Table of Contents

aidocs

AI-powered documentation generator for web applications.

How It Works

aidocs generates comprehensive documentation by combining **three sources of truth**:

1. **Vision Analysis** - Playwright captures screenshots, Claude analyzes what users actually see

2. **Codebase Analysis** - Scans your frontend components, backend routes, validation rules, and models

3. **Interactive Exploration** - Clicks buttons, fills forms, discovers conditional UI and validation messages

This produces documentation that's accurate to both the code AND the actual user experience.

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│ 📸 Screenshots │ +  │ 📄 Code Analysis │ +  │ 🖱️ UI Testing   │
│ (what users     │    │ (validation,    │    │ (conditional    │
│  see)           │    │  routes, models) │    │  fields, flows)│
└────────┬────────┘    └────────┬────────┘    └────────┬────────┘
         │                      │                      │
         └──────────────────────┼──────────────────────┘
                                ▼
                    ┌───────────────────────┐
                    │ 📚 Smart Documentation │
                    │ that stays in sync    │
                    └───────────────────────┘
```

Installation

```
# Install from PyPI
uv tool install aidocs

# Or install from GitHub
uv tool install aidocs --from git+https://github.com/binarcode/aidocs-cli.git

# Or use pipx
pipx install aidocs
```

Updating

When a new version is released, update the CLI and reinstall commands in your project:

```
# 1. Update the CLI
aidocs update

# 2. Reinstall commands in your project (adds new slash commands)
cd your-project
aidocs init . --force
```

The `--force` flag overwrites existing command files, adding any new commands from the latest version.

Tip: Run `aidocs update --github` to get the latest unreleased features from GitHub.

Quick Start

```
# Install the CLI
uv tool install aidocs

# Add to your project
aidocs init .
```

Usage Flow

```
┌───────────────────────────────────────────────────────────────────────────────┐
│                               SETUP (once)                                    │
├───────────────────────────────────────────────────────────────────────────────┤
│                                                                               │
│  $ aidocs init .                   Install commands into project              │
│           │                                                                   │
│           ▼                                                                   │
│  /docs:init                        Configure: name, auth, style, output       │
│                                                                               │
└───────────────────────────────────────────────────────────────────────────────┘
                                      │
                                      ▼
┌───────────────────────────────────────────────────────────────────────────────┐
│                     OPTION A: Document a Single Page                          │
├───────────────────────────────────────────────────────────────────────────────┤
│                                                                               │
│  /docs:generate https://myapp.com/users    ← Give it any URL!                 │
│           │                                                                │
│           ├──→ Takes screenshots with Playwright                           │
│           ├──→ Analyzes codebase for that route                            │
│           ├──→ Documents UI elements and interactions                      │
│           └──→ Creates docs/users/index.md                                 │
│                                                                            │
└────────────────────────────────────────────────────────────────────────────┘
                                     │
                                     ▼
┌────────────────────────────────────────────────────────────────────────────┐
│                    OPTION B: Document a Code Flow                          │
├────────────────────────────────────────────────────────────────────────────┤
│                                                                            │
│  /docs:flow "sync users from discord"    ← Describe the flow in words!     │
│           │                                                                │
│           ├──→ Searches codebase for relevant files                        │
│           ├──→ Traces execution path and builds call graph                 │
│           ├──→ Generates mermaid sequence diagram                          │
│           ├──→ Captures UI screenshots (if Playwright + route detected)    │
│           └──→ Creates docs/flows/sync-users-from-discord.md               │
│                                                                            │
└────────────────────────────────────────────────────────────────────────────┘
                                     │
                                     ▼
┌────────────────────────────────────────────────────────────────────────────┐
│                    OPTION C: Document Entire Project                       │
├────────────────────────────────────────────────────────────────────────────┤
│                                                                            │
│  /docs:discover                  Scan codebase, find all modules           │
│           │                                                                │
│           ▼                                                                │
│  /docs:plan                      Create ordered documentation plan         │
│           │                      → Outputs docs/plan.yml                  │
│           ▼                                                                │
│  /docs:execute                   Run through plan, generate all docs       │
│                                  → Resume with --continue if interrupted   │
│                                                                            │
└────────────────────────────────────────────────────────────────────────────┘
                                     │
                                     ▼
┌────────────────────────────────────────────────────────────────────────────┐
│                         KEEP DOCS IN SYNC                                  │
├────────────────────────────────────────────────────────────────────────────┤
│                                                                            │
│  # After implementing a feature:                                           │
│  /docs:update --base main        Detect changes, update affected docs      │
│                                                                            │
└────────────────────────────────────────────────────────────────────────────┘
                                     │
                                     ▼
```

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                         ENABLE SEMANTIC SEARCH (optional)                    │
├──────────────────────────────────────────────────────────────────────────────┤
│                                                                              │
│  # After docs are generated, setup RAG for AI-powered search:                │
│  /docs:rag                         ← One command does it all!                │
│           │                                                                  │
│           ├──→ Chunks your docs into searchable pieces                       │
│           ├──→ Creates database migration (pgvector)                         │
│           ├──→ Generates OpenAI embeddings                                   │
│           └──→ Outputs sync.sql ready to import                              │
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘
```

Quick Commands

```
# Simple: Generate docs for one page
/docs:generate https://myapp.com/dashboard

# Flow: Document a feature (user-focused by default)
/docs:flow "how to create employees"
/docs:flow "import payments" --technical    # Developer docs

# Batch: Document entire project
/docs:discover && /docs:plan && /docs:execute

# Maintain: Update after code changes
/docs:update --base main

# RAG: Setup semantic search for your docs
/docs:rag
```

CLI Commands

`aidocs init [PROJECT_NAME]`

Initialize the docs module in a project.

```
aidocs init .                  # Current directory
aidocs init my-project         # New directory
aidocs init . --force          # Overwrite existing
aidocs init . --ai cursor      # Use with Cursor
```

Options:

Option	Description
`--ai`	AI assistant: `claude`, `cursor`, `copilot` (default: `claude`)
`--force, -f`	Overwrite existing files
`--no-git`	Skip git initialization

`aidocs check`

Check for required tools and dependencies.

```
aidocs check
```

`aidocs version`

Show version information.

`aidocs update`

Update aidocs to the latest version.

```
aidocs update              # Update from PyPI
aidocs update --github     # Update from GitHub (latest)
```

Options:

Option	Description
`--github`	Install latest from GitHub instead of PyPI

Automatically detects and uses the appropriate package manager (uv, pipx, or pip).

`aidocs rag-chunks`

Chunk markdown files for vector database import.

```
aidocs rag-chunks                    # Chunk all files in docs/
aidocs rag-chunks docs/users         # Chunk specific directory
aidocs rag-chunks --force            # Re-chunk all files
aidocs rag-chunks --dry              # Preview only
```

Options:

Option	Description
`--force, -f`	Re-chunk all files (ignore cache)
`--dry`	Preview without writing files

What it does:

1. Scans directory for `.md` files
2. Splits at `##` headings into chunks
3. Creates `.chunks.json` files alongside each `.md`
4. Maintains `docs/.chunks/manifest.json` for change tracking

Output structure:

```
docs/
├── users/
|   ├── lifecycle.md
|   └── lifecycle.chunks.json    # Chunks for this file
├── campaigns/
|   ├── lifecycle.md
|   └── lifecycle.chunks.json
└── .chunks/
    └── manifest.json            # Tracking file
```

Next step: Run `aidocs rag-vectors` to generate embeddings

`aidocs rag-vectors`

Generate embeddings and SQL for vector database import.

```
aidocs rag-vectors                   # Generate embeddings and SQL
aidocs rag-vectors --dry             # Preview what would be synced
aidocs rag-vectors --force          # Re-sync all files
aidocs rag-vectors --table my_docs  # Custom table name
```

Options:

Option	Description
`--force, -f`	Re-sync all files (ignore last sync)
`--dry`	Preview without generating embeddings
`--table, -t`	Target table name (default: `doc_embeddings`)

Requires: `OPENAI_API_KEY` environment variable

What it does:

1. Reads chunk files from `docs/.chunks/`

2. Calls OpenAI API to generate embeddings (text-embedding-3-small)

3. Creates `docs/.chunks/sync.sql` with INSERT statements

4. Tracks sync state to avoid re-processing unchanged files

Output: `docs/.chunks/sync.sql`

```
BEGIN;
INSERT INTO doc_embeddings (file_path, content, chunk_index, title, metadata,
embedding)
VALUES ('docs/users/lifecycle.md', '...', 0, 'Overview', '{...}'::jsonb, '[0.001,
...]'::vector);
-- ... more inserts
COMMIT;
```

Import to database:

```
psql $DATABASE_URL -f docs/.chunks/sync.sql
```

Slash Commands

After running `aidocs init`, these commands are available in Claude Code:

Command	Description	Requires Playwright
`/docs:init`	Configure project settings, credentials, output style	No
`/docs:generate `	Generate docs for a single page with screenshots	Yes
`/docs:analyze `	Analyze codebase for a route (no browser)	No
`/docs:batch`	Generate docs for multiple pages	Yes
`/docs:update`	Update docs based on git diff	Optional
`/docs:discover`	Scan codebase, discover all modules	No
`/docs:plan`	Create ordered documentation plan	No
`/docs:execute`	Execute plan, generate all docs	Yes
`/docs:explore `	Interactive UI exploration with Playwright	Yes
`/docs:flow ""`	Document a feature with screenshots (use `--technical` for dev docs)	Optional
`/docs:rag-vectors`	Generate embeddings and SQL for vector DB import	No
`/docs:rag-init`	Generate database migration for vector embeddings	No
`/docs:rag`	Setup RAG: chunks → migration → embeddings (all-in-one)	No
`/docs:export-pdf`	Export markdown documentation to PDF with TOC	Yes (Playwright)

`/docs:init`

Interactive setup wizard that:

- Detects your tech stack (Laravel, Vue, React, Next.js, etc.)
- Asks for project name, audience, and documentation tone
- Configures authentication method (file, env vars, or manual)
- Sets output directory and screenshot preferences

`/docs:generate `

Generate documentation for a single page:

```
/docs:generate https://myapp.com/campaigns
/docs:generate /campaigns                      # Uses base URL from config
/docs:generate /settings --auth user:pass      # With authentication
```

Features:

- Captures full-page screenshots
- Analyzes UI elements visually
- Searches codebase for related code
- Detects forms, buttons, and interactive elements
- Offers to document user flows step-by-step

`/docs:update`

Update existing documentation based on code changes:

```
/docs:update                    # Compare against main
/docs:update --base staging     # Compare against staging branch
/docs:update --dry-run          # Preview changes without applying
/docs:update --screenshots      # Also refresh screenshots
```

What it does:

1. Gets git diff between current branch and base

2. Analyzes changed frontend/backend files

3. Maps code changes to affected features

4. Finds and updates related documentation

5. Optionally refreshes screenshots

6. Offers to stage/commit doc changes

Perfect for: Running before creating a PR to ensure docs stay in sync with code.

`/docs:analyze `

Analyze codebase without browser automation:

```
/docs:analyze /campaigns
/docs:analyze /api/users
```

`/docs:batch`

Generate documentation for multiple pages:

```
/docs:batch urls.txt                          # From file
/docs:batch --discover --base-url https://myapp.com  # Auto-discover routes
```

`/docs:discover`

Scan your codebase to discover all modules and their structure:

```
/docs:discover                     # Discover all modules
/docs:discover --dry               # Preview without saving
/docs:discover campaigns           # Analyze only one module
```

What it analyzes:

- Backend: Models, controllers, routes, validation rules
- Frontend: Pages, components, forms, state management
- Relationships: Foreign keys, ORM relationships, cross-module navigation

Creates `docs/.knowledge/` with:

```
docs/.knowledge/
├── _meta/
│   ├── project.json              # Project-level info
│   └── modules-index.json        # List of discovered modules
├── modules/
│   ├── campaigns/
│   │   ├── entity.json           # Fields, types, relationships
│   │   ├── routes.json           # API endpoints
│   │   ├── components.json       # UI components
│   │   └── validation.json       # Validation rules
│   └── users/
│       └── ...
└── relationships/                # Cross-module relationships
```

Next step: Run `/docs:plan` to create documentation plan

`/docs:plan`

Create an ordered documentation plan based on discovered modules:

```
/docs:plan                          # Create plan interactively
/docs:plan --auto                   # Auto-generate plan (no prompts)
/docs:plan --show                   # Show existing plan
```

What it does:

1. Reads discovered modules from `docs/.knowledge/`

2. Analyzes dependencies and relationships

3. Suggests documentation order (core modules first)

4. Creates `docs/plan.yml` with the plan

Output: `docs/plan.yml`

```
modules:
  - name: users
    priority: 1
    reason: "Core module - other modules depend on it"
    document:
      lifecycle: true
      include_errors: true
    status: pending

  - name: campaigns
    priority: 2
    document:
      lifecycle: true
      flows:
        - "duplicate campaign"
    status: pending

cross_module_flows:
  - name: "user registration to first campaign"
    modules: [users, campaigns]
    status: pending
```

Next step: Run `/docs:execute` to generate documentation

`/docs:execute`

Execute the documentation plan and generate all docs:

```
/docs:execute                      # Execute full plan
/docs:execute --module campaigns   # Execute only one module
/docs:execute --continue           # Continue from where it stopped
/docs:execute --dry                # Preview what would be generated
```

What it does:

1. Reads `docs/plan.yml`

2. For each module in order:

- Runs explore (if needed)

- Generates lifecycle documentation

- Captures screenshots

- Writes to `docs/{module}/`

3. Updates plan status as it progresses

4. Generates cross-module flows last

Output structure:

```
docs/
├── index.md                    # Auto-generated with links
├── users/
|   ├── index.md               # Module overview
|   ├── lifecycle.md           # CRUD documentation
|   ├── user-registration-to-campaign.md  # Cross-module flow (first module)
|   └── images/
└── campaigns/
    ├── index.md
    ├── lifecycle.md
    ├── duplicate-campaign.md  # Custom flow
    └── images/
```

Resume support: If execution stops, run `/docs:execute --continue` to resume

`/docs:explore `

Interactively explore a module's UI with Playwright:

```
/docs:explore campaigns                    # Explore all campaign pages
/docs:explore users --page /users/create   # Specific page
/docs:explore orders --depth deep          # Thorough exploration
```

What it discovers:

- Conditional fields (checkbox reveals more inputs)
- Validation messages (tries invalid data)
- UI state changes (what happens when you click)
- Cross-page effects (create here → appears there)

`/docs:flow ""`

Document a feature with screenshots and step-by-step instructions. By default, creates **user-focused** documentation. Use `--technical` for developer documentation.

```
/docs:flow "how to create employees"              # User guide with screenshots
/docs:flow "import payments from csv"             # User guide with screenshots
/docs:flow "payment processing" --technical       # Developer docs with code
/docs:flow "stripe webhooks" --technical          # Developer docs with code
/docs:flow "user registration" --no-screenshots   # Skip screenshots
```

Arguments:

- `--technical` - Generate developer-focused documentation with code snippets
- `--no-screenshots` - Skip UI screenshot capture

Output modes:

Mode	Audience	Output
Default	End users	Screenshots, plain English, step-by-step guide
`--technical`	Developers	Code snippets, file paths, mermaid diagrams

Output: `docs/flows/{kebab-case-title}.md`

Example: User-focused (default)

```
# How to Import Payments

Import payment records from a CSV file.

## Before You Start
- Prepare a CSV with columns: date, amount, description
- Maximum 10,000 rows per import

## Steps

### Step 1: Go to Payroll
Navigate to **Payroll** from the sidebar.

![Payroll Page](./images/payroll-page.png)

### Step 2: Click Import
Click the **Import Payments** button.

![Import Button](./images/import-button.png)

### Step 3: Upload Your File
Select your CSV file and click **Start Import**.

## What Happens Next
- Import runs in background
- You'll receive an email when complete
```

Example: Technical (`--technical`)

```
# Import Payments Flow

## Architecture
sequenceDiagram: User → Controller → Job → Database

## Entry Points
| Trigger | Route |
|---------|-------|
| UI | POST /payroll/import |
| CLI | php artisan payments:import |

## Execution Flow

**File:** `app/Http/Controllers/PayrollController.php:45`
public function import(Request $request) { ... }

**File:** `app/Jobs/ImportPaymentsJob.php:28`
public function handle() { ... }
```

Screenshots require:

- Playwright MCP installed
- `urls.base` configured in `docs/config.yml`

`/docs:rag-vectors`

Generate embeddings and SQL for syncing documentation to a PostgreSQL vector database.

```
/docs:rag-vectors                      # Generate sync SQL (smart)
/docs:rag-vectors --dry                # Preview what would be synced
/docs:rag-vectors --force              # Re-sync all files
```

Prerequisites:

- Run `aidocs rag-chunks` first to create chunk files
- Set `OPENAI_API_KEY` environment variable

What it does:

1. Reads chunk files from `docs/.chunks/manifest.json`

2. Compares against last sync to find changes

3. Generates embeddings via OpenAI API (only for new/changed chunks)

4. Creates `docs/.chunks/sync.sql` with INSERT statements

Smart sync:

- Unchanged files → Skip (no API calls)
- Changed files → Re-generate embeddings
- New files → Generate embeddings
- Deleted files → Add DELETE statements

Output:

```
📊 Sync Summary:
   Unchanged: 12 files (skipped)
   Changed: 2 files (8 chunks)
   New: 1 file (3 chunks)

📄 Generated: docs/.chunks/sync.sql
```

```
Run with:
   psql $DATABASE_URL -f docs/.chunks/sync.sql
```

`/docs:rag-init`

Generate a database migration for storing documentation embeddings with pgvector.

```
/docs:rag-init                     # Default: 1536 dimensions
/docs:rag-init --dimensions 3072   # For text-embedding-3-large
/docs:rag-init --table my_docs     # Custom table name
```

What it does:

1. Detects your framework (Laravel, Prisma, TypeORM, Drizzle, Django)

2. Generates the appropriate migration file

3. Creates table with pgvector support for similarity search

Supported Frameworks:

Framework	Detection	Output
Laravel	`composer.json`	PHP migration with `$table->vector()`
Prisma	`schema.prisma`	Prisma schema addition
TypeORM	`package.json`	TypeScript migration class
Drizzle	`drizzle-orm`	Schema + SQL migration
Django	`manage.py`	Django migration with pgvector
Fallback	None detected	Raw PostgreSQL SQL

Table Structure:

```
doc_embeddings
├── id              UUID PRIMARY KEY
├── file_path       VARCHAR(500)      # Path to .md file
├── content         TEXT              # Document content
├── chunk_index     INTEGER           # For large docs split into chunks
├── title           VARCHAR(255)      # Document title
```

```
 ├── metadata       JSONB             # Tags, module, category, etc.
 ├── embedding      VECTOR(1536)      # OpenAI embedding
 ├── created_at     TIMESTAMP
 └── updated_at     TIMESTAMP
```

Indexes:

- `file_path` - B-tree index for path lookups
- `embedding` - HNSW index for fast vector similarity search

Requirements:

- PostgreSQL with [pgvector](https://github.com/pgvector/pgvector) extension

Example workflow:

```
# 1. Generate migration
/docs:rag-init

# 2. Run migration
php artisan migrate          # Laravel
npx prisma migrate dev       # Prisma
python manage.py migrate     # Django

# 3. Chunk your docs
aidocs rag-chunks

# 4. Generate embeddings and sync
aidocs rag-vectors
```

`/docs:rag`

The easy way - Setup RAG (Retrieval Augmented Generation) for your documentation in one command:

```
/docs:rag                        # Full setup
/docs:rag --skip-migration       # Skip migration (table already exists)
/docs:rag --force                # Re-chunk and re-sync everything
/docs:rag --dry                  # Preview what would happen
```

What it does automatically:

1. Checks/creates documentation chunks (`aidocs rag-chunks`)

2. Generates database migration (`/docs:rag-init`)

3. Prompts you to run the migration

4. Generates embeddings and SQL (`aidocs rag-vectors`)

Output:

```
✅ RAG Setup Complete!

📊 Summary:
   Documentation files: 8
   Chunks created: 24
   Embeddings generated: 24

📄 Files created:
   ✓ docs/.chunks/manifest.json
   ✓ database/migrations/..._create_doc_embeddings_table.php
   ✓ docs/.chunks/sync.sql

🚀 Final step:
   psql $DATABASE_URL -f docs/.chunks/sync.sql
```

Requirements:

- PostgreSQL with [pgvector](https://github.com/pgvector/pgvector) extension
- `OPENAI_API_KEY` environment variable

`/docs:export-pdf`

Export markdown documentation to PDF with auto-generated table of contents using Playwright MCP.

```
/docs:export-pdf docs/pages/dashboard.md                    # Export single file
/docs:export-pdf docs/flows/sync-users.md --output manual.pdf  # Custom filename
```

What it does:

1. Reads the markdown file

2. Extracts H1/H2 headings to build a clickable table of contents

3. Converts markdown to styled HTML (code blocks, tables, images)

4. Uses Playwright MCP to render and export as PDF

5. Saves to `docs/exports/` directory

Output: `docs/exports/{filename}.pdf`

Features:

- Auto-generated TOC from H1/H2 headings with clickable links
- PDF-friendly styling (page breaks at H1, code block formatting)
- Embedded images (converted to base64)
- A4 format with proper margins

Example:

```
📄 Exporting: docs/pages/dashboard.md

📑 Table of Contents:
   • Dashboard Overview
     • Key Metrics
     • Navigation
   • Components
   • Configuration

🖨️ Rendering PDF...
   Format: A4
   Pages: 5

✅ PDF exported!
   📁 docs/exports/dashboard.pdf (245 KB)
```

Requirements:

- Playwright MCP must be available

Knowledge Base

The intelligent commands build a `docs/.knowledge/` folder:

```
docs/.knowledge/
├── _meta/                    # Project info
├── modules/
|   ├── campaigns/
|   |   ├── entity.json       # Entity definition
|   |   ├── routes.json       # API routes
|   |   ├── validation.json   # Validation rules
|   |   ├── flows/            # User flows
|   |   └── ui-states/        # Conditional UI
|   └── users/
|       └── ...
```

```
├── relationships/            # Cross-module relationships
└── cross-module-flows/       # Flows spanning modules
```

This knowledge powers smarter documentation generation.

Intelligent Workflow

For Single Flow (Quick)

```
/docs:flow "sync users from discord"    → Analyzes code, generates docs with diagrams
/docs:flow "import payments from csv"   → Includes UI screenshots if route detected
```

For Entire Project (Batch)

```
/docs:discover                → Scans codebase, finds all modules
         ↓
/docs:plan                    → Creates ordered documentation plan
         ↓
/docs:execute                 → Generates all docs with screenshots
```

Example Session

```
# Option A: Document a specific flow
/docs:flow "sync users from discord"      # Backend integration
/docs:flow "import payments from csv"     # Import with UI screenshots
/docs:flow "how stripe webhooks work"     # Webhook handling

# Option B: Document entire project
/docs:discover                            # Find all modules
/docs:plan                                # Create plan (docs/plan.yml)
/docs:execute                             # Generate all documentation

# Resume if interrupted
/docs:execute --continue

# After code changes
/docs:update --base main
```

What Makes It Smart

Capability	How It Works
Conditional UI	Clicks checkboxes/toggles, observes what fields appear
Validation Discovery	Submits empty/invalid forms, captures error messages
Cross-Page Tracking	Creates data, verifies it appears in lists/dashboards
Entity Lifecycle	Documents full create → view → edit → delete flow
Modular Analysis	One module at a time, scales to large projects
Code + UI Correlation	Matches frontend components to backend validation

Configuration

After running `/docs:init`, a `docs/config.yml` is created:

```
project:
  name: "My App"
  type: saas

style:
  tone: friendly  # friendly | professional | technical | minimal

urls:
  base: "https://myapp.com"

auth:
  method: file    # file | env | manual

output:
  directory: ./docs
```

Authentication Methods

Method	Description
\`file\`	Credentials stored in \`docs/.auth\` (gitignored)
\`env\`	Read from \`DOCS_AUTH_USER\` and \`DOCS_AUTH_PASS\`

`manual`	Pass `--auth user:pass` each time

Output

Generated documentation includes:

- **Overview** - What the page is for
- **Features** - What users can do
- **Key Actions** - Buttons and actions explained
- **Screenshots** - Full-page captures
- **How-to Guides** - Step-by-step flows (optional)
- **Related Pages** - Navigation links

Requirements

- Python 3.11+
- Claude Code (or Cursor/Copilot)
- Playwright MCP (for browser-based commands)

Installing Playwright MCP

Add to your `~/.claude.json` or project `.mcp.json`:

```
{
  "mcpServers": {
    "playwright": {
      "command": "npx",
      "args": ["@anthropic/mcp-playwright"]
    }
  }
}
```

Development

```
git clone https://github.com/binarcode/aidocs-cli.git
cd aidocs-cli
```

```
uv venv && uv pip install -e .
aidocs check
```

License

MIT

Generated by aidocs export-pdf