up one level:

```
---navigation:
  visible: false---
```

Note

`mdu lint` rejects the retired `sidebar.*` front matter. Rename it to `navigation.*`; `mdu fix` does not migrate it.

Page footer

Each page ends with links to the previous and next page in sidebar order, so readers can work through the documentation from start to finish. Hidden pages have no such links and are skipped by their neighbors.

The footer also shows when the page last changed, taken from the last git commit that touched its Markdown file. Uncommitted changes do not count, and files that were never committed show no date. With `--editPageRoot`, an edit icon appears beside the level-one heading, or an **Edit this page** link in the footer when there is no heading.

Important

The date needs the full git history. In a shallow clone, which CI services create by default, MDU leaves the date out instead of showing a wrong one.

Fetch the full history in GitHub Actions with `fetch-depth: 0`:

```
-uses: actions/checkout@v7
  with:
    fetch-depth: 0
```

In GitLab CI, set the `GIT_DEPTH` variable to `0` for the job that renders the site.

Online Store

The online store is the customer-facing application for browsing the catalog and placing orders. It owns product descriptions, search, the shopping basket, and the customer experience. Order processing and inventory are handled by separate services.

Main interactions

- Customers browse products and submit a basket from the web shop.
- The store sends checkout requests to the [Order Service](#).
- Product availability is shown using data from the [Inventory System](#).
- Identity, payment, shipping, and email are provided by external systems shown in the [context diagram](#).

The [clickable flowchart example](#) links the Store node to this detail page.

OpenAPI Spec

MDU turns a local OpenAPI specification into a static API reference inside a Markdown page. The specification stays in its JSON or YAML file; MDU adds the generated reference when it renders the page.

For a complete short example with the generated output, see [OpenAPI Spec example](#). The public Swagger Petstore specification shows a larger reference in both formats: the [OpenAPI 3.0 example](#) and the [Swagger 2.0 example](#).

Add an API reference

Set `openapi` in the page's front matter. Its path is relative to the Markdown file:

```
---title: Orders API
description: Reference for the Orders service.
openapi: ./orders.yaml
---#OrdersAPI
```

This introduction appears before the generated reference.

`mdu render`, `mdu serve`, and `mdu pdf` support `.json`, `.yaml`, and `.yml` specifications in the OpenAPI 3 and Swagger 2.0 formats. The reference follows the page's Markdown content. `mdu serve` also watches the folders containing referenced specifications and refreshes the page when a specification changes.

Generated reference

Operations are grouped by their first tag in a compact list. Tag groups and endpoints start collapsed; expand a tag to browse its endpoints, then expand an endpoint to inspect its method and path, parameters, request body, and responses. Each endpoint row shows the operation summary next to its path. Deprecated operations carry a **Deprecated** badge, a struck-through path, and a faded method stripe, and an expanded deprecated operation starts with a notice; a lock marks operations that require authorization; hover the lock to see the required security schemes. The method counts beside a tag name the HTTP method on hover and say how many of those endpoints are deprecated. When the API has deprecated operations, a **Show deprecated** switch next to the search field hides or shows them. The search field above the tags filters endpoint paths without regard to case. Nonmatching endpoints and empty tags disappear, and the method counts show only the matching endpoints; clearing the search restores the full list, the counts, and the tags' previous expanded state.

The overview lists the API's license, contact, and terms of service, its servers, and its security schemes. An expanded endpoint names the schemes it requires, with OAuth scopes, and repeats no description that only restates the summary. A required request body carries a **Required** badge.

The response and media type selectors open a list that works with the mouse and the keyboard: arrow keys move, Enter or Space chooses, Escape closes, and typing jumps to a matching entry. The response selector shows the status code and name, followed by response headers when the response declares any, and by a media type selector when the response has content. Responses and request bodies provide Example, Schema, and Schema JSON tabs. Schema descriptions appear below their names, and referenced schemas are linked and displayed inline. Response and media type selectors show their position as 1 of N. 2xx statuses are green, 4xx are yellow, and 5xx are red.

Parameter and schema tables combine names and types as `name: Type`. Primitive types with a format show the format instead, such as `uuid`, `int64`, or `dateTime`. Maps from `additionalProperties` read like arrays, for example `[string: Type]`. Required fields carry a red asterisk, and default values follow the type as `= value`. Deprecated fields carry a **Deprecated** badge. Below a field, MDU lists its access mode and constraints, such as `read-only`, `length 3-64`, `? 1`, or a pattern, and the example of a parameter. The allowed values of an enum appear below the field that uses it, so enums need no separate schema table. Table descriptions start hidden; use **Show descriptions** in the table header to reveal them. Tables without description text omit this control.

Generated examples use typical values for common string formats, such as a UUID or a UTC timestamp, when the specification provides no example. Generated numbers respect declared minimums, request examples leave out read-only properties, and response examples leave out write-only properties.

The generated reference gives each HTTP method a distinct label, highlights endpoint paths, and separates API groups and schemas. It follows the site's light or dark theme and is published as static HTML and CSS without making API requests. References within the specification are resolved; external `$ref` files are not loaded.

OpenAPI Spec example

This page shows the Markdown front matter and a short OpenAPI document. MDU generates the reference below from the same YAML file.

The Markdown page uses this front matter:

```
---title: Orders API
openapi: ./openapi-example.yaml
---
```

The referenced `openapi-example.yaml` contains one endpoint and its response schema:

```
openapi: 3.1.0info:
  title: Orders API
  version: 1.0.0paths:
  /orders/{id}:
    get:
      tags: [Orders]summary: Get an order
      parameters:
        -name: id
          in: path
          required: true
      schema: { type: string }responses:
      '200':
        description: Found
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Order'components:
schemas:
  Order:
    type: object
    required: [id, status]properties:
      id: { type: string }status:
        type: string
        enum: [pending, shipped]
```

Generated reference

API overview

Orders API

Version 1.0.0

Orders

GET /orders/{id}

Get an order

Path Parameters

| Parameter | Description |
|------------|-------------|
| id: string | |

Response 200 - OK

Found

Media type application/json

Example

Order

```
{"id": "string", "status": "pending"}
```

Schema

Order

| Property | Description |
|------------------------------|-------------|
| id: string | |
| status: stringpendingshipped | |

Schema JSON

Order

```
{"required":["id","status"],"type":"object","properties":{"id":{"type":"string"},"status":{"enum":["pending","shipped"],"type":"string"}}
```

Order Service

The Order Service turns a customer's checkout request into a durable order. It validates prices and customer details, reserves stock, starts payment, and publishes order events for fulfillment.

Order flow

1. The service checks the basket and creates an order with a stable request identifier so a retry cannot create a duplicate.
2. It asks the [Inventory System](#) to reserve the requested quantities.
3. It requests payment authorization and stores the resulting status with the order.
4. It publishes an order-confirmed event after payment succeeds; fulfillment and notification services

consume that event.

If a reservation or payment fails, the order remains visible with a failure status and any successful reservation is released. The [clickable component example](#) links its Order Service node to this page.

PDF Link

`mdu render` and `mdu serve` create a combined `documentation.pdf` in the site output folder only when at least one Markdown page contains a PDF Link. With no PDF Link, no PDF asset is generated. `mdu serve` refreshes the PDF when Markdown pages change.

Syntax

Place a link anywhere in your Markdown and choose its label:

Markdown

```
[DownloadDocumentationasPDF](mdu:pdf)
```

MDU resolves the link to the generated PDF, including on pages in nested folders. It renders with the site's normal link styling.

Rendered example

[Download Documentation as PDF](#)

The PDF follows the documentation navigation order and honors [pdf.visible](#) in page front matter.

The PDF uses the same Markdown rendering pipeline as [mdu_pdf](#), including syntax highlighting, math, callouts, and supported diagrams. See [PDF output](#) for the layout and its limitations.

Repository Imports

MDU can combine Markdown pages from several GitHub repositories into one documentation site. Add an import declaration to the front matter of a page where the imported section should appear:

docs/API/README.md

```
---title: API reference
mdu:
  import:
    source: https://github.com/example/api-docs/tree/main/Docs
  navigation:
    order: 20---
```

Copy the GitHub `tree` URL for the folder into `source`. That URL identifies the repository, branch or tag, and folder. Branch names containing `/` are supported.

MDU imports the folder's page tree at the declaration's position. The imported folder must contain a

`README.md` or `index.md` at its root. The declaration page's front matter controls the root page metadata; keep the declaration page's body empty and put its content in the imported root page instead. Front matter on imported child pages is preserved. Imports declared on local child pages are expanded recursively; imports declared inside the imported repository are rejected.

Edit links on imported pages point to their source files in GitHub and are derived from the same `tree` URL. Repository imports are resolved by `mdx-render`, `mdx-serve`, `mdx-tree`, `mdx-lint`, and `mdx-pdf`.

Git must be installed. Public repositories work without credentials; private repositories use the authentication already configured for Git.

Swagger Petstore example

This page renders the [Swagger Petstore](#) sample API, a larger reference than the [short OpenAPI Spec example](#). The specification uses the Swagger 2.0 format and contains pets, store orders, and users with path, query, header, and form parameters, enums, and file uploads. The [OpenAPI 3.0 Petstore example](#) shows the same API in the newer format.

The page needs only this front matter:

```
---openapi: ./openapi-petstore.json
---
```

`openapi-petstore.json` is the specification from <https://petstore.swagger.io/v2/swagger.json>, formatted with indentation. The specification declares the [Apache License 2.0](#).

API overview

Swagger Petstore

Version `1.0.7`

License [Apache 2.0](#) · Contact apiteam@swagger.io · [Terms of service](#)

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on <irc.freenode.net>, [#swagger](#). For this sample, you can use the api key `special-key` to test the authorization filters.

Servers

- `https://petstore.swagger.io/v2`
- `http://petstore.swagger.io/v2`

Authentication

- `api_key` API key `api_key` in header
- `petstore_auth` OAuth 2.0

pet

Everything about your Pets

GET /pet/findByStatus ■

Finds Pets by status

Operation ID `findPetsByStatus`

Multiple status values can be provided with comma separated strings



Authorization

- `petstore_auth OAuth 2.0 write:pets read:pets`

Query Parameters

| Parameter | Description |
|---------------------------------------|---|
| status: [string]availablepending sold | Status values that need to be considered for filter |

Response 200 - OK

successful operation

Media type `application/json`

Example

[Pet]

```
[{"id":0,"category":{"id":0,"name":"string"},"name":"doggie","photoUrls":["string"],"tags":[{"id":0,"name":"string"}],"status":"available"}]
```

Schema

[Pet]

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| category: Category | |
| name: string | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablepending sold | pet status in the store |

Category

| Property | Description |
|----------|-------------|
|----------|-------------|

id: int64

name: string

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

[Pet]

```
{"type": "array", "items": {"$ref": "#/components/schemas/Pet"}}
```

Pet

```
{"required": ["name", "photoUrls"], "type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "category": {"$ref": "#/components/schemas/Category"}, "name": {"type": "string", "example": "doggie"}, "photoUrls": {"type": "array", "items": {"type": "string", "xml": {"name": "photoUrl"}}, "xml": {"wrapped": true}}, "tags": {"type": "array", "items": {"$ref": "#/components/schemas/Tag"}, "xml": {"wrapped": true}}, "status": {"enum": ["available", "pending", "sold"], "type": "string", "description": "pet status in the store"}}, "xml": {"name": "Pet"}}
```

Category

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string"}}, "xml": {"name": "Category"}}
```

Tag

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string"}}, "xml": {"name": "Tag"}}
```

Media type application/xml

Example

[Pet]

```
[  
  {  
    "id": 0,  
    "category": {  
      "id": 0,  
      "name": "string"  
    },  
    "name": "doggie",  
    "photoUrls": [  
      "string"  
    ],  
    "tags": [  
      {  
        "id": 0,  
        "name": "string"  
      }  
    ],  
    "status": "available"  
  }  
]
```

Schema

[Pet]

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| category: Category | |
| name: string | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

[Pet]

```
{"type": "array", "items": {"$ref": "#/components/schemas/Pet"}}
```

Pet

```
{"required": ["name", "photoUrls"], "type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "category": {"$ref": "#/components/schemas/Category"}, "name": {"type": "string", "example": "doggie"}, "photoUrls": {"type": "array", "items": {"type": "string", "xml": {"name": "photoUrl"}}, "xml": {"wrapped": true}}, "tags": {"type": "array", "items": {"$ref": "#/components/schemas/Tag"}, "xml": {"wrapped": true}}, "status": {"enum": ["available", "pending", "sold"], "type": "string", "description": "pet status in the store"}}, "xml": {"name": "Pet"}}
```

Category

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string"}}, "xml": {"name": "Category"}}
```

Tag

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string"}}, "xml": {"name": "Tag"}}
```

Response 400 - Bad Request

Invalid status value

Media type `application/json`

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type `application/xml`

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

GET /pet/findByTagsDeprecated 🚧

Finds Pets by tags

This operation is deprecated and should no longer be used.

Operation ID `findPetsByTags`

Multiple tags can be provided with comma separated strings. Use tag1, tag2, tag3 for testing.



Authorization

- `petstore_oauth2.0write:petsread:pets`

Query Parameters

| Parameter | Description |
|----------------|-------------------|
| tags: [string] | Tags to filter by |

Response 200 - OK

successful operation

Media type application/json

Example

[Pet]

```
{ "id": 0, "category": { "id": 0, "name": "string", "name": "doggie", "photoUrls": [ "string" ], "tags": [ { "id": 0, "name": "string" } ], "status": "available" }
```

Schema

[Pet]

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| category: Category | |
| name: string | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

[Pet]

```
{ "type": "array", "items": { "$ref": "#/components/schemas/Pet" } }
```

Pet

```
{ "required": [ "name", "photoUrls" ], "type": "object", "properties": { "id": { "type": "integer", "format": "int64" }, "category": { "$ref": "#/components/schemas/Category" }, "name": { "type": "string", "example": "doggie" }, "photoUrls": { "type": "array", "items": { "type": "string", "xml": { "name": "photoUrl" } }, "xml": { "wrapped": true } }, "tags": { "type": "array", "items": { "$ref": "#/components/schemas/Tag" }, "xml": { "wrapped": true } }, "status": { "enum": [ "available", "pending", "sold" ], "type": "string", "description": "pet status in the store" }, "xml": { "name": "Pet" } }
```

Category

```
{ "type": "object", "properties": { "id": { "type": "integer", "format": "int64" }, "name": { "type": "string" }, "xml": { "name": "Category" } }
```

Tag

```
{ "type": "object", "properties": { "id": { "type": "integer", "format": "int64" }, "name": { "type": "string" } }, "xml": { "name": "Tag" } }
```

Media type application/xml

Example

[Pet]

```
[ {
  {
    "id": 0,
    "category": {
      "id": 0,
      "name": "string"
    },
    "name": "doggie",
    "photoUrls": [
      "string"
    ],
    "tags": [
      {
        "id": 0,
        "name": "string"
      }
    ],
    "status": "available"
  }
}]
```

Schema

[Pet]

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| category: Category | |
| name: string | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

[Pet]

```
{"type": "array", "items": {"$ref": "#/components/schemas/Pet"}}
```

Pet

```
{"required": ["name", "photoUrls"], "type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "category": {"$ref": "#/components/schemas/Category"}, "name": {"type": "string", "example": "doggie"}, "photoUrls": {"type": "array", "items": {"type": "string", "xml": {"name": "photoUrl"}}, "xml": {"wrapped": true}}, "tags": {"type": "array", "items": {"$ref": "#/components/schemas/Tag"}, "xml": {"wrapped": true}}, "status": {"enum": ["available", "pending", "sold"], "type": "string", "description": "pet status in the store"}, "xml": {"name": "Pet"}}
```

Category

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string"}}, "xml": {"name": "Category"}}
```

Tag

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string"}}, "xml": {"name": "Tag"}}
```

Response 400 - Bad Request

Invalid tag value

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

GET /pet/{petId} ■

Find pet by ID

Operation ID `getPetById`

Returns a single pet



Authorization

- `api_key` API key `api_key` in header

Path Parameters

| Parameter | Description |
|----------------------------|---------------------|
| <code>petId</code> : int64 | ID of pet to return |

Response 200 - OK

successful operation

Media type `application/json`

Example

Pet

```
{ "id": 0, "category": { "id": 0, "name": "string" }, "name": "doggie", "photoUrls": [ "string" ], "tags": [ { "id": 0, "name": "string" } ], "status": "available" }
```

Schema

Pet

| Property | Description |
|---|-------------------------|
| <code>id</code> : int64 | |
| <code>category</code> : Category | |
| <code>name</code> : string | |
| <code>photoUrls</code> : [string] | |
| <code>tags</code> : [Tag] | |
| <code>status</code> : stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|----------------------------|-------------|
| <code>id</code> : int64 | |
| <code>name</code> : string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "category": {
      "$ref": "#/components/schemas/Category",
      "name": {
        "type": "string",
        "example": "doggie"
      },
      "photoUrls": {
        "type": "array",
        "items": {
          "type": "string",
          "xml": {
            "name": "photoUrl"
          },
          "xml": {
            "wrapped": true
          }
        },
        "tags": {
          "type": "array",
          "items": {
            "$ref": "#/components/schemas/Tag",
            "xml": {
              "wrapped": true
            }
          },
          "status": {
            "enum": ["available", "pending", "sold"],
            "type": "string",
            "description": "pet status in the store"
          },
          "xml": {
            "name": "Pet"
          }
        }
      }
    }
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    },
    "xml": {
      "name": "Category"
    }
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    },
    "xml": {
      "name": "Tag"
    }
  }
}
```

Media type application/xml

Example

Pet

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Schema

Pet

| Property | Description |
|--------------------|-------------|
| id: int64 | |
| category: Category | |

name: string

photoUrls: [string]

tags: [Tag]

status: stringavailablependingssold

pet status in the store

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{"required":["name","photoUrls"],"type":"object","properties":{"id":{"type":"integer","format":"int64"},"category":{"$ref":"#/components/schemas/Category"},"name":{"type":"string","example":"doggie"},"photoUrls":{"type":"array","items":{"type":"string","xml":{"name":"photoUrl"},"xml":{"wrapped":true}}},"tags":{"type":"array","items":{"$ref":"#/components/schemas/Tag"},"xml":{"wrapped":true},"status":{"enum":["available","pending","sold"],"type":"string","description":"pet status in the store"}}, "xml":{"name":"Pet"}}
```

Category

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64"},"name":{"type":"string"}}, "xml":{"name":"Category"}}
```

Tag

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64"},"name":{"type":"string"}}, "xml":{"name":"Tag"}}
```

Response 400 - Bad Request

Invalid ID supplied

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Response 404 - Not Found

Pet not found

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

POST/pet 🐾

Add a new pet to the store

Operation ID addPet



Authorization

- petstore_authOAuth 2.0write:petsread:pets

Request body Required

Pet object that needs to be added to the store

Media type application/json

Example

Pet

```
{ "id": 0, "category": { "id": 0, "name": "string" }, "name": "doggie", "photoUrls": [ "string" ], "tags": [ { "id": 0, "name": "string" } ], "status": "available" }
```

Schema

Pet

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| category: Category | |
| name: string | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl",
          "wrapped": true
        }
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "Pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "Category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "Tag"
  }
}
```

Media type application/xml

Example

Pet

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Schema

Pet

| Property | Description |
|------------------------------------|-------------------------|
| id: int64 | |
| category: Category | |
| name: string | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingsold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{ "required": [ "name", "photoUrls", "type", "properties", { "id": { "type": "integer", "format": "int64" }, "category": { "$ref": "#/components/schemas/Category", "name": { "type": "string", "example": "doggie" }, "photoUrls": { "type": "array", "items": { "type": "string", "xml": { "name": "photoUrl" }, "xml": { "wrapped": true } }, "tags": { "type": "array", "items": { "$ref": "#/components/schemas/Tag", "xml": { "wrapped": true } }, "status": { "enum": [ "available", "pending", "sold" ], "type": "string", "description": "pet status in the store" } }, "xml": { "name": "Pet" } }
```

Category

```
{ "type": "object", "properties": { "id": { "type": "integer", "format": "int64" }, "name": { "type": "string" } }, "xml": { "name": "Category" } }
```

Tag

```
{ "type": "object", "properties": { "id": { "type": "integer", "format": "int64" }, "name": { "type": "string" } }, "xml": { "name": "Tag" } }
```

Response 405 - Method Not Allowed

Invalid input

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

POST/pet/{petId} ■

Updates a pet in the store with form data

Operation ID `updatePetWithForm`



Authorization

- `petstore_auth` OAuth 2.0 `write:pets` `read:pets`

Path Parameters

| Parameter | Description |
|----------------------------|------------------------------------|
| <code>petId</code> : int64 | ID of pet that needs to be updated |

Request body

Media type `application/x-www-form-urlencoded`

Example

object

```
{
  "name": "string",
  "status": "string"
}
```

Schema

object

| Property | Description |
|------------------------------|---------------------------|
| <code>name</code> : string | Updated name of the pet |
| <code>status</code> : string | Updated status of the pet |

Schema JSON

object

```
{"type":"object","properties":{"name":{"type":"string","description":"Updated name of the pet"},
"status":{"type":"string","description":"Updated status of the pet"}}
```

Response 405 - Method Not Allowed

Invalid input

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

POST/pet/{petId}/uploadImage 📷

uploads an image

Operation ID `uploadFile`



Authorization

- `petstore_auth` OAuth 2.0 `write:pets` `read:pets`

Path Parameters

| Parameter | Description |
|--------------|---------------------|
| petId: int64 | ID of pet to update |

Request body

Media type multipart/form-data

Example

object

```
{
  "additionalMetadata": "string",
  "file": "string"
}
```

Schema

object

| Property | Description |
|----------------------------|-----------------------------------|
| additionalMetadata: string | Additional data to pass to server |
| file: binary | file to upload |

Schema JSON

object

```
{"type":"object","properties":{"additionalMetadata":{"type":"string","description":"Additional data to pass to server"},"file":{"type":"string","description":"file to upload","format":"binary"}}
```

Response 200 - OK

successful operation

Media type application/json

Example

ApiResponse

```
{"code":0,"type":"string","message":"string"}
```

Schema

ApiResponse

| Property | Description |
|-----------------|-------------|
| code: int32 | |
| type: string | |
| message: string | |

Schema JSON

ApiResponse

```
{"type":"object","properties":{"code":{"type":"integer","format":"int32"},"type":{"type":"string"},"message":{"type":"string"}}
```

PUT/pet

Update an existing pet

Operation ID `updatePet`



Authorization

- `petstore_authOAuth 2.0write:petsread:pets`

Request body Required

Pet object that needs to be added to the store

Media type `application/json`

Example

Pet

```
{"id":0,"category":{"id":0,"name":"string"},"name":"doggie","photoUrls":["string"],"tags":[{"id":0,"name":"string"}],"status":"available"}
```

Schema

Pet

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| category: Category | |
| name: string | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "Pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "Category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "Tag"
  }
}
```

Media type application/xml

Example

Pet

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Schema

Pet

| Property | Description |
|------------------------------------|-------------------------|
| id: int64 | |
| category: Category | |
| name: string | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingsold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{ "required": [ "name", "photoUrls", "type", "properties", { "id": { "type": "integer", "format": "int64" }, "category": { "$ref": "#/components/schemas/Category", "name": { "type": "string", "example": "doggie" }, "photoUrls": { "type": "array", "items": { "type": "string", "xml": { "name": "photoUrl" }, "xml": { "wrapped": true } }, "tags": { "type": "array", "items": { "$ref": "#/components/schemas/Tag", "xml": { "wrapped": true } }, "status": { "enum": [ "available", "pending", "sold" ], "type": "string", "description": "pet status in the store" } }, "xml": { "name": "Pet" } }
```

Category

```
{ "type": "object", "properties": { "id": { "type": "integer", "format": "int64" }, "name": { "type": "string" } }, "xml": { "name": "Category" } }
```

Tag

```
{ "type": "object", "properties": { "id": { "type": "integer", "format": "int64" }, "name": { "type": "string" } }, "xml": { "name": "Tag" } }
```

Response 400 - Bad Request

Invalid ID supplied

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Response 404 - Not Found

Pet not found

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Response 405 - Method Not Allowed

Validation exception

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type `application/xml`

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

`DELETE/pet/{petId}` 🐕

Deletes a pet

Operation ID `deletePet`



Authorization

- `petstore_authOAuth 2.0write:petsread:pets`

Path Parameters

| Parameter | Description |
|---------------------------|------------------|
| <code>petId: int64</code> | Pet id to delete |

Header Parameters

| Parameter | Description |
|------------------------------|-------------|
| <code>api_key: string</code> | |

Response `400 - Bad Request`

Invalid ID supplied

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Response 404 - Not Found

Pet not found

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

store

Access to Petstore orders

GET/store/inventory 📌

Returns pet inventories by status

Operation ID `getInventory`

Returns a map of status codes to quantities



Authorization

- `api_key` API key `api_key` in header

Response 200 - OK

successful operation

Media type `application/json`

Example

[string: int32]

```
{"additionalProp1":0}
```

Schema

[string: int32]

| Property | Description |
|------------------------|-------------|
| value: [string: int32] | |

Schema JSON

[string: int32]

```
{"type":"object","additionalProperties":{"type":"integer","format":"int32"}}
```

GET/store/order/{orderId}

Find purchase order by ID

Operation ID getOrderById

For valid response try integer IDs with value ≥ 1 and ≤ 10 . Other values will generated exceptions

Path Parameters

| Parameter | Description |
|-----------------------|------------------------------------|
| orderId: int64? 1? 10 | ID of pet that needs to be fetched |

Response 200 - OK

successful operation

Media type application/json

Example

Order

```
{"id":0,"petId":0,"quantity":0,"shipDate":"2025-01-31T12:00:00Z","status":"placed","complete":false}
```

Schema

Order

| Property | Description |
|---------------------------------------|--------------|
| id: int64 | |
| petId: int64 | |
| quantity: int32 | |
| shipDate: dateTime | |
| status: stringplacedapproveddelivered | Order Status |
| complete: boolean | |

Schema JSON

Order

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64"},"petId":{"type":"integer","format":"int64"},"quantity":{"type":"integer","format":"int32"},"shipDate":{"type":"string","format":"date-time"},"status":{"enum":["placed","approved","delivered"],"type":"string","description":"Order Status"},"complete":{"type":"boolean"}},"xml":{"name":"Order"}}
```

Media type application/xml

Example

Order

```
{
  "id": 0,
  "petId": 0,
  "quantity": 0,
  "shipDate": "2025-01-31T12:00:00Z",
  "status": "placed",
  "complete": false
}
```

Schema

Order

| Property | Description |
|---------------------------------------|--------------|
| id: int64 | |
| petId: int64 | |
| quantity: int32 | |
| shipDate: dateTime | |
| status: stringplacedapproveddelivered | Order Status |
| complete: boolean | |

Schema JSON

Order

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64"},"petId":{"type":"integer","format":"int64"},"quantity":{"type":"integer","format":"int32"},"shipDate":{"type":"string","format":"date-time"},"status":{"enum":["placed","approved","delivered"],"type":"string","description":"Order Status"},"complete":{"type":"boolean"}},"xml":{"name":"Order"}}
```

Response 400 - Bad Request

Invalid ID supplied

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Response 404 - Not Found

Order not found

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

POST/store/order

Place an order for a pet

Operation ID placeOrder

Request body Required

order placed for purchasing the pet

Media type application/json

Example

Order

```
{"id":0,"petId":0,"quantity":0,"shipDate":"2025-01-31T12:00:00Z","status":"placed","complete":false}
```

Schema

Order

| Property | Description |
|---------------------------------------|--------------|
| id: int64 | |
| petId: int64 | |
| quantity: int32 | |
| shipDate: dateTime | |
| status: stringplacedapproveddelivered | Order Status |
| complete: boolean | |

Schema JSON

Order

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64"},"petId":{"type":"integer","format":"int64"},"quantity":{"type":"integer","format":"int32"},"shipDate":{"type":"string","format":"date-time"},"status":{"enum":["placed","approved","delivered"],"type":"string","description":"Order Status"},"complete":{"type":"boolean"},"xml":{"name":"Order"}}
```

Response 200 - OK

successful operation

Media type application/json

Example

Order

```
{"id":0,"petId":0,"quantity":0,"shipDate":"2025-01-31T12:00:00Z","status":"placed","complete":false}
```

Schema

Order

| Property | Description |
|----------|-------------|
|----------|-------------|

id: int64

petId: int64

quantity: int32

shipDate: dateTime

status: stringplacedapproveddelivered

Order Status

complete: boolean

Schema JSON

Order

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "petId": {"type": "integer", "format": "int64"}, "quantity": {"type": "integer", "format": "int32"}, "shipDate": {"type": "string", "format": "date-time"}, "status": {"enum": ["placed", "approved", "delivered"], "type": "string", "description": "Order Status"}, "complete": {"type": "boolean"}}, "xml": {"name": "Order"}}
```

Media type application/xml

Example

Order

```
{
  "id": 0,
  "petId": 0,
  "quantity": 0,
  "shipDate": "2025-01-31T12:00:00Z",
  "status": "placed",
  "complete": false
}
```

Schema

Order

| Property | Description |
|---------------------------------------|--------------|
| id: int64 | |
| petId: int64 | |
| quantity: int32 | |
| shipDate: dateTime | |
| status: stringplacedapproveddelivered | Order Status |
| complete: boolean | |

Schema JSON

Order

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "petId": {"type": "integer", "format": "int64"}, "quantity": {"type": "integer", "format": "int32"}, "shipDate": {"type": "string", "format": "date-time"}, "status": {"enum": ["placed", "approved", "delivered"], "type": "string", "description": "Order Status"}, "complete": {"type": "boolean"}}, "xml": {"name": "Order"}}
```

Response 400 - Bad Request

Invalid Order

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

DELETE/store/order/{orderId}

Delete purchase order by ID

Operation ID deleteOrder

For valid response try integer IDs with positive integer value. Negative or non-integer values will generate API errors

Path Parameters

| Parameter | Description |
|-------------------|--|
| orderId: int64? 1 | ID of the order that needs to be deleted |

Response 400 - Bad Request

Invalid ID supplied

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Response 404 - Not Found

Order not found

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

user

Operations about user

GET/user/login

Logs user into the system

Operation ID `loginUser`

Query Parameters

| Parameter | Description |
|------------------|--------------------------------------|
| username: string | The user name for login |
| password: string | The password for login in clear text |

Response 200 - OK

successful operation

Response headers

| Header | Description |
|---------------------------|------------------------------------|
| X-Expires-After: dateTime | date in UTC when token expires |
| X-Rate-Limit: int32 | calls per hour allowed by the user |

Media type `application/json`

Example

`string`

```
"string"
```

Schema

`string`

| Property | Description |
|---------------|-------------|
| value: string | |

Schema JSON

string

```
{"type": "string"}
```

Media type application/xml

Example

string

```
string
```

Schema

string

| Property | Description |
|---------------|-------------|
| value: string | |

Schema JSON

string

```
{"type": "string"}
```

Response 400 - Bad Request

Invalid username/password supplied

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

GET/user/logout

Logs out current logged in user session

Operation ID `logoutUser`

Response default - Default

successful operation

Media type `application/json`

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type `application/xml`

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

GET/user/{username}

Get user by user name

Operation ID `getUserByName`

Path Parameters

| Parameter | Description |
|------------------|---|
| username: string | The name that needs to be fetched. Use user1 for testing. |

Response 200 - OK

successful operation

Media type application/json

Example

User

```
{"id":0,"username":"string","firstName":"string","lastName":"string","email":"string","password":"string","phone":"string","userStatus":0}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64"},"username":{"type":"string"},"firstName":{"type":"string"},"lastName":{"type":"string"},"email":{"type":"string"},"password":{"type":"string"},"phone":{"type":"string"},"userStatus":{"type":"integer"},"description":{"type":"string","format":"int32"},"xml":{"name":"User"}}
```

Media type application/xml

Example

User

```
{  
  "id": 0,  
  "username": "string",  
  "firstName": "string",  
  "lastName": "string",  
  "email": "string",  
  "password": "string",  
  "phone": "string",  
  "userStatus": 0,  
  "description": "string",  
  "xml": {"name": "User"}  
}
```

```
"email": "string",
"password": "string",
"phone": "string",
"userStatus": 0
}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "username": {"type": "string"}, "firstName": {"type": "string"}, "lastName": {"type": "string"}, "email": {"type": "string"}, "password": {"type": "string"}, "phone": {"type": "string"}, "userStatus": {"type": "integer", "format": "int32"}, "description": {"type": "string", "format": "int32"}}, "xml": {"name": "User"}}
```

Response 400 - Bad Request

Invalid username supplied

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Response 404 - Not Found

User not found

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

POST/user

Create user

Operation ID createUser

This can only be done by the logged in user.

Request body Required

Created user object

Media type application/json

Example

User

```
{"id":0,"username":"string","firstName":"string","lastName":"string","email":"string","password":"string","phone":"string","userStatus":0}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64"},"username":{"type":"string"},"firstName":{"type":"string"},"lastName":{"type":"string"},"email":{"type":"string"},"password":{"type":"string"},"phone":{"type":"string"},"userStatus":{"type":"integer","description":"User Status","format":"int32"},"xml":{"name":"User"}}
```

Response default - Default

successful operation

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

POST/user/createWithArray

Creates list of users with given input array

Operation ID createUsersWithArrayInput

Request body Required

List of user object

Media type application/json

Example

[User]

```
[{"id":0,"username":"string","firstName":"string","lastName":"string","email":"string","password":"string","phone":"string","userStatus":0}]
```

Schema

[User]

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

[User]

```
{"type": "array", "items": {"$ref": "#/components/schemas/User"}}
```

User

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "username": {"type": "string"}, "firstName": {"type": "string"}, "lastName": {"type": "string"}, "email": {"type": "string"}, "password": {"type": "string"}, "phone": {"type": "string"}, "userStatus": {"type": "integer", "description": "User Status", "format": "int32"}}, "xml": {"name": "User"}}
```

Response default - Default

successful operation

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

POST/user/createWithList

Creates list of users with given input array

Operation ID createUsersWithListInput

Request body Required

List of user object

Media type application/json

Example

[User]

```
[{"id":0,"username":"string","firstName":"string","lastName":"string","email":"string","password":"string","phone":"string","userStatus":0}]
```

Schema

[User]

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

[User]

```
{"type":"array","items":{"$ref":"#/components/schemas/User"}}
```

User

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64"},"username":{"type":"string"},"firstName":{"type":"string"},"lastName":{"type":"string"},"email":{"type":"string"},"password":{"type":"string"},"phone":{"type":"string"},"userStatus":{"type":"integer","description":"User Status","format":"int32"},"xml":{"name":"User"}}
```

Response default - Default

successful operation

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

PUT/user/{username}

Updated user

Operation ID updateUser

This can only be done by the logged in user.

Path Parameters

| Parameter | Description |
|------------------|------------------------------|
| username: string | name that need to be updated |

Request body Required

Updated user object

Media type application/json

Example

User

```
{"id":0,"username":"string","firstName":"string","lastName":"string","email":"string","password":"string","phone":"string","userStatus":0}
```

Schema

User

| Property | Description |
|------------------|-------------|
| id: int64 | |
| username: string | |

| | |
|-------------------|-------------|
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "username": {
      "type": "string"
    },
    "firstName": {
      "type": "string"
    },
    "lastName": {
      "type": "string"
    },
    "email": {
      "type": "string"
    },
    "password": {
      "type": "string"
    },
    "phone": {
      "type": "string"
    },
    "userStatus": {
      "type": "integer",
      "description": "User Status",
      "format": "int32"
    }
  },
  "xml": {
    "name": "User"
  }
}
```

Response 400 - Bad Request

Invalid user supplied

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Response 404 - Not Found

User not found

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

DELETE/user/{username}

Delete user

Operation ID deleteUser

This can only be done by the logged in user.

Path Parameters

| Parameter | Description |
|------------------|-----------------------------------|
| username: string | The name that needs to be deleted |

Response 400 - Bad Request

Invalid username supplied

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Response 404 - Not Found

User not found

Media type application/json

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Media type application/xml

Example

No example is available.

Schema

No schema is defined.

Schema JSON

No schema is defined.

Swagger Petstore OpenAPI 3.0 example

This page renders the [Swagger Petstore - OpenAPI 3.0](#) sample API. It describes the same pets, store orders, and users as the [Swagger 2.0 Petstore example](#), but uses OpenAPI 3.0 with reusable components, request bodies in several media types, and enums.

The page needs only this front matter:

```
---openapi: ./openapi-petstore-v3.json
---
```

`openapi-petstore-v3.json` is the specification from <https://petstore3.swagger.io/api/v3/openapi.json>, formatted with indentation. The specification declares the [Apache License 2.0](#).

API overview

Swagger Petstore - OpenAPI 3.0

Version `1.0.27`

License [Apache 2.0](#) · Contact apiteam@swagger.io · [Terms of service](#)

This is a sample Pet Store Server based on the OpenAPI 3.0 specification. You can find out more about Swagger at <https://swagger.io>. In the third iteration of the pet store, we've switched to the design first approach! You can now help us improve the API whether it's by making changes to the definition itself or to the code. That way, with time, we can improve the API in general, and expose some of the new features in OAS3.

Some useful links:

- [The Pet Store repository](#)
- [The source API definition for the Pet Store](#)

Server

- `/api/v3`

Authentication

- `petstore_auth`OAuth 2.0
- `api_key`API key `api_key` in header

pet

Everything about your Pets

GET /pet/findByStatus ■

Finds Pets by status.

Operation ID `findPetsByStatus`

Multiple status values can be provided with comma separated strings.



Authorization

- `petstore_auth` OAuth 2.0 `write:pets` `read:pets`

Query Parameters

| Parameter | Description |
|---|---|
| status: string = "available"availablepending sold | Status values that need to be considered for filter |

Response 200 - OK

successful operation

Media type `application/json`

Example

[Pet]

```
[{"id":10,"name":"doggie","category":{"id":1,"name":"Dogs"},"photoUrls":["string"],"tags":[{"id":0,"name":"string"}],"status":"available"}]
```

Schema

[Pet]

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablepending sold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

[Pet]

```
{"type": "array", "items": {"$ref": "#/components/schemas/Pet"}}
```

Pet

```
{"required": ["name", "photoUrls"], "type": "object", "properties": {"id": {"type": "integer", "format": "int64", "example": 10}, "name": {"type": "string", "example": "doggie"}, "category": {"$ref": "#/components/schemas/Category"}, "photoUrls": {"type": "array", "items": {"type": "string", "xml": {"name": "photoUrl"}}}, "xml": {"wrapped": true}}, "tags": {"type": "array", "items": {"$ref": "#/components/schemas/Tag"}, "xml": {"wrapped": true}}, "status": {"enum": ["available", "pending", "sold"], "type": "string", "description": "pet status in the store"}, "xml": {"name": "pet"}}
```

Category

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64", "example": 1}, "name": {"type": "string", "example": "Dogs"}, "xml": {"name": "category"}}
```

Tag

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string"}, "xml": {"name": "tag"}}
```

Media type application/xml

Example

[Pet]

```
[
  {
    "id": 10,
    "name": "doggie",
    "category": {
      "id": 1,
      "name": "Dogs"
    },
    "photoUrls": [
      "string"
    ],
    "tags": [
      {
        "id": 0,
        "name": "string"
      }
    ],
    "status": "available"
  }
]
```

Schema

[Pet]

| Property | Description |
|------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingsold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

[Pet]

```
{"type":"array","items":{"$ref":"#/components/schemas/Pet"}}
```

Pet

```
{"required":["name","photoUrls"],"type":"object","properties":{"id":{"type":"integer","format":"int64","example":10},"name":{"type":"string","example":"doggie"},"category":{"$ref":"#/components/schemas/Category"},"photoUrls":{"type":"array","items":{"type":"string","xml":{"name":"photoUrl"},"xml":{"wrapped":true}}},"tags":{"type":"array","items":{"$ref":"#/components/schemas/Tag"},"xml":{"wrapped":true},"status":{"enum":["available","pending","sold"],"type":"string","description":"pet status in the store"},"xml":{"name":"pet"}}
```

Category

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64","example":1},"name":{"type":"string","example":"Dogs"},"xml":{"name":"category"}}
```

Tag

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64"},"name":{"type":"string"},"xml":{"name":"tag"}}
```

Response 400 - Bad Request

Invalid status value

Response default - Default

Unexpected error

GET /pet/findByTags 🐾

Finds Pets by tags.

Operation ID `findPetsByTags`

Multiple tags can be provided with comma separated strings. Use tag1, tag2, tag3 for testing.



Authorization

- `petstore_auth` OAuth 2.0 `write:pets` `read:pets`

Query Parameters

| Parameter | Description |
|----------------|-------------------|
| tags: [string] | Tags to filter by |

Response 200 - OK

successful operation

Media type `application/json`

Example

[Pet]

```
[{"id":10,"name":"doggie","category":{"id":1,"name":"Dogs"},"photoUrls":["string"],"tags":[{"id":0,"name":"string"}],"status":"available"}]
```

Schema

[Pet]

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

[Pet]

```
{"type": "array", "items": {"$ref": "#/components/schemas/Pet"}}
```

Pet

```
{"required": ["name", "photoUrls"], "type": "object", "properties": {"id": {"type": "integer", "format": "int64", "example": 10}, "name": {"type": "string", "example": "doggie"}, "category": {"$ref": "#/components/schemas/Category"}, "photoUrls": {"type": "array", "items": {"type": "string", "xml": {"name": "photoUrl"}}}, "xml": {"wrapped": true}}, "tags": {"type": "array", "items": {"$ref": "#/components/schemas/Tag"}, "xml": {"wrapped": true}}, "status": {"enum": ["available", "pending", "sold"], "type": "string", "description": "pet status in the store"}, "xml": {"name": "pet"}}
```

Category

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64", "example": 1}, "name": {"type": "string", "example": "Dogs"}, "xml": {"name": "category"}}
```

Tag

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string"}, "xml": {"name": "tag"}}
```

Media type application/xml

Example

[Pet]

```
[
  {
    "id": 10,
    "name": "doggie",
    "category": {
      "id": 1,
      "name": "Dogs"
    },
    "photoUrls": [
      "string"
    ],
    "tags": [
      {
        "id": 0,
        "name": "string"
      }
    ],
    "status": "available"
  }
]
```

Schema

[Pet]

| Property | Description |
|------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingsold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

[Pet]

```
{"type": "array", "items": {"$ref": "#/components/schemas/Pet"}}
```

Pet

```
{"required": ["name", "photoUrls"], "type": "object", "properties": {"id": {"type": "integer", "format": "int64", "example": 10}, "name": {"type": "string", "example": "doggie"}, "category": {"$ref": "#/components/schemas/Category"}, "photoUrls": {"type": "array", "items": {"type": "string", "xml": {"name": "photoUrl", "wrapped": true}}, "tags": {"type": "array", "items": {"$ref": "#/components/schemas/Tag"}, "xml": {"wrapped": true}}, "status": {"enum": ["available", "pending", "sold"], "type": "string", "description": "pet status in the store"}, "xml": {"name": "pet"}}
```

Category

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64", "example": 1}, "name": {"type": "string", "example": "Dogs"}, "xml": {"name": "category"}}
```

Tag

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string", "xml": {"name": "tag"}}
```

Response 400 - Bad Request

Invalid tag value

Response default - Default

Unexpected error

GET /pet/{petId} 🐾

Find pet by ID.

Operation ID `getPetById`

Returns a single pet.



Authorization

- `api_key` API key `api_key` in header
- `petstore_auth` OAuth 2.0 `write:pets` `read:pets`

Path Parameters

| Parameter | Description |
|----------------------------|---------------------|
| <code>petId</code> : int64 | ID of pet to return |

Response 200 - OK

successful operation

Media type `application/json`

Example

Pet

```
{"id":10,"name":"doggie","category":{"id":1,"name":"Dogs"},"photoUrls":["string"],"tags":[{"id":0,"name":"string"}],"status":"available"}
```

Schema

Pet

| Property | Description |
|--|-------------------------|
| <code>id</code> : int64 | |
| <code>name</code> : string | |
| <code>category</code> : Category | |
| <code>photoUrls</code> : [string] | |
| <code>tags</code> : [Tag] | |
| <code>status</code> : string <code>available</code> <code>pending</code> <code>sold</code> | pet status in the store |

Category

| Property | Description |
|----------------------------|-------------|
| <code>id</code> : int64 | |
| <code>name</code> : string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 1
    },
    "name": {
      "type": "string",
      "example": "Dogs"
    }
  },
  "xml": {
    "name": "category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "tag"
  }
}
```

Media type application/xml

Example

Pet

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Schema

Pet

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

category: Category

photoUrls: [string]

tags: [Tag]

status: stringavailablependingssold

pet status in the store

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls", "type"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 1
    },
    "name": {
      "type": "string",
      "example": "Dogs"
    }
  },
  "xml": {
    "name": "category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "tag"
  }
}
```

Response 400 - Bad Request

Invalid ID supplied

Response 404 - Not Found

Pet not found

Response default - Default

Unexpected error

POST/pet 🐾

Add a new pet to the store.

Operation ID `addPet`



Authorization

- `petstore_authOAuth 2.0write:petsread:pets`

Request body Required

Create a new pet in the store

Media type `application/json`

Example

Pet

```
{ "id": 10, "name": "doggie", "category": { "id": 1, "name": "Dogs" }, "photoUrls": [ "string" ], "tags": [ { "id": 0, "name": "string" } ], "status": "available" }
```

Schema

Pet

| Property | Description |
|------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingsold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{ "required": [ "name", "photoUrls" ], "type": "object", "properties": { "id": { "type": "integer", "format": "int64", "example": 10 }, "name": { "type": "string", "example": "doggie" }, "category": { "$ref": "#/components/schemas/Category" }, "photoUrls": { "type": "array", "items": { "type": "string", "xml": { "name": "phot
```

```
oUrl"}}, "xml": {"wrapped": true}}, "tags": {"type": "array", "items": {"$ref": "#/components/schemas/Tag"}, "xml": {"wrapped": true}}, "status": {"enum": ["available", "pending", "sold"], "type": "string", "description": "pet status in the store"}}, "xml": {"name": "pet"}}
```

Category

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64", "example": 1}, "name": {"type": "string", "example": "Dogs"}}, "xml": {"name": "category"}}
```

Tag

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string"}}, "xml": {"name": "tag"}}
```

Media type application/xml

Example

Pet

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Schema

Pet

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 1
    },
    "name": {
      "type": "string",
      "example": "Dogs"
    }
  },
  "xml": {
    "name": "category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "tag"
  }
}
```

Media type application/x-www-form-urlencoded

Example

Pet

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Schema

Pet

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

category: Category

photoUrls: [string]

tags: [Tag]

status: stringavailablependingssold

pet status in the store

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{"required":["name","photoUrls"],"type":"object","properties":{"id":{"type":"integer","format":"int64","example":10},"name":{"type":"string","example":"doggie"},"category":{"$ref":"#/components/schemas/Category"},"photoUrls":{"type":"array","items":{"type":"string","xml":{"name":"photoUrl"}},"xml":{"wrapped":true}},"tags":{"type":"array","items":{"$ref":"#/components/schemas/Tag"},"xml":{"wrapped":true}},"status":{"enum":["available","pending","sold"],"type":"string","description":"pet status in the store"},"xml":{"name":"pet"}}
```

Category

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64","example":1},"name":{"type":"string","example":"Dogs"},"xml":{"name":"category"}}
```

Tag

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64"},"name":{"type":"string"},"xml":{"name":"tag"}}
```

Response 200 - OK

Successful operation

Media type application/json

Example

Pet

```
{"id":10,"name":"doggie","category":{"id":1,"name":"Dogs"},"photoUrls":["string"],"tags":[{"id":0,"name":"string"}],"status":"available"}
```

Schema

Pet

| Property | Description |
|------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingsold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 1
    },
    "name": {
      "type": "string",
      "example": "Dogs"
    }
  },
  "xml": {
    "name": "category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "tag"
  }
}
```

Media type application/xml

Example

Pet

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "http://example.com/pet1.jpg",
    "http://example.com/pet2.jpg"
  ],
  "tags": [
    "dog",
    "pet"
  ],
  "status": "available"
}
```

```
"photoUrls": [
  "string"
],
"tags": [
  {
    "id": 0,
    "name": "string"
  }
],
"status": "available"
}
```

Schema

Pet

| Property | Description |
|------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingsold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 1
    },
    "name": {
      "type": "string",
      "example": "Dogs"
    }
  },
  "xml": {
    "name": "category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  }
}
```

```
}}, "xml": {"name": "tag"}}
```

Response 400 - Bad Request

Invalid input

Response 422 - Unprocessable Entity

Validation exception

Response default - Default

Unexpected error

POST/pet/{petId} ■

Updates a pet in the store with form data.

Operation ID `updatePetWithForm`

Updates a pet resource based on the form data.



Authorization

- `petstore_auth` OAuth 2.0 `write:pets` `read:pets`

Path Parameters

| Parameter | Description |
|----------------------------|------------------------------------|
| <code>petId</code> : int64 | ID of pet that needs to be updated |

Query Parameters

| Parameter | Description |
|------------------------------|--|
| <code>name</code> : string | Name of pet that needs to be updated |
| <code>status</code> : string | Status of pet that needs to be updated |

Response 200 - OK

successful operation

Media type `application/json`

Example

Pet

```
{"id":10,"name":"doggie","category":{"id":1,"name":"Dogs"},"photoUrls":["string"],"tags":[{"id":0,"name":"string"}],"status":"available"}
```

Schema

Pet

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 1
    },
    "name": {
      "type": "string",
      "example": "Dogs"
    }
  },
  "xml": {
    "name": "category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "tag"
  }
}
```

Media type application/xml

Example

Pet

```
{
  "id": 10,
```

```

"name": "doggie",
"category": {
  "id": 1,
  "name": "Dogs"
},
"photoUrls": [
  "string"
],
"tags": [
  {
    "id": 0,
    "name": "string"
  }
],
"status": "available"
}

```

Schema

Pet

| Property | Description |
|------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingsold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```

{"required":["name","photoUrls"],"type":"object","properties":{"id":{"type":"integer","format":"int64","example":10},"name":{"type":"string","example":"doggie"},"category":{"$ref":"#/components/schemas/Category"},"photoUrls":{"type":"array","items":{"type":"string","xml":{"name":"photoUrl"},"xml":{"wrapped":true}}},"tags":{"type":"array","items":{"$ref":"#/components/schemas/Tag"},"xml":{"wrapped":true}},"status":{"enum":["available","pending","sold"],"type":"string","description":"pet status in the store"},"xml":{"name":"pet"}}

```

Category

```

{"type":"object","properties":{"id":{"type":"integer","format":"int64","example":1},"name":{"ty

```

```
pe":"string", "example":"Dogs"}}, "xml":{"name":"category"}}
```

Tag

```
{"type":"object", "properties":{"id":{"type":"integer", "format":"int64"}, "name":{"type":"string"}}, "xml":{"name":"tag"}}
```

Response 400 - Bad Request

Invalid input

Response default - Default

Unexpected error

POST /pet/{petId}/uploadImage 📄

Uploads an image.

Operation ID `uploadFile`

Upload image of the pet.



Authorization

- `petstore_auth`OAuth 2.0 `write:pets``read:pets`

Path Parameters

| Parameter | Description |
|--------------|---------------------|
| petId: int64 | ID of pet to update |

Query Parameters

| Parameter | Description |
|----------------------------|---------------------|
| additionalMetadata: string | Additional Metadata |

Request body

Media type `application/octet-stream`

Example

binary

```
string
```

Schema

binary

| Property | Description |
|----------|-------------|
|----------|-------------|

value: binary

Schema JSON

binary

```
{"type": "string", "format": "binary"}
```

Response 200 - OK

successful operation

Media type application/json

Example

ApiResponse

```
{"code": 0, "type": "string", "message": "string"}
```

Schema

ApiResponse

| Property | Description |
|-----------------|-------------|
| code: int32 | |
| type: string | |
| message: string | |

Schema JSON

ApiResponse

```
{"type": "object", "properties": {"code": {"type": "integer", "format": "int32"}, "type": {"type": "string"}, "message": {"type": "string"}}, "xml": {"name": "##default"}}
```

Response 400 - Bad Request

No file uploaded

Response 404 - Not Found

Pet not found

Response default - Default

Unexpected error

PUT/pet

Update an existing pet.

Operation ID updatePet

Update an existing pet by Id.



Authorization

- petstore_auth OAuth 2.0 write:pets read:pets

Request body Required

Update an existent pet in the store

Media type application/json

Example

Pet

```
{ "id": 10, "name": "doggie", "category": { "id": 1, "name": "Dogs" }, "photoUrls": [ "string" ], "tags": [ { "id": 0, "name": "string" } ], "status": "available" }
```

Schema

Pet

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablepending sold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{ "required": [ "name", "photoUrls" ], "type": "object", "properties": { "id": { "type": "integer", "format": "int64", "example": 10 }, "name": { "type": "string", "example": "doggie" }, "category": { "$ref": "#/components/schemas/Category" }, "photoUrls": { "type": "array", "items": { "type": "string", "xml": { "name": "phot
```

```
oUrl"}, "xml": {"wrapped": true}}, "tags": {"type": "array", "items": {"$ref": "#/components/schemas/Tag"}, "xml": {"wrapped": true}}, "status": {"enum": ["available", "pending", "sold"], "type": "string", "description": "pet status in the store"}}, "xml": {"name": "pet"}}
```

Category

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64", "example": 1}, "name": {"type": "string", "example": "Dogs"}}, "xml": {"name": "category"}}
```

Tag

```
{"type": "object", "properties": {"id": {"type": "integer", "format": "int64"}, "name": {"type": "string"}}, "xml": {"name": "tag"}}
```

Media type application/xml

Example

Pet

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Schema

Pet

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 1
    },
    "name": {
      "type": "string",
      "example": "Dogs"
    }
  },
  "xml": {
    "name": "category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "tag"
  }
}
```

Media type application/x-www-form-urlencoded

Example

Pet

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Schema

Pet

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

category: Category

photoUrls: [string]

tags: [Tag]

status: stringavailablependingssold

pet status in the store

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 1
    },
    "name": {
      "type": "string",
      "example": "Dogs"
    }
  },
  "xml": {
    "name": "category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "tag"
  }
}
```

Response 200 - OK

Successful operation

Media type application/json

Example

Pet

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Schema

Pet

| Property | Description |
|------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingsold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 1
    },
    "name": {
      "type": "string",
      "example": "Dogs"
    }
  },
  "xml": {
    "name": "category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "xml": {
    "name": "tag"
  }
}
```

Media type application/xml

Example

Pet

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
}
```

```
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Schema

Pet

| Property | Description |
|-------------------------------------|-------------------------|
| id: int64 | |
| name: string | |
| category: Category | |
| photoUrls: [string] | |
| tags: [Tag] | |
| status: stringavailablependingssold | pet status in the store |

Category

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Tag

| Property | Description |
|--------------|-------------|
| id: int64 | |
| name: string | |

Schema JSON

Pet

```
{
  "required": ["name", "photoUrls"],
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "name": {
      "type": "string",
      "example": "doggie"
    },
    "category": {
      "$ref": "#/components/schemas/Category"
    },
    "photoUrls": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "photoUrl"
        }
      },
      "xml": {
        "wrapped": true
      }
    },
    "tags": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/Tag"
      },
      "xml": {
        "wrapped": true
      }
    },
    "status": {
      "enum": ["available", "pending", "sold"],
      "type": "string",
      "description": "pet status in the store"
    }
  },
  "xml": {
    "name": "pet"
  }
}
```

Category

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 1
    },
    "name": {
      "type": "string",
      "example": "Dogs"
    }
  },
  "xml": {
    "name": "category"
  }
}
```

Tag

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  }
}
```

```
}}, "xml": {"name": "tag"}}
```

Response 400 - Bad Request

Invalid ID supplied

Response 404 - Not Found

Pet not found

Response 422 - Unprocessable Entity

Validation exception

Response default - Default

Unexpected error

DELETE /pet/{petId} 🚧

Deletes a pet.

Operation ID deletePet

Delete a pet.



Authorization

- petstore_auth OAuth 2.0 write:pets read:pets

Path Parameters

| Parameter | Description |
|--------------|------------------|
| petId: int64 | Pet id to delete |

Header Parameters

| Parameter | Description |
|-----------------|-------------|
| api_key: string | |

Response 200 - OK

Pet deleted

Response 400 - Bad Request

Invalid pet value

Response default - Default

Unexpected error

store

Access to Petstore orders

GET/store/inventory ■

Returns pet inventories by status.

Operation ID `getInventory`

Returns a map of status codes to quantities.



Authorization

- `api_key` API key `api_key` in header

Response 200 - OK

successful operation

Media type `application/json`

Example

[string: int32]

```
{"additionalProp1":0}
```

Schema

[string: int32]

| Property | Description |
|------------------------|-------------|
| value: [string: int32] | |

Schema JSON

[string: int32]

```
{"type":"object","additionalProperties":{"type":"integer","format":"int32"}}
```

Response default - Default

Unexpected error

GET/store/order/{orderId}

Find purchase order by ID.

Operation ID `getOrderById`

For valid response try integer IDs with value ≤ 5 or > 10 . Other values will generate exceptions.

Path Parameters

| Parameter | Description |
|----------------|--------------------------------------|
| orderId: int64 | ID of order that needs to be fetched |

Response 200 - OK

successful operation

Media type application/json

Example

Order

```
{"id":10,"petId":198772,"quantity":7,"shipDate":"2025-01-31T12:00:00Z","status":"approved","complete":false}
```

Schema

Order

| Property | Description |
|---------------------------------------|--------------|
| id: int64 | |
| petId: int64 | |
| quantity: int32 | |
| shipDate: dateTime | |
| status: stringplacedapproveddelivered | Order Status |
| complete: boolean | |

Schema JSON

Order

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64","example":10},"petId":{"type":"integer","format":"int64","example":198772},"quantity":{"type":"integer","format":"int32","example":7},"shipDate":{"type":"string","format":"date-time"},"status":{"enum":["placed","approved","delivered"],"type":"string","description":"Order Status","example":"approved"},"complete":{"type":"boolean"}},"xml":{"name":"order"}}
```

Media type application/xml

Example

Order

```
{
  "id": 10,
  "petId": 198772,
  "quantity": 7,
  "shipDate": "2025-01-31T12:00:00Z",
  "status": "approved",
```

```
"complete": false
}
```

Schema

Order

| Property | Description |
|---------------------------------------|--------------|
| id: int64 | |
| petId: int64 | |
| quantity: int32 | |
| shipDate: dateTime | |
| status: stringplacedapproveddelivered | Order Status |
| complete: boolean | |

Schema JSON

Order

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "petId": {
      "type": "integer",
      "format": "int64",
      "example": 198772
    },
    "quantity": {
      "type": "integer",
      "format": "int32",
      "example": 7
    },
    "shipDate": {
      "type": "string",
      "format": "date-time"
    },
    "status": {
      "enum": [
        "placed",
        "approved",
        "delivered"
      ],
      "type": "string",
      "description": "Order Status",
      "example": "approved"
    },
    "complete": {
      "type": "boolean"
    }
  },
  "xml": {
    "name": "order"
  }
}
```

Response 400 - Bad Request

Invalid ID supplied

Response 404 - Not Found

Order not found

Response default - Default

Unexpected error

POST/store/order

Place an order for a pet.

Operation ID placeOrder

Place a new order in the store.

Request body

Media type application/json

Example

Order

```
{"id":10,"petId":198772,"quantity":7,"shipDate":"2025-01-31T12:00:00Z","status":"approved","complete":false}
```

Schema

Order

| Property | Description |
|---------------------------------------|--------------|
| id: int64 | |
| petId: int64 | |
| quantity: int32 | |
| shipDate: dateTime | |
| status: stringplacedapproveddelivered | Order Status |
| complete: boolean | |

Schema JSON

Order

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64","example":10},"petId":{"type":"integer","format":"int64","example":198772},"quantity":{"type":"integer","format":"int32","example":7},"shipDate":{"type":"string","format":"date-time"},"status":{"enum":["placed","approved","delivered"],"type":"string","description":"Order Status","example":"approved"},"complete":{"type":"boolean"}},"xml":{"name":"order"}}
```

Media type application/xml

Example

Order

```
{
  "id": 10,
  "petId": 198772,
  "quantity": 7,
  "shipDate": "2025-01-31T12:00:00Z",
  "status": "approved",
  "complete": false
}
```

Schema

Order

| Property | Description |
|---------------------------------------|--------------|
| id: int64 | |
| petId: int64 | |
| quantity: int32 | |
| shipDate: dateTime | |
| status: stringplacedapproveddelivered | Order Status |
| complete: boolean | |

Schema JSON

Order

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "petId": {
      "type": "integer",
      "format": "int64",
      "example": 198772
    },
    "quantity": {
      "type": "integer",
      "format": "int32",
      "example": 7
    },
    "shipDate": {
      "type": "string",
      "format": "date-time"
    },
    "status": {
      "enum": [
        "placed",
        "approved",
        "delivered"
      ],
      "type": "string",
      "description": "Order Status",
      "example": "approved"
    },
    "complete": {
      "type": "boolean"
    }
  },
  "xml": {
    "name": "order"
  }
}
```

Media type application/x-www-form-urlencoded

Example

Order

```
{
  "id": 10,
  "petId": 198772,
  "quantity": 7,
  "shipDate": "2025-01-31T12:00:00Z",
  "status": "approved",
  "complete": false
}
```

Schema

Order

| Property | Description |
|---------------------------------------|--------------|
| id: int64 | |
| petId: int64 | |
| quantity: int32 | |
| shipDate: dateTime | |
| status: stringplacedapproveddelivered | Order Status |
| complete: boolean | |

Schema JSON

Order

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "petId": {
      "type": "integer",
      "format": "int64",
      "example": 198772
    },
    "quantity": {
      "type": "integer",
      "format": "int32",
      "example": 7
    },
    "shipDate": {
      "type": "string",
      "format": "date-time"
    },
    "status": {
      "enum": [
        "placed",
        "approved",
        "delivered"
      ],
      "type": "string",
      "description": "Order Status",
      "example": "approved"
    },
    "complete": {
      "type": "boolean"
    }
  },
  "xml": {
    "name": "order"
  }
}
```

Response 200 - OK

successful operation

Media type application/json

Example

Order

```
{ "id": 10, "petId": 198772, "quantity": 7, "shipDate": "2025-01-31T12:00:00Z", "status": "approved", "complete": false }
```

Schema

Order

| Property | Description |
|---------------------------------------|--------------|
| id: int64 | |
| petId: int64 | |
| quantity: int32 | |
| shipDate: dateTime | |
| status: stringplacedapproveddelivered | Order Status |
| complete: boolean | |

Schema JSON

Order

```
{ "type": "object", "properties": { "id": { "type": "integer", "format": "int64", "example": 10 }, "petId": { "type": "integer", "format": "int64", "example": 198772 }, "quantity": { "type": "integer", "format": "int32", "example": 7 }, "shipDate": { "type": "string", "format": "date-time" }, "status": { "enum": [ "placed", "approved", "delivered" ], "type": "string", "description": "Order Status", "example": "approved" }, "complete": { "type": "boolean" } }, "xml": { "name": "order" } }
```

Response 400 - Bad Request

Invalid input

Response 422 - Unprocessable Entity

Validation exception

Response default - Default

Unexpected error

DELETE/store/order/{orderId}

Delete purchase order by identifier.

Operation ID deleteOrder

For valid response try integer IDs with value < 1000. Anything above 1000 or non-integers will generate API errors.

Path Parameters

| Parameter | Description |
|----------------|--|
| orderId: int64 | ID of the order that needs to be deleted |

Response 200 - OK

order deleted

Response 400 - Bad Request

Invalid ID supplied

Response 404 - Not Found

Order not found

Response default - Default

Unexpected error

user

Operations about user

GET/user/login

Logs user into the system.

Operation ID loginUser

Log into the system.

Query Parameters

| Parameter | Description |
|------------------|--------------------------------------|
| username: string | The user name for login |
| password: string | The password for login in clear text |

Response 200 - OK

successful operation

Response headers

| Header | Description |
|---------------------------|------------------------------------|
| X-Rate-Limit: int32 | calls per hour allowed by the user |
| X-Expires-After: dateTime | date in UTC when token expires |

Media type application/xml

Example

string

```
string
```

Schema

string

| Property | Description |
|---------------|-------------|
| value: string | |

Schema JSON

string

```
{"type": "string"}
```

Media type application/json

Example

string

```
"string"
```

Schema

string

| Property | Description |
|---------------|-------------|
| value: string | |

Schema JSON

string

```
{"type": "string"}
```

Response 400 - Bad Request

Invalid username/password supplied

Response default - Default

Unexpected error

GET/user/logout

Logs out current logged in user session.

Operation ID `logoutUser`

Log user out of the system.

Response `200 - OK`

successful operation

Response `default - Default`

Unexpected error

GET /user/{username}

Get user by user name.

Operation ID `getUserByName`

Get user detail based on username.

Path Parameters

| Parameter | Description |
|------------------|--|
| username: string | The name that needs to be fetched. Use user1 for testing |

Response `200 - OK`

successful operation

Media type `application/json`

Example

User

```
{ "id": 10, "username": "theUser", "firstName": "John", "lastName": "James", "email": "john@email.com", "password": "12345", "phone": "12345", "userStatus": 1 }
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |

userStatus: int32

User Status

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser"
    },
    "firstName": {
      "type": "string",
      "example": "John"
    },
    "lastName": {
      "type": "string",
      "example": "James"
    },
    "email": {
      "type": "string",
      "example": "john@email.com"
    },
    "password": {
      "type": "string",
      "example": "12345"
    },
    "phone": {
      "type": "string",
      "example": "12345"
    },
    "userStatus": {
      "type": "integer",
      "description": "User Status",
      "format": "int32",
      "example": 1
    }
  },
  "xml": {
    "name": "user"
  }
}
```

Media type application/xml

Example

User

```
{
  "id": 10,
  "username": "theUser",
  "firstName": "John",
  "lastName": "James",
  "email": "john@email.com",
  "password": "12345",
  "phone": "12345",
  "userStatus": 1
}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser"
    },
    "firstName": {
      "type": "string",
      "example": "John"
    },
    "lastName": {
      "type": "string",
      "example": "James"
    },
    "email": {
      "type": "string",
      "example": "john@email.com"
    },
    "password": {
      "type": "string",
      "example": "12345"
    },
    "phone": {
      "type": "string",
      "example": "12345"
    },
    "userStatus": {
      "type": "integer",
      "description": "User Status",
      "format": "int32",
      "example": 1
    }
  },
  "xml": {
    "name": "user"
  }
}
```

Response 400 - Bad Request

Invalid username supplied

Response 404 - Not Found

User not found

Response default - Default

Unexpected error

POST/user

Create user.

Operation ID createUser

This can only be done by the logged in user.

Request body

Created user object

Media type application/json

Example

User

```
{ "id": 10, "username": "theUser", "firstName": "John", "lastName": "James", "email": "john@email.com", "password": "12345", "phone": "12345", "userStatus": 1 }
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser",
      "firstName": {
        "type": "string",
        "example": "John"
      },
      "lastName": {
        "type": "string",
        "example": "James"
      },
      "email": {
        "type": "string",
        "example": "john@email.com"
      },
      "password": {
        "type": "string",
        "example": "12345"
      },
      "phone": {
        "type": "string",
        "example": "12345"
      },
      "userStatus": {
        "type": "integer",
        "description": "User Status",
        "format": "int32",
        "example": 1
      },
      "xml": {
        "name": "user"
      }
    }
  }
}
```

Media type application/xml

Example

User

```
{
  "id": 10,
  "username": "theUser",
  "firstName": "John",
  "lastName": "James",
  "email": "john@email.com",
  "password": "12345",
  "phone": "12345",
  "userStatus": 1
}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser",
      "firstName": {
        "type": "string",
        "example": "John"
      },
      "lastName": {
        "type": "string",
        "example": "James"
      },
      "email": {
        "type": "string",
        "example": "john@email.com"
      },
      "password": {
        "type": "string",
        "example": "12345"
      },
      "phone": {
        "type": "string",
        "example": "12345"
      },
      "userStatus": {
        "type": "integer",
        "description": "User Status",
        "format": "int32",
        "example": 1
      },
      "xml": {
        "name": "user"
      }
    }
  }
}
```

Media type application/x-www-form-urlencoded

Example

User

```
{
```

```
{
  "id": 10,
  "username": "theUser",
  "firstName": "John",
  "lastName": "James",
  "email": "john@email.com",
  "password": "12345",
  "phone": "12345",
  "userStatus": 1
}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser"
    },
    "firstName": {
      "type": "string",
      "example": "John"
    },
    "lastName": {
      "type": "string",
      "example": "James"
    },
    "email": {
      "type": "string",
      "example": "john@email.com"
    },
    "password": {
      "type": "string",
      "example": "12345"
    },
    "phone": {
      "type": "string",
      "example": "12345"
    },
    "userStatus": {
      "type": "integer",
      "description": "User Status",
      "format": "int32",
      "example": 1
    }
  },
  "xml": {
    "name": "user"
  }
}
```

Response 200 - OK

successful operation

Media type application/json

Example

User

```
{
  "id": 10,
  "username": "theUser",
  "firstName": "John",
  "lastName": "James",
  "email": "john@email.com",
  "password": "12345",
  "phone": "12345",
  "userStatus": 1
}
```

Schema

User

| Property | Description |
|-----------|-------------|
| id: int64 | |

| | |
|-------------------|-------------|
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser"
    },
    "firstName": {
      "type": "string",
      "example": "John"
    },
    "lastName": {
      "type": "string",
      "example": "James"
    },
    "email": {
      "type": "string",
      "example": "john@email.com"
    },
    "password": {
      "type": "string",
      "example": "12345"
    },
    "phone": {
      "type": "string",
      "example": "12345"
    },
    "userStatus": {
      "type": "integer",
      "description": "User Status",
      "format": "int32",
      "example": 1
    }
  },
  "xml": {
    "name": "user"
  }
}
```

Media type application/xml

Example

User

```
{
  "id": 10,
  "username": "theUser",
  "firstName": "John",
  "lastName": "James",
  "email": "john@email.com",
  "password": "12345",
  "phone": "12345",
  "userStatus": 1
}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{ "type": "object", "properties": { "id": { "type": "integer", "format": "int64", "example": 10 }, "username": { "type": "string", "example": "theUser" }, "firstName": { "type": "string", "example": "John" }, "lastName": { "type": "string", "example": "James" }, "email": { "type": "string", "example": "iohn@email.com" }, "password": { "type": "string", "example": "12345" }, "phone": { "type": "string", "example": "12345" }, "userStatus": { "type": "integer", "description": "User Status", "format": "int32", "example": 1 }, "xml": { "name": "user" } }
```

Response default - Default

Unexpected error

POST/user/createWithList

Creates list of users with given input array.

Operation ID `createUsersWithListInput`

Request body

Media type `application/json`

Example

[User]

```
[{"id":10,"username":"theUser","firstName":"John","lastName":"James","email":"iohn@email.com","password":"12345","phone":"12345","userStatus":1}]
```

Schema

[User]

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

[User]

```
{ "type": "array", "items": { "$ref": "#/components/schemas/User" } }
```

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser"
    },
    "firstName": {
      "type": "string",
      "example": "John"
    },
    "lastName": {
      "type": "string",
      "example": "James"
    },
    "email": {
      "type": "string",
      "example": "john@email.com"
    },
    "password": {
      "type": "string",
      "example": "12345"
    },
    "phone": {
      "type": "string",
      "example": "12345"
    },
    "userStatus": {
      "type": "integer",
      "description": "User Status",
      "format": "int32",
      "example": 1
    }
  },
  "xml": {
    "name": "user"
  }
}
```

Response 200 - OK

Successful operation

Media type application/json

Example

User

```
{
  "id": 10,
  "username": "theUser",
  "firstName": "John",
  "lastName": "James",
  "email": "john@email.com",
  "password": "12345",
  "phone": "12345",
  "userStatus": 1
}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser"
    },
    "firstName": {
      "type": "string",
      "example": "John"
    },
    "lastName": {
      "type": "string",
      "example": "James"
    },
    "email": {
      "type": "string",
      "example": "john@email.com"
    },
    "password": {
      "type": "string",
      "example": "12345"
    },
    "phone": {
      "type": "string",
      "example": "12345"
    },
    "userStatus": {
      "type": "integer",
      "description": "User Status",
      "format": "int32",
      "example": 1
    }
  },
  "xml": {
    "name": "user"
  }
}
```

Media type application/xml

Example

User

```
{
  "id": 10,
```

```
{
  "username": "theUser",
  "firstName": "John",
  "lastName": "James",
  "email": "john@email.com",
  "password": "12345",
  "phone": "12345",
  "userStatus": 1
}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser"
    },
    "firstName": {
      "type": "string",
      "example": "John"
    },
    "lastName": {
      "type": "string",
      "example": "James"
    },
    "email": {
      "type": "string",
      "example": "john@email.com"
    },
    "password": {
      "type": "string",
      "example": "12345"
    },
    "phone": {
      "type": "string",
      "example": "12345"
    },
    "userStatus": {
      "type": "integer",
      "description": "User Status",
      "format": "int32",
      "example": 1
    }
  },
  "xml": {
    "name": "user"
  }
}
```

Response default - Default

Unexpected error

PUT/user/{username}

Update user resource.

Operation ID updateUser

This can only be done by the logged in user.

Path Parameters

| Parameter | Description |
|------------------|------------------------------|
| username: string | name that need to be deleted |

Request body

Update an existent user in the store

Media type application/json

Example

User

```
{"id":10,"username":"theUser","firstName":"John","lastName":"James","email":"iohn@email.com","password":"12345","phone":"12345","userStatus":1}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{"type":"object","properties":{"id":{"type":"integer","format":"int64","example":10},"username":{"type":"string","example":"theUser"},"firstName":{"type":"string","example":"John"},"lastName":{"type":"string","example":"James"},"email":{"type":"string","example":"iohn@email.com"},"password":{"type":"string","example":"12345"},"phone":{"type":"string","example":"12345"},"userStatus":{"type":"integer","description":"User Status","format":"int32","example":1},"xml":{"name":"user"}}
```

Media type application/xml

Example

User

```
{
  "id": 10,
  "username": "theUser",
  "firstName": "John",
  "lastName": "James",
  "email": "iohn@email.com",
  "password": "12345",
  "phone": "12345",
  "userStatus": 1
}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |
| password: string | |
| phone: string | |
| userStatus: int32 | User Status |

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser"
    },
    "firstName": {
      "type": "string",
      "example": "John"
    },
    "lastName": {
      "type": "string",
      "example": "James"
    },
    "email": {
      "type": "string",
      "example": "john@email.com"
    },
    "password": {
      "type": "string",
      "example": "12345"
    },
    "phone": {
      "type": "string",
      "example": "12345"
    },
    "userStatus": {
      "type": "integer",
      "description": "User Status",
      "format": "int32",
      "example": 1
    }
  },
  "xml": {
    "name": "user"
  }
}
```

Media type application/x-www-form-urlencoded

Example

User

```
{
  "id": 10,
  "username": "theUser",
  "firstName": "John",
  "lastName": "James",
  "email": "john@email.com",
  "password": "12345",
  "phone": "12345",
  "userStatus": 1
}
```

Schema

User

| Property | Description |
|-------------------|-------------|
| id: int64 | |
| username: string | |
| firstName: string | |
| lastName: string | |
| email: string | |

password: string

phone: string

userStatus: int32

User Status

Schema JSON

User

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64",
      "example": 10
    },
    "username": {
      "type": "string",
      "example": "theUser"
    },
    "firstName": {
      "type": "string",
      "example": "John"
    },
    "lastName": {
      "type": "string",
      "example": "James"
    },
    "email": {
      "type": "string",
      "example": "john@email.com"
    },
    "password": {
      "type": "string",
      "example": "12345"
    },
    "phone": {
      "type": "string",
      "example": "12345"
    },
    "userStatus": {
      "type": "integer",
      "description": "User Status",
      "format": "int32",
      "example": 1
    }
  },
  "xml": {
    "name": "user"
  }
}
```

Response 200 - OK

successful operation

Response 400 - Bad Request

bad request

Response 404 - Not Found

user not found

Response default - Default

Unexpected error

DELETE/user/{username}

Delete user resource.

Operation ID deleteUser

This can only be done by the logged in user.

Path Parameters

| Parameter | Description |
|------------------|-----------------------------------|
| username: string | The name that needs to be deleted |

Response 200 - OK

User deleted

Response 400 - Bad Request

Invalid username supplied

Response 404 - Not Found

User not found

Response default - Default

Unexpected error

Tables

MDU renders GitHub-style pipe tables. Wide tables scroll horizontally on small screens instead of breaking the layout.

Basic table

Separate columns with `|` and the header from the body with a row of dashes:

Markdown

```
Command	Purpose	Output
`mdulint`	Check Markdown files	Lint results
`mduserve`	Preview documentation	Local website
`mdurender`	Build static documentation	HTML files
```

| Command | Purpose | Output |
|------------|----------------------------|---------------|
| mdu lint | Check Markdown files | Lint results |
| mdu serve | Preview documentation | Local website |
| mdu render | Build static documentation | HTML files |

Column alignment

Colons in the separator row align a column left, center, or right:

Markdown

```
Left aligned	Center aligned	Right aligned
Markdown	Markdown	Markdown
Short text	Centered text	123.45
```

| Left aligned | Center aligned | Right aligned |
|--------------|----------------|---------------|
| Markdown | Markdown | Markdown |
| Short text | Centered text | 123.45 |

Formatting inside cells

Cells can contain inline Markdown such as emphasis, code, links, icons, and keyboard keys:

Markdown

| | | |
|----------|---|--|
| Feature | Example | |
| ----- | ----- | |
| Emphasis | Important and <i>helpful</i> | |
| Code | <code>`mdurender`</code> | |
| Link | <code>[GitHub](https://github.com)</code> | |
| Icon | <code>:material-check-circle: Done</code> | |
| Keys | <code>++ctrl+c++</code> | |

| Feature | Example |
|----------|-----------------------|
| Emphasis | Important and helpful |
| Code | mdurender |
| Link | GitHub |
| Icon | Done |
| Keys | CtrlCtrl+C |

Architecture

MDU is a single .NET tool. It reads a folder of ordinary Markdown files, builds a page tree from the folder structure, and then lints, renders, or rewrites those pages. Everything a published site needs is produced at build time, so the result is a folder of static files.

Design principles

| Principle | What it means |
|------------------|---|
| Portable sources | Pages stay plain Markdown that reads well on GitHub. MDU features use code blocks, shortcodes, or front matter that degrade to readable text elsewhere. |
| Static output | Diagrams, math, icons, and syntax highlighting become HTML and SVG while rendering. The site needs no server, CDN, or client-side renderer. |
| Few dependencies | One dotnet tool: no Node.js, Python, Java, or browser engine. MDU writes PDF files itself. |
| Scriptable CLI | Commands print plain text and return exit codes; mdurender also writes JSON and GitHub annotations. |

System context

MDU sits between the documentation author and the published site. Click a node to read more.

In this section

- [Components](#) - What each project does and which libraries it uses
- [Processing Pipeline](#) - How commands build the page tree, render a site, and refresh the live preview
- [Rendering](#) - How a Markdown page becomes an HTML page with diagrams, math, and icons

Components

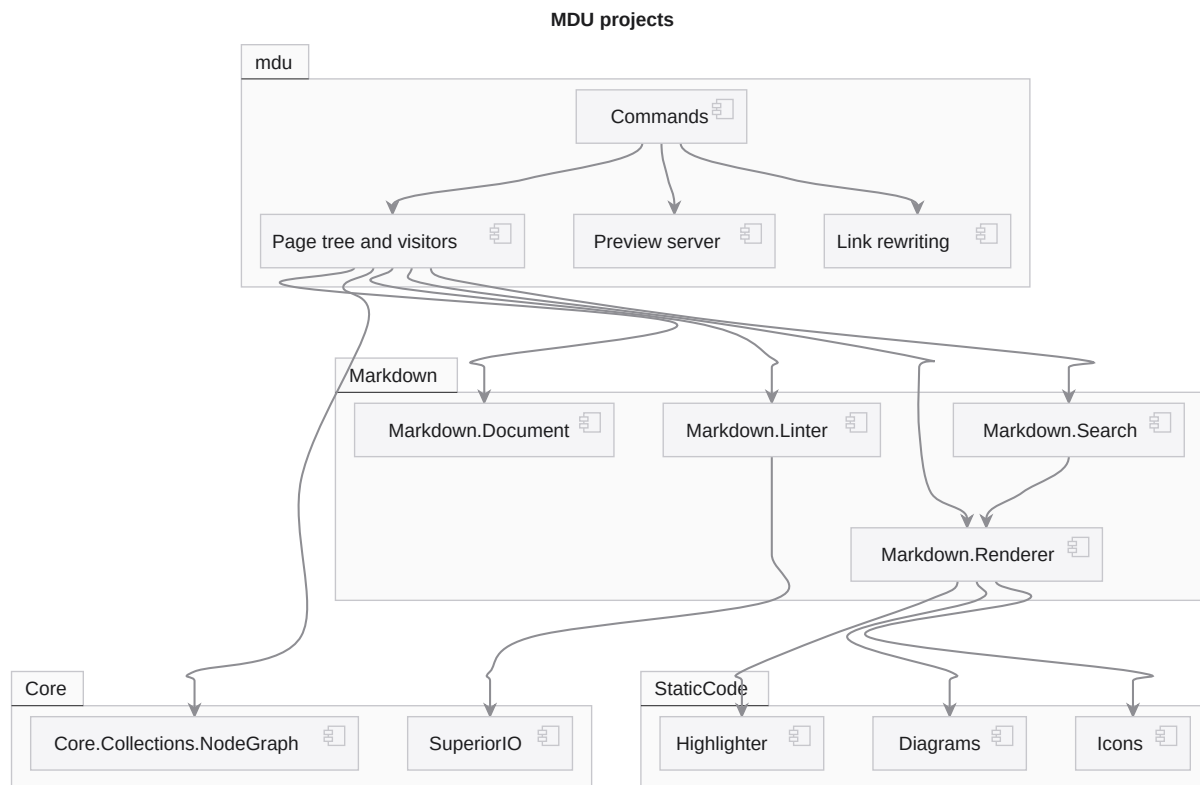
MDU is one solution, `Mdu.sln`, with one executable project and several libraries. All projects target .NET 10, and each library has a matching `.Test` project with xUnit tests.

Overview

| Project | Responsibility | Depends on |
|----------------------------|--|--|
| mdu | mdu command line, page tree, orchestration | all libraries below |
| Markdown.Document | Reads titles, front matter, and links of a page | Markdig, Cap.Core |
| Markdown.Linter | Validators for titles, link casing, and link targets | SuperiorIO, FluentValidation |
| Markdown.Renderer | HTML and PDF rendering, including OpenAPI references | StaticCode.*, Markdig, Microsoft.OpenApi |
| Markdown.Search | Static search index and search script | Markdown.Renderer |
| StaticCode.Highlighter | Syntax highlighting for code blocks | - |
| StaticCode.Diagrams | Mermaid, PlantUML, and context diagrams as SVG | - |
| StaticCode.Icons | Embedded icon sets for shortcodes | - |
| Core.Collections.NodeGraph | Tree and visitor abstractions | - |
| SuperiorIO | Path parsing helpers | - |

Project dependencies

This diagram shows how the solution projects depend on one another. Select a project to jump to its description.



mdu

The packaged .NET tool `mdu` with the command name `mdu`.

- `Program.cs` defines all commands, aliases, and options with `System.CommandLine` and registers the services with `Microsoft.Extensions.DependencyInjection`.
- `Commands/` holds one class per command: `LintCommand`, `FixCommand`, `PrintTreeCommand`, `RenderCommand`, `PdfCommand`, `ServeCommand`, and `PageCommand`.
- `GraphBuilder` and the visitors in `Service/Visitors` build and walk the [page tree](#).
- `PreviewServer` serves the rendered site for `mdu serve` and pushes reload events to the browser.
- `MarkdownLinks` finds and rewrites relative Markdown links. `mdu fix`, `mdu page move` (through `PageMover`), and `mdu page links` share it.
- `MarkdownCompatibility` detects MkDocs Material syntax and converts it for `mdu fix`.
- `SiteConfigurationReader` reads `mdu.yml`, `GitLastModified` reads page dates from git, and `LintResultRenderer` prints lint results as text, JSON, or GitHub annotations.

Markdown.Document

Reads what MDU needs to know about a page without rendering it: the YAML front matter, the level-one title, and the links. Links include images, reference definitions, and node links in Mermaid, context, and PlantUML diagrams, so the linter checks them all.

Markdown.Linter

Contains the rule codes `MDU001`-`MDU018` and the validators that need only one page and the file system:

| Validator | Rule |
|-----------|------|
|-----------|------|

| | |
|----------------|----------------------------|
| Rule2Validator | MDU002 DirectoryTitleMatch |
| Rule3Validator | MDU003 LinkCasing |
| Rule4Validator | MDU004 LocalLinkExists |

The validators use `FluentValidation`. Rules that need the whole page tree, such as duplicate pages, navigation placement, and MkDocs migration issues, are checked by the `Lintervisitor` in `mdu`. See the [rules reference](#) for every rule.

Markdown.Renderer

Turns Markdown into pages. `HtmlRenderer` holds the Markdig pipeline and the embedded HTML template, stylesheet, and scripts; `OpenApiMarkdownRenderer` reads referenced JSON or YAML specifications and appends their API reference as Markdown; `PdfRenderer` lays out the same content on A4 pages. [Rendering](#) describes both in detail.

Markdown.Search

`SearchIndexBuilder` splits every page into sections at its headings and writes their plain text to `search-index.json`. `SearchClientAsset` writes `js/search.js`, which searches that index in the browser and links to the matching section.

StaticCode.Highlighter

A small lexer with lightweight grammars for C#, JavaScript, Swift, TypeScript, Python, Kotlin, JSON, SQL, HTML, CSS, Java, Rust, Go, YAML, shell, and Markdown. `HtmlFormatter` wraps the tokens in `span` elements whose classes the stylesheet colors. Other languages stay readable as plain code.

StaticCode.Diagrams

Parses diagram sources and writes SVG. `DiagramRenderer.TryRenderSvg` handles Mermaid flowcharts, sequence and class diagrams, PlantUML component diagrams, and MDU's own context diagrams.

- Flowcharts use their own layered (Sugiyama) layout in `Layout/LayeredLayout.cs`.
- PlantUML containers are laid out from the inside out, so nested packages size to their content.
- Context diagrams place the central system in the middle and its neighbors around it.
- The SVG carries only CSS classes and no colors; the site stylesheet themes it for light and dark mode.

When a source uses unsupported syntax, `TryRenderSvg` returns `false` and the block stays a code block.

StaticCode.Icons

Embeds the icon sets behind the [icon shortcodes](#) as gzip-compressed, reduced Iconify JSON. `IconCatalog` resolves a shortcode such as `:material-rocket:` to its SVG. The script `update-icon-sets.sh` regenerates the sets with `npm`, and `dashboard-icons.mjs` sanitizes the colored Dashboard Icons with `SVGO`.

Core.Collections.NodeGraph

A general tree: every node has an optional parent and any number of children. `IVisitor<T>` and the `Visit` extension walk the tree depth-first or breadth-first. Each command is a set of visitors over the page tree.

SuperiorIO

Parses paths into elements so the linter can compare a link target with the file-system entry, including its casing.

Third-party libraries

| Library | Used for | Project |
|--|---|----------------------------|
| Markdig | CommonMark parsing and HTML output | Document, Renderer, Search |
| System.CommandLine | Commands, options, and help | mdu |
| Microsoft.Extensions.DependencyInjection | Service registration | mdu |
| YamlDotNet | Front matter, mdu.yml, and hero options | mdu, Renderer |
| FluentValidation | Lint validators | Linter |
| CSharpMath and VectSharp | Math formulas as SVG | Renderer |
| Cap.Core | Parser and text helpers | mdu, Document, Highlighter |

Note

The rendered site loads none of these libraries; they run only while MDU works. The notices for the bundled icon sets and the math renderer ship in `THIRD-PARTY-NOTICES.md` inside the package.

Processing Pipeline

Most commands follow the same pattern: find the Markdown files, build a page tree, and walk it with one or more visitors. The visitors decide what a command does.

Page tree

`GraphBuilder` turns the documentation folder into a tree of `MarkdownFile` nodes:

1. It lists the files that match the search pattern, `*.md` by default, folder by folder.
2. A folder's `README.md`, or else its `index.md`, becomes the parent of the other pages in that folder and of the pages in its subfolders.
3. A folder without such a page adds its pages to the nearest parent above it.
4. `DataPreparationVisitor` reads each page's level-one heading and front matter: `title`, `description`, `icon`, `openapi`, `mdu.import.source`, and the `navigation.*` settings.
5. `GraphBuilder` resolves GitHub imports and mounts each imported folder's page tree at the import

page's position. Each imported page keeps its repository source path and its site-relative path.

```
docs/
??? README.md      -> root page
??? Setup/
?   ??? README.md   -> child of README.md, parent of the pages below
?   ??? DotNet-Tool-Installation.md
?   ??? Portable-Installation.md
??? Usage/
    ??? README.md    -> child of README.md, parent of the pages below
    ??? render.md
    ??? lint/
        ??? README.md -> child of Usage/README.md
        ??? Rules.md
```

The tree is the single source for the sidebar, the top navigation, breadcrumbs, previous and next links, and the page order of the PDF. `mdu tree` prints it.

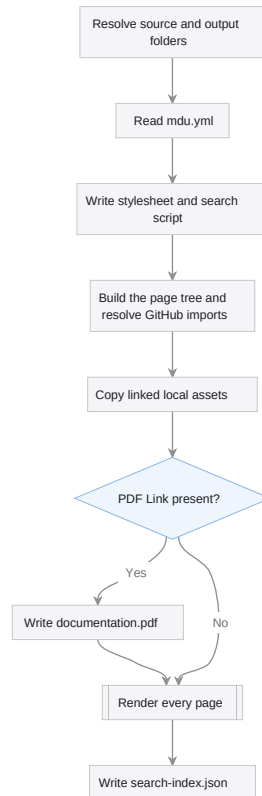
Commands and visitors

| Command | Walks the tree with | Result |
|---------|---|--|
| tree | PrintTreeVisitor | Indented page titles |
| lint | LintVisitor, then LintResultRenderer | Diagnostics as text, JSON, or GitHub annotations |
| fix | LintVisitor, then MarkdownCompatibility and MarkdownLinks | Converted syntax, moved pages, and updated links |
| render | LocalAssetsCopy, PdfCommand, MarkdigVisitor | Static site and, when a PDF Link is present, documentation.pdf |
| serve | Same as render, then a file watcher | Preview site and a refreshed PDF when a PDF Link is present |
| pdf | The tree in navigation order, then PdfRenderer | One PDF file |

`mdu page move` and `mdu page links` need no page tree. They read every Markdown file below the documentation folder with `MarkdownLinks`, so they also see pages that are hidden from navigation.

Rendering a site

`mdu render` runs these steps once and exits. Click a step to read more.



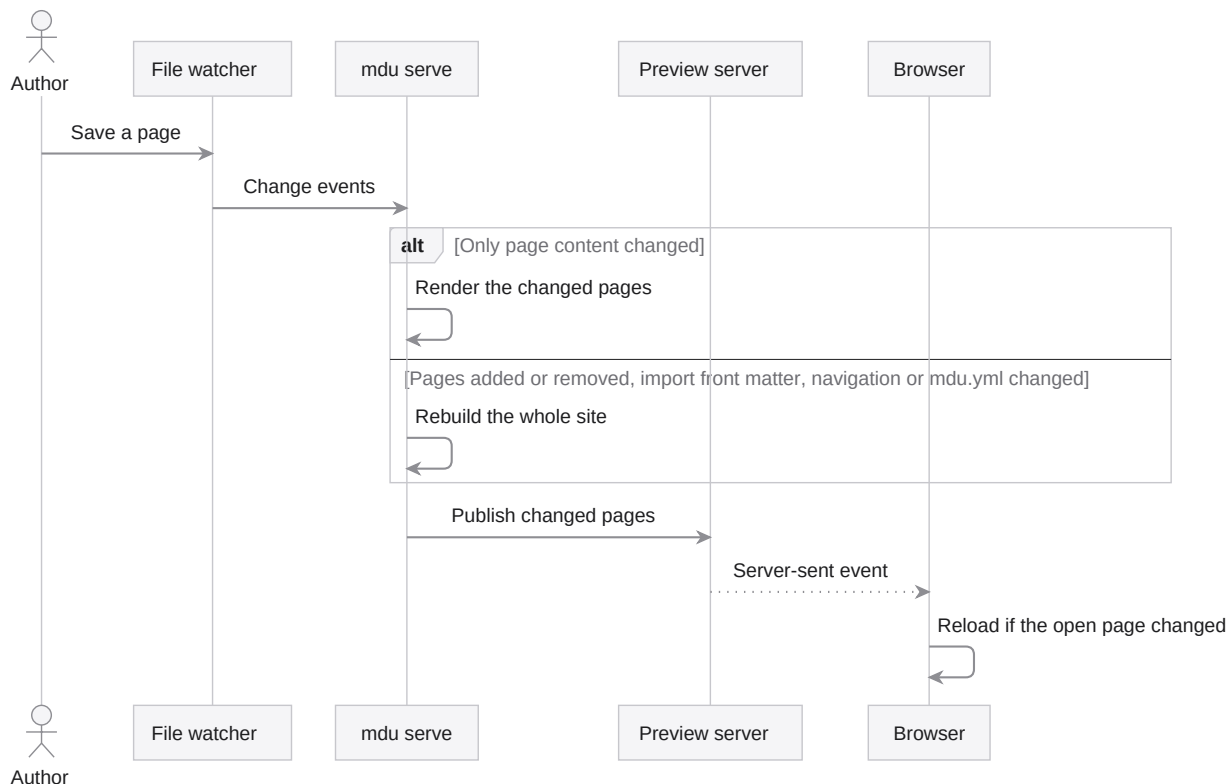
For each page, `MarkdigVisitor`:

- removes the front matter, appends any generated OpenAPI reference, and adds the page text, split at its headings, to the search index;
- collects the navigation, top navigation, breadcrumbs, and last-changed date from the tree and git;
- lets `HtmlRenderer` write the page to a temporary file and then moves it into place, so a browser never reads a half-written page;
- writes `README.md` as `index.html`, and adds a small redirect page so that `Setup/` also reaches `Setup.html`.

Assets are found through the links on each page. Only linked local files are copied, with their relative paths unchanged; external URLs stay where they are.

Live preview

`mdx serve` renders the site once, starts a local HTTP server, and watches the source folder. Changes are collected for a short moment and handled as one batch:



`serve` remembers which assets each page links to. When an image changes, it copies the new file and reloads only the pages that show it. Changes to an OpenAPI specification rebuild the site so its generated reference stays current. A changed import page triggers a rebuild and resolves its GitHub URL again; restarting `serve` picks up remote branch changes when no local import page changed. Files that disappear from the site are removed from the output folder.

When Markdown content changes, `serve` refreshes `documentation.pdf` if a PDF Link remains in the project. Adding the first link creates the file; removing the last link deletes it.

PDF export

`mdu pdf` builds the same page tree and orders it like the navigation. `PdfRenderer` renders every page with the HTML pipeline, lays out the resulting elements on A4 pages, and writes the PDF file itself. A folder export adds a title page and a linked table of contents. See [Rendering](#) for the details.

Linting and fixing

`LintVisitor` checks each page in the tree. It reads the page's links with `Markdown.Document` and runs the [validators](#) for titles, link casing, and link targets. It checks rules that depend on the tree itself, such as duplicate pages, section pages, and navigation placement. The results go to a receiver, which `LintResultRenderer` then prints in the chosen format.

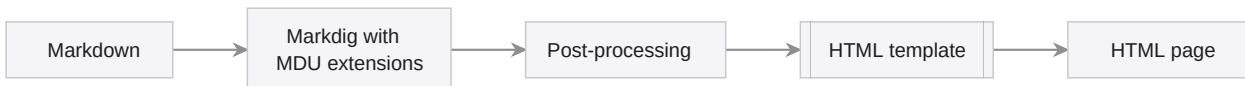
`mdu fix` runs the same linter, then applies what it can fix: it converts MkDocs Material syntax, renames `index.md` pages, creates missing section pages, and moves pages flagged by MDU006. Whenever a page moves, `MarkdownLinks` rewrites the links to it and the relative links inside it.

Rendering

Markdown.Renderer turns the Markdown of one page into a complete HTML page, or into PDF content. The page tree supplies everything around the content, such as navigation and breadcrumbs; the renderer supplies the content itself and the page frame.

Pages with an `openapi` front matter entry have a generated API reference appended to their Markdown before rendering. `OpenApiMarkdownRenderer` reads the local JSON or YAML file, then the normal HTML and PDF renderers process the combined content. Large specifications repeat the same examples and schemas across many responses, so the HTML renderer emits a tab content that occurs more than once only once as a `<template>`. The page script copies it into an Example, Schema, or Schema JSON tab when that tab becomes visible in an open endpoint, and restores the schema anchor prefix of each media type in the copy. See the [OpenAPI feature](#) for its usage and supported content.

From Markdown to HTML



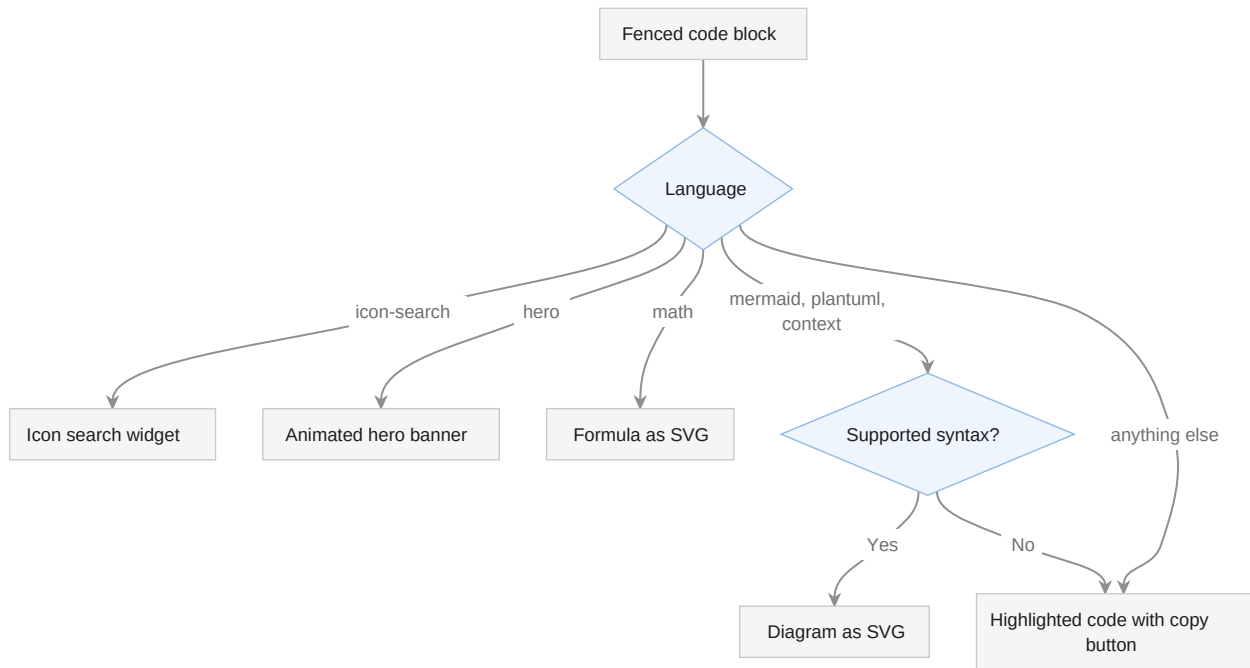
1. **Parse.** Markdig parses the page with the extensions listed below.
2. **Rewrite links.** Relative links to `.md` files point to the rendered `.html` pages; `README.md` becomes `index.html`. Anchors and query strings are kept.
3. **Render.** Markdig writes the HTML. Code blocks go through MDU's own [code block renderer](#).
4. **Post-process.** Standalone video links become consent placeholders, a full-width hero moves above the page layout, headings get permalinks, and the *On this page* list is collected.
5. **Fill the template.** The embedded `main.html` contains placeholders such as `%%MARKDOWN%%`, `%%NAVIGATION_ITEMS%%`, and `%%PAGE_BREADCRUMBS%%`. `HtmlRenderer` replaces them with HTML-encoded values; there is no separate template engine.

Markdig extensions

| Extension | Adds |
|-----------------------------|---|
| Markdig advanced extensions | Pipe and grid tables, footnotes, task lists, emphasis extras, heading IDs, automatic links, and attribute lists |
| ScrollableTableExtension | Wraps wide tables in a horizontally scrollable container |
| RenderedMathExtension | Math as inline SVG |
| GitHubAlertExtension | Callouts with GitHub's alert syntax |
| ShortcodeExtension | Icons and emojis from shortcodes |
| KeysExtension | Keyboard keys such as <code>Ctrl+Alt+Delete</code> |
| HighlighterExtension | The code block renderer below |

Code blocks

Every fenced code block passes through one renderer, which picks the output by the block's language:



Because a diagram that cannot be rendered stays a code block, a page never breaks on unknown syntax. `mdu lint` reports unsupported Mermaid syntax as `MDU010`.

Page assets

Every page links the shared `css/main.css`. Everything else is added only to the pages that need it:

| Asset | Added when |
|--|--|
| <code>js/search.js</code> and <code>search-index.json</code> | Always; the search runs in the browser without a service |
| Icon search script and index in <code>js/</code> | A page contains an icon-search block |
| Hero styles and WebGL scene, inline | A page contains a hero block |
| Media consent script, inline | A page contains a standalone YouTube or Vimeo link |
| Logos and favicons from <code>mdu.yml</code> | They are configured |

A rendered site looks like this:

```

site/
??? index.html
??? search-index.json
??? css/
?   ??? main.css
??? is/
?   ??? search.js
??? assets/
?   ??? logo.svg
??? Usage/
    ??? index.html
    ??? render.html
  
```

In the browser

The generated pages work without a build step or framework. Small inline scripts add:

- light and dark themes, with an automatic mode that follows the operating system;

- the collapsible sidebar, top navigation tabs, and a drawer on narrow screens;
- client-side search with links to the matching section and highlighted hits;
- copy buttons for code blocks, image lightboxes, and footnote previews;
- the consent dialog for embedded videos.

Theme colors from `mdu.yml` become CSS custom properties. Secondary colors, lines, and hover backgrounds are mixed from them in CSS, so a custom palette stays consistent.

PDF output

`PdfRenderer` reuses the HTML pipeline for `mdu pdf` and for the `documentation.pdf` generated by `mdu render` and `mdu serve` when the project contains a PDF Link. It renders every page to an HTML fragment, reads that fragment as XML, and lays out its blocks on A4 pages. Diagrams and formulas are drawn from their SVG paths. The writer uses the fonts built into every PDF reader, so it embeds no font files; emoji shortcodes therefore stay text. Links between included pages become internal PDF links.

Other Tools

MDU focuses on one job: publishing a folder of plain Markdown with a single .NET tool and no theme or plugin setup. If you need a plugin ecosystem, custom components, versioned documentation, or translations, one of these generators may suit you better.

| Tool | Built on | Typical use |
|---------------------|----------------|---|
| Docusaurus | React, Node.js | Product documentation with MDX components, versions, and translations |
| Material for MkDocs | MkDocs, Python | Feature-rich MkDocs theme; end of life on May 5, 2027 |
| MkDocs | Python | Simple project documentation configured in one YAML file |
| Rspress | React, Rsbuild | Fast documentation sites with MDX |
| Starlight | Astro | Documentation sites that mix Markdown with Astro components |
| VuePress | Vue | Documentation with Vue components in Markdown |
| Zensical | Python, Rust | Successor to Material for MkDocs from the same team |

Tip

Coming from Material for MkDocs? MDU understands the same icon shortcodes, and `mdu fix` converts admonitions, tabs, card grids, and `.pages` files to portable Markdown.