# Kawa Code ユーザーマニュアル

Kawa Code へようこそ。リアルタイム AI 対応コラボレーションプラットフォームです。

## 目次



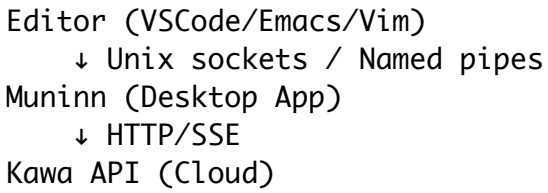
---

## 概要

Kawa Code エコシステムは、複数の相互接続されたコンポーネントで構成されています。

| コンポーネント | 説明 |
| --- | --- |
| **Muninn** | Kawa Code デスクトップアプリケーション - Git 操作を管理し、差分を生成し、クラウドと同期する中心的なハブ |
| **Muninn Extensions** | kawa.i18n コード翻訳拡張機能など、Kawa Code メインアプリケーション用の拡張機能 |
| **Huginn Extensions** | Kawa Code に接続し、コラボレーション機能を表示するエディタプラグイン (VSCode、Emacs、Vim) |
| **Kawa API** | チームメンバーを調整するクラウドサービス |
| **MCP Server** | インテント追跡機能を備えた AI 支援開発のための Claude Code との統合 |

## アーキテクチャ




## データストレージと同期

Kawa Code は、まずデータをローカルに保存し、次にチームコラボレーションのためにクラウドに同期します。

| データ種類 | ローカルストレージ | クラウド同期 | チーム共有 |
| --- | --- | --- | --- |
| **Intents** | `~/.kawa-code/.storage.caw` | 1 分ごと | はい |

| データ種類 | ローカルストレージ | クラウド同期 | チーム共有 |
|---|---|---|---|
| **Decisions** | `~/.kawa-code/.storage.caw` | 1 分ごと | はい |
| **Lessons** | `~/.kawa-code/.storage.caw` | 1 分ごと | はい |
| **Diffs** | 一時ファイル | 即座 | はい |

**主な機能:**

- **ローカルファースト**: すべての操作はオフラインで動作し、接続時にデータが同期されます
- **ゼロ知識暗号化**: すべてのコードはアップロード前にクライアント側で暗号化され、API には復号化キーがありません
- **自動チーム同期**: チームメンバーのデータはローカルでダウンロードされ復号化されます
- **競合検出**: コミット前に重複する作業が検出されます

---

## システム要件

### すべてのプラットフォーム

- Git 2.25 以降
- クラウド同期のためのインターネット接続

### Muninn デスクトップアプリ

- **macOS**: 10.15 (Catalina) 以降
- **Windows**: Windows 10 以降
- **Linux**: Ubuntu 20.04+ または同等

### エディタ拡張機能

- **VSCode**: バージョン 1.105.0 以降
- **Emacs**: バージョン 26.1 以降 (タブバー機能には 27.1+ を推奨)
- **Vim**: Python3 サポート付き 8.2+ または Neovim 0.5.0+

### MCP サーバー (Claude Code 用)

- Node.js 18.0.0 以降
- Claude Code CLI がインストールされていること

---

## インストール

### 1. Kawa Code デスクトップアプリ (Muninn)

コードネーム Muninn の Kawa Code メインアプリケーションは、他のすべてのコンポーネントが動作するために実行されている必要があるコアデスクトップアプリケーションです。

codeawareness.com から最新リリースをダウンロードしてください。

- **macOS**: `KawaCode-x.x.x.dmg`
- **Windows**: Microsoft Store からインストール
- **Linux**: `KawaCode-x.x.x.AppImage` または `.deb`

### 2. エディタ拡張機能

IDE (Visual Studio Code、emacs) を開き、Kawa Code 拡張機能をインストールしてください。 すべての拡張機能は Kawa Code が実行されている必要があります。

### 3. Claude Code 用 MCP サーバー

MCP (Model Context Protocol) サーバーは、Claude Code が開発インテントを追跡し、Kawa Code のコラボレーション機能と統合できるようにします。

**前提条件:**

- Node.js 18.0.0 以降
- Claude Code CLI がインストールされていること
- Kawa Code が実行されていること

**インストール:**

```
cd kawa.mcp

# 依存関係のインストール
npm install

# ビルド
npm run build
```

**Claude Code の設定:**

MCP サーバーを Claude Code で利用できるようにする方法は 3 つあります。セットアップに合った方法を選択してください。

**オプション A: プロジェクトレベルの設定 (チームに推奨)**

プロジェクトのルートディレクトリに `.mcp.json` ファイルを作成します。

```
{
  "mcpServers": {
    "kawa-intents": {
      "type": "stdio",
      "command": "node",
      "args": ["/absolute/path/to/kawa.mcp/build/index.js"]
    }
  }
}
```

このファイルを git にコミットしてください。リポジトリをクローンしたすべてのチームメンバーは、MCP サーバーを自動的に利用できるようになります (Claude Code は初回使用時に承認を求めます)。

**オプション B: ユーザーレベルの設定 (すべてのプロジェクトで利用可能)**

任意のディレクトリから実行します。

```
claude mcp add --transport stdio kawa-intents --scope user -- node /absolute/path/to/kawa.mcp/build/index.js
```

これにより、Claude Code で開くすべてのプロジェクトで kawa-intents サーバーが利用できるようになります。

**オプション C: 単一プロジェクトの設定 (プライベート、1 つのプロジェクトのみ)**

プロジェクトディレクトリ内から実行します。

```
claude mcp add --transport stdio kawa-intents -- node /absolute/path/to/kawa.mcp/build/index.js
```

これは、プロジェクトのパスの下の ~/.claude.json に設定を保存します。この設定は、Claude Code がその特定のディレクトリから起動された場合にのみ機能します。

上記のすべての例で、/absolute/path/to/kawa.mcp を kawa.mcp ディレクトリへの実際のパスに置き換えてください。

**インストールの確認:**

プロジェクトディレクトリで新しい Claude Code セッションを開始し、/mcp を実行します。以下のツールが利用可能な kawa-intents が表示されるはずです。

- get_project_context
- check_active_intent
- create_and_activate_intent
- list_lessons
- check_lessons_for_code
- その他...

**利用可能な MCP ツール:**

| ツール | 説明 |
| --- | --- |
| get_project_context | **セッション開始時に呼び出し** - プロジェクト概要、アクティブインテント、制約を読み込みます |
| get_relevant_context | **リクエストごとに呼び出し** - ユーザーの特定のタスクに関連するコンテキストを取得します |
| check_active_intent | 作業を開始する前にアクティブインテントを確認します |
| create_and_activate_intent | タスク用の新しいインテントを作成します |
| get_intents_for_file | ファイルのチーム競合を確認します |
| assign_blocks_to_intent | コード変更をインテントに関連付けます |
| complete_intent | インテントを committed/done/abandoned としてマークします |
| list_team_intents | チームメイトが取り組んでいることを確認します |
| get_project_decisions | プロジェクトのすべてのアーキテクチャ決定を確認します |
| record_decision | アーキテクチャ決定を記録します |
| list_lessons | 学習済みパターンを発見します (ローカル + チーム) |
| check_lessons_for_code | 既知のパターンに対してコードをチェックします (ローカル + チーム) |
| record_lesson | 新しい学習を文書化します (チームに同期) |

**トークン効率的なコンテキスト取得:**

CLAUDE.md のヒント:

多くのインテント、決定、レッスンを含む大規模プロジェクトの場合は、すべてを取得するのではなく `get_relevant_context` を使用してください。これは以下を行います。

- ユーザーのプロンプトからキーワードとファイルパターンを抽出します
- 現在のタスクへの関連性によってすべてのアイテムをスコアリングします
- 関連性スコア付きの一致するアイテムのみを返します
- すべての最近のアイテムを読み込むよりもはるかに効率的です

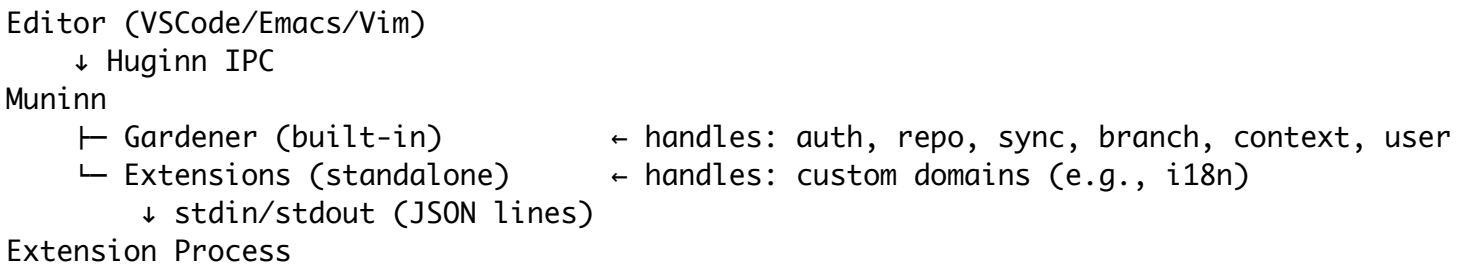
---

## Muninn 拡張機能

Kawa Code は **スタンドアロン拡張機能** をサポートしています。これは、組み込みの Kawa Code が提供するものを超えた機能を追加する別個のプロセスです。拡張機能は、Kawa IPC プロトコルを使用して stdin/stdout 経由で Kawa Code と通信し、オプションで Kawa Code の UI 内に表示される UI コンポーネントを提供できます。

**kawa.i18n** (コード翻訳) 拡張機能は、最初のスタンドアロン拡張機能として出荷されています。時間の経過とともにより多くの拡張機能が利用可能になる可能性があります。または、独自の拡張機能を構築することもできます。

### 拡張機能の仕組み




Kawa Code の起動時に、以下を行います。

1. `extension.json` マニフェストを含むサブディレクトリの `~/.kawa-code/extensions/` をスキャンします

2. 各マニフェストを検証します
3. 依存関係順に拡張機能をソートします
4. 各拡張機能を子プロセスとして生成します
5. ドメインサブスクリプションに基づいて IPC メッセージを拡張機能にルーティングします

## 拡張機能のインストール

### リリースバイナリから

1. プラットフォーム用の拡張機能アーカイブをダウンロードします
2. `~/.kawa-code/extensions/` に展開します (またはそこにリンクします)。

```
mkdir -p ~/.kawa-code/extensions
# 以下の構造になるように展開します:
# ~/.kawa-code/extensions/<extension-name>/
#   ├── extension.json
#   ├── binaries/
#   |   └── <binary-for-your-platform>
#   └── ui/dist/  (optional)
```

3. Kawa Code を再起動します - 拡張機能は起動時に自動的に検出されます

## ソースからのビルド

ソースから拡張機能をビルドしたい場合 (例: 開発用またはビルド済みバイナリがないプラットフォーム用)、kawa.i18n を例として以下の方法があります。

### 前提条件

- Node.js 18.0.0 以降
- npm または yarn

### ビルド手順

```
cd kawa.i18n

# 依存関係のインストール
npm install

# プラットフォーム用のビルド (1 つを選択):
npm run build:macos      # macOS Intel/Apple Silicon
npm run build:linux      # Linux x64
npm run build:windows    # Windows x64
```

これにより、TypeScript がコンパイルされ、pkg を使用してすべてを binaries/ 下のスタンドアロンバイナリにバンドルします。

**ビルド後のインストール**

**オプション A: 開発用のシンボリックリンク** (ソースへの変更が開発モードで即座に反映されます):

```
# 拡張機能はセットアップスクリプトを提供します:
./setup-dev-config.sh

# または手動で:
ln -s /path/to/kawa.i18n ~/.kawa-code/extensions/i18n
```

**オプション B: 本番用のコピー**:

```
mkdir -p ~/.kawa-code/extensions/i18n
cp extension.json ~/.kawa-code/extensions/i18n/
cp -r binaries/ ~/.kawa-code/extensions/i18n/
cp -r ui/dist/ ~/.kawa-code/extensions/i18n/ui/dist/   # if the extension has UI
```

インストール後、Muninn を再起動してください。

## 独自の拡張機能の作成

拡張機能は、stdin から JSON メッセージを読み取り、stdout に JSON レスポンスを書き込む任意の実行可能ファイルです。任意の言語 (Node.js、Python、Rust、Go など) で拡張機能を作成できます。

**最小要件**

1. 拡張機能のルートディレクトリに **`extension.json` マニフェスト**
2. マニフェストで参照される **実行可能バイナリまたはスクリプト**
3. メッセージルーティング用の **少なくとも 1 つのドメインサブスクリプション**

### マニフェスト形式 (`extension.json`)

以下は最小限のマニフェストです。

```
{
  "id": "my-extension",
  "name": "My Extension",
  "version": "1.0.0",
  "description": "What this extension does",
  "binary": {
    "path": "./binaries/my-extension"
  },
  "domains": {
    "subscribe": ["my-domain"]
  }
}
```

完全なマニフェストリファレンス:

| フィールド | 必須 | 説明 |
| --- | --- | --- |
| `id` | はい | 一意の識別子 (英数字、ハイフン、アンダースコア) |

| フィールド | 必須 | 説明 |
| --- | --- | --- |
| `name` | はい | 表示名 |
| `version` | はい | セマンティックバージョン ( `major.minor.patch` ) |
| `description` | はい | 短い説明 |
| `author` | いいえ | 著者名または組織 |
| `license` | いいえ | ライセンス識別子 (MIT、Apache-2.0 など) |
| `homepage` | いいえ | ドキュメント URL |
| `binary.path` | はい | 本番バイナリへのパス (拡張機能ディレクトリからの相対パス) |
| `binary.devPath` | いいえ | 開発モードスクリプトへのパス (例: `./dev.sh` ) |
| `binary.devMode` | いいえ | `"spawn"` (デフォルト) または `"socket"` |
| `binary.env` | いいえ | 設定する環境変数 |
| `binary.args` | いいえ | コマンドライン引数 |
| `domains.subscribe` | はい | 受信するメッセージドメインの配列 (少なくとも 1 つ) |
| `ui.webComponent.enabled` | いいえ | 拡張機能が UI を提供するかどうか |
| `ui.webComponent.path` | いいえ | コンパイルされた Web Component JS バンドルへのパス |

| フィールド | 必須 | 説明 |
| --- | --- | --- |
| `ui.webComponent.panels` | いいえ | サイドバー/ボトムパネルの定義 |
| `ui.webComponent.screens` | いいえ | メニュー項目付きの全画面定義 |
| `ui.settings.enabled` | いいえ | 拡張機能に設定パネルがあるかどうか |
| `dependencies` | いいえ | 必要な拡張機能の配列 (読み込み順にソート) |

**IPC プロトコル**

拡張機能は、stdin/stdout の JSON 行経由で Muninn と通信します。

**メッセージの受信** (stdin、1 行に 1 つの JSON オブジェクト):


```
{"flow":"req","domain":"my-domain","action":"do-something","caw":"client-1","data":{"key":"value"},"_msgId":"
```


**レスポンスの送信** (stdout、1 行に 1 つの JSON オブジェクト):


```
{"flow":"res","domain":"my-domain","action":"do-something","caw":"client-1","data":{"result":"ok"},"_msgId":"
```


**メッセージフィールド:**

| フィールド | 値 | 説明 |
| --- | --- | --- |
| `flow` | `req` , `res` , `err` , `brdc` | リクエスト、レスポンス、エラー、またはブロードキャスト |

| フィールド | 値 | 説明 |
|---|---|---|
| `domain` | string | メッセージドメイン (サブスクリプションと一致する必要があります) |
| `action` | string | ドメイン内のアクション名 |
| `caw` | string | クライアント ID (どのエディタがリクエストを送信したか) |
| `data` | any | ペイロード |
| `_msgId` | string | リクエストとレスポンスを関連付けるためのメッセージ ID |

**重要**: ログには stderr を使用してください。stdout に書き込まれるものは有効な JSON でなければなりません。不正なログ出力はプロトコルを壊します。

**ディレクトリ構造**

拡張機能を `~/.kawa-code/extensions/<your-extension-id>/` に配置します。

```
~/.kawa-code/extensions/my-extension/
├── extension.json          # Manifest (required)
├── binaries/
|   ├── my-extension-macos  # macOS binary
|   ├── my-extension-linux  # Linux binary
|   └── my-extension.exe    # Windows binary
├── dev.sh                  # Dev mode wrapper (optional)
└── ui/
    └── dist/
        └── my-ui.js        # Web Component bundle (optional)
```

**UI 統合 (オプション)**

拡張機能は、Lit Web Components を使用して UI コンポーネントを提供できます。Muninn は JS バンドルを読み込み、2 つの方法でコンポーネントをレンダリングします。

- **Panels**: Muninn のサイドバーまたは下部エリアに表示されます
- **Screens**: Muninn のメニューからアクセスできる全画面ビュー

両方のパターンの実用例については、kawa.i18n 拡張機能のソースを参照してください。

---

## はじめに

### 1. 初期セットアップ

1. **Kawa Code をインストール** してアカウントを作成します
2. **エディタ拡張機能をインストール** (VSCode、Emacs、または Vim)
3. **kawa.mcp をインストール** (オプション) Claude Code で作業したい場合
4. **kawa.i18n をインストール** (オプション) 自分の言語でコードを読みたい場合 (Claude Code API に依存)
5. **Git リポジトリを開く** エディタで
6. **接続を確認**: エディタで Kawa Code のステータスインジケーターを確認します

### 2. 基本的なワークフロー

1. **通常通りコードを作業** します
2. **チームメイトが変更している行のハイライトを確認** します
3. **サイドバー/パネルでピア差分を表示** します
4. **Claude Code** にコード作成を手伝ってもらいます
5. コードを開発するにつれて進化する **Intents** を確認します
6. kawa.i18n がインストールされている場合は **Code View** をクリックします。IDE (VSCode、emacs) で移動すると、翻訳されたファイルが Code View モードで自動的に表示されます。
7. 準備ができたら **コミットしてプッシュ** します

### 3. Claude Code での使用

1. **MCP サーバーをインストール** (上記参照)
2. Claude Code で **コーディングセッションを開始** します
3. Claude は自動的に以下を行います。
    - セッション開始時にプロジェクト概要を読み込みます (`get_project_context`)
    - 各リクエストのタスク関連コンテキストを取得します (`get_relevant_context`)
    - アクティブインテントを確認します
    - 新しいタスクのインテントを作成します
    - コード変更を追跡します
    - コードを書く前に学習済みレッスンを確認します
    - 実装中にアーキテクチャ決定を記録します
    - **コミット前に会話をレビュー** して見逃した決定をキャッチします
    - コミットを支援し、決定の要約を含めます

---

## CLAUDE.md のセットアップ

`CLAUDE.md` は、リポジトリで作業する際に Claude Code に指示を提供する特別なファイルです。これにより、Claude はインテント追跡、決定記録、学習済みレッスンを含む Kawa Code の全機能セットを使用できるようになります。

### CLAUDE.md が重要な理由

`CLAUDE.md` ファイルがないと、Claude Code は以下を行いません。

- インテントで作業を追跡する
- アーキテクチャ決定を記録する
- コードを書く前に学習済みレッスンを確認する
- MCP ツールに正しいリポジトリオリジンを使用する

### 最小限必要な CLAUDE.md

最低限、以下を含む `CLAUDE.md` ファイルを作成してください。

```
# CLAUDE.md

## Project Overview
[プロジェクトの 1-2 文の説明]

## AI Workflow
1. セッション開始時にプロジェクト概要のために `get_project_context` を呼び出します
2. ユーザーのリクエストを探索します (ファイルを読み、範囲を理解します)
3. プロンプト + 発見したファイルで `get_relevant_context` を呼び出します
4. コーディング前に `check_active_intent` を使用します
5. 編集後に `assign_blocks_to_intent` を使用します

Repository origin: `git@github.com:your-org/your-repo.git`
Repository path: `/absolute/path/to/your-repo`
```

**重要**: repoPath は `.git` フォルダを含むディレクトリを指す必要があります。マルチプロジェクトセットアップ (モノレポ) では、親ディレクトリではなく、各サブプロジェクトのパスを使用してください。

### インテント追跡の有効化

Claude が作業内容を追跡できるように、このセクションを追加してください。

```
## AI Code Implementation Workflow

### Workflow Checkpoints
```

| When | Action | Tool |
| --- | --- | --- |
| **Session start** | プロジェクト概要を読み込みます | `get_project_context` |

| When | Action | Tool |
|---|---|---|
| **After exploring request** | タスク関連コンテキストを取得します (発見したファイルと共に) | `get_relevant_context` |
| **Before coding** | アクティブインテントを確認します | `check_active_intent`, `create_and_activate_intent` |
| **After file edits** | コードブロックを割り当てます | `assign_blocks_to_intent` |
| **On commit** | インテントを完了します | `get_intent_changes`, `complete_intent` |

**注**: 関与するファイルを探索した後に `get_relevant_context` を呼び出してください - より良い関連性マッチングのためにプロンプ

```
### Repository Origin and Path
MCP ツールにこれらを使用します:
- `repoOrigin`: `git@github.com:your-org/your-repo.git`
- `repoPath`: `/absolute/path/to/your-repo` (`.git` ディレクトリを含む必要があります)
```

## 決定記録の有効化

アーキテクチャ決定を自動的にキャプチャするには、このセクションを追加してください。

```
### Recording Decisions

以下の場合に `record_decision` を使用して決定を静かに記録します:
```

| Trigger | Decision Type |
| --- | --- |
| 代替案の間で選択する | `fork` |
| 失敗するアプローチを試す | `abandoned` |
| 予期しない制限を見つける | `discovery` |
| 厳しい要件を特定する | `constraint` |
| 明示的なトレードオフを行う | `tradeoff` |
| ライブラリ/依存関係を選択する | `dependency` |

決定はコミット前にレビューされ、コミットメッセージに含まれます。

**決定はいつ記録されますか?** 決定は 2 つの方法でキャプチャされます:

1. **実装中** - コーディング中、Claude はアーキテクチャの選択を行う際に静かに決定を記録します
2. **コミット前（自動レビュー）** - Claude は会話を分析し、見逃した決定をキャッチします

**コミット前の決定レビュー:**
コミット前に、Claude は実装中に記録されなかったアーキテクチャ決定について会話を自動的にレビューします:
- 議論された代替案: "X または Y を使用すべきか?" → "なぜなら... という理由で X を使用しよう"
- 拒否されたアプローチ: "X はできません、なぜなら..." または "これは... のために機能しません"
- 行われたトレードオフ: "トレードオフは..." または "パフォーマンスよりもシンプルさを選択しました"
- 発見された制約: "...する必要があります" または "これは... によって制限されています"
- 選択された依存関係: "[目的] のために [ライブラリ] を使用しよう"
- アプローチの変更: "最初に X を試しましたが、... という理由で Y に切り替えました"

これにより、会話の決定（計画/議論中に行われた）が実装の決定と一緒にキャプチャされることが保証されます。

## 制約チェックの有効化

プロジェクトにアーキテクチャ制約 (セキュリティ、パフォーマンスなど) がある場合:

1. 制約をリストした CONSTRAINTS.md ファイルを作成します
2. CLAUDE.md にこれを追加します。

```
### Constraint Pre-Check

アーキテクチャソリューションを提案する前に:
1. 制約を読み込むために `get_project_constraints` を呼び出します
2. 各制約に対して各オプションをチェックします
3. `constraintViolations` を使用して決定レコードに違反を文書化します
4. 準拠しているオプションのみを提案します
```

## 学習済みレッスンの有効化

学習済みレッスンは、同じミスを繰り返さないようにするのに役立ちます。レッスンは MongoDB に保存され、暗号化された同期を介してチームメンバーと自動的に共有されます。

**仕組み:**

1. **ローカルファーストストレージ**: レッスンは Muninn にローカルに保存され、クラウドに同期されます
2. **チーム共有**: チームメンバーのレッスンは自動的にダウンロードされ、コードに対してチェックされます
3. **ゼロ知識**: すべてのレッスンコンテンツはアップロード前にクライアント側で暗号化されます
4. **パターンマッチング**: AI は書く前に既知のパターンに対してコードをチェックします

**CLAUDE.md にこれを追加:**

```
### Lessons Learned Integration

1. **セッション開始時**: パターンを発見するために `list_lessons` を呼び出します
2. **コードを書く前**: コードが発見されたパターンに関与する場合、`check_lessons_for_code` を呼び出します
3. **問題を修正した後**: パターンを文書化するために `record_lesson` を呼び出します
```

**レッスンの記録:**

問題を引き起こすパターンを発見した場合、Claude はそれを記録できます。

```
record_lesson({
  repoOrigin: "git@github.com:your-org/repo.git",
  title: "block_on_runtime panics inside async context",
  patterns: ["block_on_runtime", "tokio", "async handler"],
  symptom: "Runtime panic: Cannot start runtime from within runtime",
  solution: "Use block_on_runtime_safe() which handles nested runtime detection",
  context: "Tokio doesn't allow blocking on a runtime from within an async context"
})
```

**LESSONS_LEARNED.md からのインポート (オプション):**

既存の LESSONS_LEARNED.md ファイルがある場合、データベースにインポートできます。ファイル形式は以下の通りです。

```
# Lessons Learned

## How to Use This File
**For AI Assistants**: Before implementing code, scan this file for matching patterns.

---

### LL-001: [Short description]
```

| Field | Value |
| --- | --- |
| **Pattern** | `keyword1`, `keyword2` |
| **Symptom** | [What goes wrong] |
| **Solution** | [The fix] |
| **Context** | [Why this happens] |

```
**Files**: `path/to/relevant/files`
```

そのリポジトリのデータベースにレッスンが存在しない場合、MCP サーバーは LESSONS_LEARNED.md からレッスンを自動的にインポートします。

## 完全な MCP ツールリファレンス

Claude がすべての利用可能なツールを知ることができるように、この表を含めてください。

```
### MCP Tools Reference

#### Context (Call First)
```

| Tool | When to Use |
|---|---|
| `get_project_context` | **SESSION START** - プロジェクト概要、アクティブインテント、制約を読み込みます |
| `get_relevant_context` | **EACH REQUEST** - ユーザーのプロンプトに関連するコンテキストを取得します |

```
#### Intent Tools
```

| Tool | When to Use |
|---|---|
| `check_active_intent` | 任意のコードタスクを開始する前 |
| `create_and_activate_intent` | アクティブインテントが存在しない場合 |
| `get_intents_for_file` | ファイルを変更する前 (競合をチェック) |
| `assign_blocks_to_intent` | コードを書いた後 |
| `get_intent_changes` | コミット前 |

| Tool | When to Use |
|---|---|
| `complete_intent` | git コミット後 |
| `list_team_intents` | チームメイトの作業を確認するため |

#### Decision Tools

| Tool | When to Use |
|---|---|
| `record_decision` | アーキテクチャ決定を行う際 |
| `get_session_decisions` | コミット前 |
| `get_project_decisions` | すべてのプロジェクト決定を確認するため |
| `edit_session_decision` | コミット前に決定を変更するため |
| `get_project_constraints` | ソリューションを提案する前 |
| `detect_intent_conflicts` | コミット前 (チーム競合をチェック) |

#### Lessons Tools

| Tool | When to Use |
| --- | --- |
| `list_lessons` | セッション開始時 (チームレッスンを含む) |
| `check_lessons_for_code` | パターンに一致するコードを書く前 |
| `record_lesson` | 繰り返し発生する問題を修正した後 (チームに同期) |

### マルチプロジェクト / モノレポのセットアップ

ワークスペースに単一の親ディレクトリの下に複数の git リポジトリが含まれている場合、各サブプロジェクトには独自のオリジンとパスマッピングが必要です。CLAUDE.md に表を追加してください。

```
### Repository Origins and Paths

各サブプロジェクトには独自の `.git` ディレクトリがあります。MCP ツールを呼び出す際には、`repoPath` として **サブプロジェクトパ
```

| Sub-project | `repoOrigin` | `repoPath` |
| --- | --- | --- |
| my-api | `git@github.com:my-org/my-api.git` | `/path/to/workspace/my-api` |
| my-frontend | `git@github.com:my-org/my-frontend.git` | `/path/to/workspace/my-frontend` |

---

### 関連性ベースの取得

2 つの補完的なツールを使用します。

| Tool | When | What it returns |
| --- | --- | --- |
| `get_project_context` | セッション開始 | プロジェクト概要: アクティブインテント、制約、チームステータス、カウント |
| `get_relevant_context` | 各ユーザーリクエスト | 特定のタスクに関連するアイテムのみ |

## 関連性スコアリングの仕組み

"ユーザー登録エンドポイントに検証を追加する" などのユーザープロンプトで `get_relevant_context` を呼び出すと、システムは以下を行います。

1. **キーワードを抽出**: `["validation", "user", "registration", "endpoint"]`
2. **ファイルパターンを推論**: `["**/valid*", "**/user*", "**/routes/**"]`
3. **ドメインを特定**: `["API", "Validation"]`
4. **すべてのアイテムをスコアリング**:
    - タイトル/説明/要約のキーワード一致 (重み: 5)
    - コードブロックとのファイルパターンの重複 (重み: 10)
    - 制約違反ブースト (決定の場合 1.3 倍)
    - アクティブステータスブースト (インテントの場合 1.5 倍)
5. **スコアと一致詳細付きで上位の一致を返します**

## レスポンス例

```
{
  "relevantIntents": [
    {
      "id": "abc-123",
      "title": "Add user registration API",
      "score": 0.85,
      "matchedOn": ["title:user", "title:registration", "file:src/routes/user.rs"]
    }
  ],
  "relevantDecisions": [
    {
      "id": "dec-456",
      "summary": "Chose Zod for validation schemas",
      "score": 0.72,
      "matchedOn": ["summary:validation"]
    }
  ],
  "relevantLessons": [
    {
      "id": "LL-003",
      "title": "Validation error messages must be user-friendly",
      "score": 0.68,
      "matchedOn": ["pattern:validation", "file:**/routes/**"]
    }
  ],
  "extractedCues": {
    "keywords": ["validation", "user", "registration", "endpoint"],
    "filePatterns": ["**/valid*", "**/user*", "**/routes/**"],
    "domainHints": ["API", "Validation"]
  }
}
```


推奨ワークフロー

```
セッション開始:
  └> get_project_context
       └> 返す: アクティブインテント、制約、チーム概要、カウント

各ユーザーリクエスト:
  1. リクエストを探索します
     └> ファイルを読み、範囲を理解し、関与するコードを特定します
     └> どのファイルが関連しているかに注意します

  2. 関連するコンテキストを取得します (探索後)
     └> get_relevant_context(prompt, activeFiles: [discovered files])
     └> 返す: 関連するインテント、決定、レッスンのみ
     └> ファイルコンテキストが含まれているとはるかに良い一致になります

  3. 実装を進めます
     └> 関連するコンテキストを使用して作業を通知します
```

**探索後に呼び出す理由は?** "バグを修正する" のような曖昧なプロンプトでは、キーワード抽出が不十分になります。探索後、関与する特定のファイルと概念がわかるため、関連性マッチングがはるかに効果的になります。

このアプローチにより、トークン使用量はプロジェクトサイズではなく、タスクの複雑さに比例します。

---

## トラブルシューティング

### Kawa Code が起動しない

1. システム要件を確認してください
2. 他のインスタンスが実行されていないことを確認してください
3. `~/.kawa-code/logs/` のログを確認してください
4. ターミナルから実行してエラーメッセージを確認してください

### エディタ拡張機能が接続しない

1. Kawa Code (Muninn) が実行されていることを確認してください
2. ソケットファイルが存在することを確認してください: `~/.kawa-code/sockets/muninn`
3. Muninn とエディタの両方を再起動してください
4. エディタの出力/コンソールでエラーメッセージを確認してください

### MCP サーバーの問題

1. Node.js のバージョンを確認してください: `node --version` (18+ である必要があります)
2. Claude Code で `/mcp` を実行して `kawa-intents` がリストされているか確認してください
3. 欠落している場合は、設定を確認してください。
    - **Project scope**: リポジトリルートの `.mcp.json`
    - **User/Local scope**: `~/.claude.json` の `projects` → <your-project-path> → `mcpServers` キーの下
4. 設定のすべてのパスが絶対パスであることを確認してください (相対パスではありません)
5. `kawa.mcp/` で `npm run build` が正常に完了したことを確認してください
6. Muninn が実行されているか確認してください
7. 設定変更後、Claude Code を再起動してください

### Windows 固有の問題

- Unix ソケットの代わりに名前付きパイプが使用されます
- アンチウイルスが `~/.kawa-code/` をスキャンする場合があります - 除外の追加を検討してください
- 権限の問題が発生した場合は、Muninn を管理者として実行してください

### 一般的なエラーメッセージ

| エラー | 解決策 |
| --- | --- |
| "Cannot connect to Muninn" | Muninn デスクトップアプリを起動してください |
| "Socket not found" | `~/.kawa-code/sockets/` ディレクトリを確認してください |

| エラー | 解決策 |
| --- | --- |
| "Authentication failed" | Muninn で再ログインしてください |
| "MCP server not responding" | Node.js のバージョンを確認して再ビルドしてください |

## ヘルプの入手

- **GitHub Issues**: バグを報告し、機能をリクエストしてください
- **Discord**: リアルタイムヘルプのために コミュニティに参加 してください
- **Documentation**: codeawareness.com/docs を訪問してください

## ライセンス

Kawa Code は **オープンソースではありません** が、kawa.i18n Muninn 拡張機能、VSCode、emacs などの Huginn 拡張機能、および kawa.MCP はオープンソースです。

| コンポーネント | ライセンス | オープンソース |
| --- | --- | --- |
| **kawa.mcp** (MCP Server) | MIT | はい |
| **kawa.vim** | MIT | はい |
| **kawa.emacs** | GPLv3 | はい |
| **kawa.vscode** | Proprietary | ソース利用可能 |

完全なライセンスの詳細については、個々のコンポーネントリポジトリを参照してください。