Spec Driven Docs Infra

ドキュメンテーション戦略

Documentation Team

目次

1. 生成AI時代のドキュメント基盤

生成AI時代におけるドキュメント基盤のテンプレート。

Markdown文書を静的サイトジェネレーターでHTMLに変換・公開することで、次を実現する。

- Markdownによる文書記述
- Mermaidによる図表作成
- Draw.ioによるSVG図表作成
- GitHub Pages*1 もしくは Azure Static Web Apps（以降SWA）による正式文書公開
- Pull Request時にSWAでのプレビュー
- GitHubリポジトリの権限に応じたセキュリティ管理*1
- GitHub Discussionsを通じた新規ユーザーへの招待・承認フロー*2

*1: GitHub Pages*でリポジトリ権限に応じた閲覧制御を行うには、*GitHub Enterprise*プランが必要 2: SWAへのアクセス権限はGitHub Discussionsの招待を承認することで付与される

1.1 技術スタック

技術	用途
Python + uv	MkDocs の実行環境と依存関係管理
MkDocs + Material for MkDocs	静的サイトジェネレーター
MkDocs プラグイン群	Mermaid の SVG/PNG 変換、PDF 出力、表読込
Node.js + pnpm	Marp と textlint の実行基盤
Mermaid	Markdown内での図表作成
Draw.io	SVG図表作成
Marp	Markdownスライド作成
Playwright	Mermaid 変換時のブラウザ自動化
WeasyPrint	PDF生成
textlint	ドキュメント品質チェック

1.2 外部リソース

- デモサイト
- プロポーザル
- プレゼン資料

1.3 技術スタック

- Markdown
- **MkDocs(Material for MkDocs)**: 静的サイトジェネレーター
- **Mermaid**: 図表作成ライブラリ

- **Draw.io（diagrams.net）**: SVG描画ツール
- **Marp**: Markdownプレゼンテーションツール

1.4 利用サービス・ツール

- Coding Agent（Codex CLI, Claude Code, GitHub Copilotなど）
- GitHub
- GitHub Actions
- Azure Static Web Apps
- swa-github-repo-auth
- Visual Studio Code
- Draw.io Integration
- テキスト校正くん
- MkDocs
- Python
- uv
- mkdocs-mermaid-to-svg
- mkdocs-svg-to-png
- mkdocs-to-pdf
- mkdocs-table-reader-plugin
- Marp
- Node.js
- Excel
- CopyToMarkdownAddIn

以下、T.B.D

1.5 🚀 セットアップ

1.5.1 前提条件

- Windows 10/11
- PowerShell 5.1 以上
- インターネット接続（パッケージダウンロード用）

1.5.2 自動セットアップ

管理者権限でPowerShellを起動し、以下のコマンドを実行してください：

```
.\scripts\Setup-Environments.ps1
```

このスクリプトは以下を自動でインストール・セットアップする：

- Python 3.13
- uv（Python環境管理）
- Node.js
- Mermaid CLI
- GTK+ Runtime（PDF生成用）
- プロジェクト依存関係

1.5.3 手動セットアップ

1. **リポジトリのクローン**

```
git clone <repository-url>
cd DocumentationStrategy
```

2. **Python環境のセットアップ**

```
uv sync
```

3. **開発サーバーの起動**

```
uv run mkdocs serve
```

4. **ブラウザでアクセス**

```
http://localhost:8000
```

1.6 📝 使用方法

1.6.1 ローカル開発

```
# 開発サーバー起動
uv run mkdocs serve

# 本番ビルド
uv run mkdocs build

# PDF生成
uv run mkdocs build --config-file mkdocs.yml
```

1.6.2 ドキュメント編集

1. `docs/` ディレクトリ内のMarkdownファイルを編集
2. Mermaid図表は、コードブロック内で `mermaid` 言語を指定
3. 変更は自動的にライブリロードで反映

1.6.3 新しい図表の追加

```
```mermaid
graph TD
    A[開始] --> B[処理]
    B --> C[終了]
```
```

1.7 📁 プロジェクト構造

```
DocumentationStrategy/
├── docs/                          # ドキュメントソース
│   ├── 01.システム設計/          # システム設計関連
│   ├── プロセス・フロー/         # プロセス図表
│   ├── プロジェクト管理/         # プロジェクト管理図表
│   ├── stylesheets/               # カスタムCSS
│   └── index.md                   # トップページ
├── scripts/                       # セットアップスクリプト
├── mkdocs.yml                     # MkDocs設定
├── pyproject.toml                 # Python依存関係
└── README.md                      # このファイル
```

1.8 🔧 設定

1.8.1 MkDocs設定（mkdocs.yml）

主要な設定項目：

- **テーマ**: Material for MkDocs（日本語対応）
- **プラグイン**:
- `mermaid-to-image`: PDF出力用の図表変換
- `to-pdf`: PDF生成機能
- **拡張機能**: Mermaid図表サポート、コードハイライト

1.8.2 Python依存関係（pyproject.toml）

- MkDocs関連パッケージ
- PDF生成用ライブラリ（WeasyPrint）
- Mermaid図表処理プラグイン

1.9 📄 PDF出力

ビルド時に自動的にPDFが生成される：

```
uv run mkdocs build
```

生成されたPDF: `site/pdf/ドキュメンテーション戦略.pdf`

1.10 🤝 コントリビューション

1. このリポジトリをフォーク
2. フィーチャーブランチを作成
3. 変更をコミット
4. プルリクエストを送信

1.11 🆘 トラブルシューティング

1.11.1 よくある問題

1. **PDF生成エラー**
2. GTK+ Runtimeがインストールされているか確認
3. WeasyPrintの依存関係を確認
4. **Mermaid図表が表示されない**

5. ブラウザのキャッシュをクリア
6. JavaScriptが有効になっているか確認
7. **セットアップスクリプトが動作しない**
8. PowerShellを管理者権限で実行
9. ExecutionPolicyを確認

1.11.2 サポート

問題が発生した場合は、GitHubのIssuesで報告すること。

2. クラウド環境構築手順

2.1 前提条件

- Azure サブスクリプションに対するリソース作成権限があること
- GitHub リポジトリの管理者権限があること
- `az` CLI と `gh` CLI がインストール済みであること
- `az login` と `gh auth login` が完了していること

2.2 手順

2.2.1 1. 命名規則とパラメーターを設定する

GitHubのOwnerとRepository名を設定する。

PowerShell **Bash**

```
$Owner = "<org-or-user>"
$Repository = "<repo>"
$githubRepo = "$Owner/$Repository"

$location = "japaneast"
$swaLocation = "eastasia"

$resourceGroupName = "rg-$Repository-prod"
$swaName = "stapp-$Repository-prod"
$identityName = "id-$Repository-prod"
$federatedCredentialName = "fc-github-actions-main"

Owner="<org-or-user>"
Repository="<repo>"
githubRepo="${Owner}/${Repository}"

location="japaneast"
swaLocation="eastasia"

resourceGroupName="rg-${Repository}-prod"
swaName="stapp-${Repository}-prod"
identityName="id-${Repository}-prod"
federatedCredentialName="fc-github-actions-main"
```

2.2.2 2. リソースグループと Static Web App の作成

PowerShell **Bash**

```
az group create --name $resourceGroupName --location $location

az staticwebapp create `
  --name $swaName `
  --resource-group $resourceGroupName `
  --location $swaLocation `
  --sku Standard

$defaultHostname = az staticwebapp show `
  --name $swaName `
  --resource-group $resourceGroupName `
  --query defaultHostname `
  -o tsv

az group create --name $resourceGroupName --location $location

az staticwebapp create \
  --name $swaName \
  --resource-group $resourceGroupName \
  --location $swaLocation \
  --sku Standard

defaultHostname=$(az staticwebapp show \
  --name $swaName \
  --resource-group $resourceGroupName \
  --query defaultHostname \
  -o tsv)
```

2.2.3 3. マネージド ID と OIDC フェデレーション資格情報の作成

既定ブランチが `main` 以外の場合は `--subject` の `refs/heads/main` を置き換える。

PowerShell **Bash**

```
az identity create --name $identityName --resource-group $resourceGroupName --location $location

$identity = az identity show `
  --name $identityName `
  --resource-group $resourceGroupName `
  --query "{clientId:clientId, principalId:principalId}" `
  -o json | ConvertFrom-Json
$clientId = $identity.clientId
$principalId = $identity.principalId

az identity federated-credential create `
  --name $federatedCredentialName `
  --identity-name $identityName `
  --resource-group $resourceGroupName `
  --issuer "https://token.actions.githubusercontent.com" `
  --subject "repo:$githubRepo:ref:refs/heads/main" `
  --audiences "api://AzureADTokenExchange"
```

```
az identity create --name $identityName --resource-group $resourceGroupName --location $location

clientId=$(az identity show \
  --name $identityName \
  --resource-group $resourceGroupName \
  --query clientId \
  -o tsv)
principalId=$(az identity show \
  --name $identityName \
  --resource-group $resourceGroupName \
  --query principalId \
  -o tsv)

az identity federated-credential create \
  --name $federatedCredentialName \
  --identity-name $identityName \
  --resource-group $resourceGroupName \
  --issuer "https://token.actions.githubusercontent.com" \
  --subject "repo:$githubRepo:ref:refs/heads/main" \
  --audiences "api://AzureADTokenExchange"
```

2.2.4 4. Static Web App への RBAC 付与

伝播待ちで失敗する場合は数十秒待って再実行する。

PowerShell **Bash**

```
$swaId = az staticwebapp show `
  --name $swaName `
  --resource-group $resourceGroupName `
  --query id `
  -o tsv
az role assignment create `
  --assignee-object-id $principalId `
  --assignee-principal-type ServicePrincipal `
  --role Contributor `
  --scope $swaId
```

```
swaId=$(az staticwebapp show \
  --name $swaName \
  --resource-group $resourceGroupName \
  --query id \
  -o tsv)
az role assignment create \
  --assignee-object-id $principalId \
  --assignee-principal-type ServicePrincipal \
  --role Contributor \
  --scope $swaId
```

2.2.5 5. GitHub Pages の有効化

GitHub Pagesにデプロイするため、リポジトリ設定で有効化する必要がある。

1. GitHub リポジトリの `Settings -> Pages` に移動する。
2. `Build and deployment` の `Source` で `GitHub Actions` を選択する。

> **Environment の自動作成**
>
> GitHub Actions をソースに設定すると、`github-pages` という名前の environment が自動的に作成される。ワークフローはこの environment を使用してデプロイを行う。

2.2.6 6. GitHub Discussions の有効化とカテゴリ作成

```
gh repo edit $githubRepo --enable-discussions
```

1. GitHub の `Settings -> Discussions` に移動する。
2. `Set up discussions` をクリックする（初回のみ）。
3. `Categories -> New category` をクリックする。
4. `Name: Invitation`, `Description: SWA role sync invitations`, `Format: Announcement` を設定して作成する。

2.2.7 7. GitHub Actions の Variables と Secrets の登録

PowerShell **Bash**

```powershell
$tenantId = az account show --query tenantId -o tsv
$subscriptionId = az account show --query id -o tsv
$swaApiToken = az staticwebapp secrets list `
  --name $swaName `
  --resource-group $resourceGroupName `
  --query properties.apiKey `
  -o tsv

gh variable set AZURE_CLIENT_ID --body $clientId --repo $githubRepo
gh variable set AZURE_TENANT_ID --body $tenantId --repo $githubRepo
gh variable set AZURE_SUBSCRIPTION_ID --body $subscriptionId --repo $githubRepo
gh variable set AZURE_SWA_NAME --body $swaName --repo $githubRepo
gh variable set AZURE_SWA_RESOURCE_GROUP --body $resourceGroupName --repo $githubRepo

gh secret set AZURE_SWA_API_TOKEN --body $swaApiToken --repo $githubRepo
```

```bash
tenantId=$(az account show --query tenantId -o tsv)
subscriptionId=$(az account show --query id -o tsv)
swaApiToken=$(az staticwebapp secrets list \
  --name $swaName \
  --resource-group $resourceGroupName \
  --query properties.apiKey \
  -o tsv)

gh variable set AZURE_CLIENT_ID --body "${clientId}" --repo $githubRepo
gh variable set AZURE_TENANT_ID --body "${tenantId}" --repo $githubRepo
gh variable set AZURE_SUBSCRIPTION_ID --body "${subscriptionId}" --repo $githubRepo
gh variable set AZURE_SWA_NAME --body "${swaName}" --repo $githubRepo
gh variable set AZURE_SWA_RESOURCE_GROUP --body "${resourceGroupName}" --repo $githubRepo

gh secret set AZURE_SWA_API_TOKEN --body "${swaApiToken}" --repo $githubRepo
```

2.2.8 8. GitHub App の作成と連携情報の登録

1. `GitHub -> Settings -> Developer settings -> GitHub Apps -> New GitHub App` を開く。
2. `App name` は任意、`Homepage URL` はリポジトリ URL を入力する。
3. `Webhook` の `Active` をオフにする。
4. `Permissions -> Repository -> Discussions: Read and write` を設定する。
5. `Create GitHub App` をクリックする。
6. `App ID` を控える。
7. `Private keys -> Generate a private key` で PEM をダウンロードする。

8. 連携情報を登録する：

PowerShell Bash

```
gh variable set ROLE_SYNC_APP_ID --body "<appId>" --repo $githubRepo
gh secret set ROLE_SYNC_APP_PRIVATE_KEY --body (Get-Content -Raw -Path "<pemPath>") --repo $githubRepo

gh variable set ROLE_SYNC_APP_ID --body "<appId>" --repo $githubRepo
gh secret set ROLE_SYNC_APP_PRIVATE_KEY --body "$(cat '<pemPath>')" --repo $githubRepo
```

2.3 完了確認

- `https://$defaultHostname` にアクセスできることを確認する。

3. アーキテクチャー

3.1 アーキテクチャ

このセクションでは、ドキュメント基盤の技術的な仕組みと設定について説明する。

3.1.1 ドキュメント

- デプロイ構成 - GitHub Actions と Azure Static Web Apps を使った配信構成
- ワークフロー アーキテクチャ - GitHub Actions ワークフローの実行条件と内部構造
- テキスト校正 - textlintによる日本語文書の品質管理

3.2 デプロイ構成

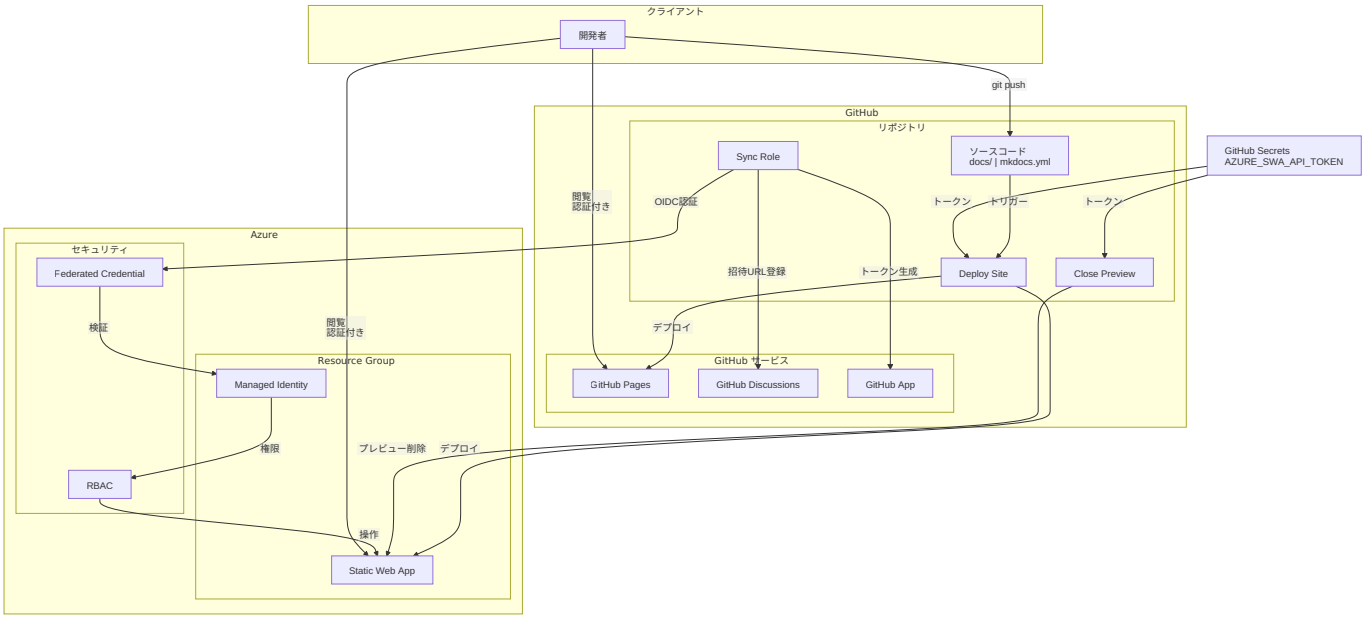
本ドキュメント基盤のデプロイアーキテクチャについて説明する。

3.2.1 概要

本システムは GitHub と Azure を組み合わせたハイブリッド構成で、以下の特徴を持つ。

- **デュアルデプロイ**: GitHub Pages と Azure Static Web Apps（SWA）への同時デプロイ
- **役割の分離**: 広範な閲覧は GitHub Pages、PRプレビューや限定配布は SWA を主に使用（同一成果物を両環境へ配布）
- **スケーラビリティ**: SWA のロール認証制限（25人）を補うため、広範な閲覧には Pages を活用
- **OIDC認証**: Sync Role が OIDC で Azure にログインして SWA ロールを同期（Azure 資格情報の長期保存を回避）
- **SWA API トークン**: Deploy Site / Close Preview が `AZURE_SWA_API_TOKEN` で SWA へデプロイ・プレビュー削除
- **ロールベースアクセス制御**: リポジトリ権限に基づく閲覧制御（SWA および Enterprise 版 Pages）
- **招待管理**: GitHub Discussions を活用した閲覧権限の配布と承認フロー（SWA用）
- **最適化されたビルド**: Web 表示用には軽量な SVG、PDF 生成用には互換性の高い PNG を使い分け

3.2.2 配置モデル



3.2.3 コンポーネント詳細

GitHub 側リソース

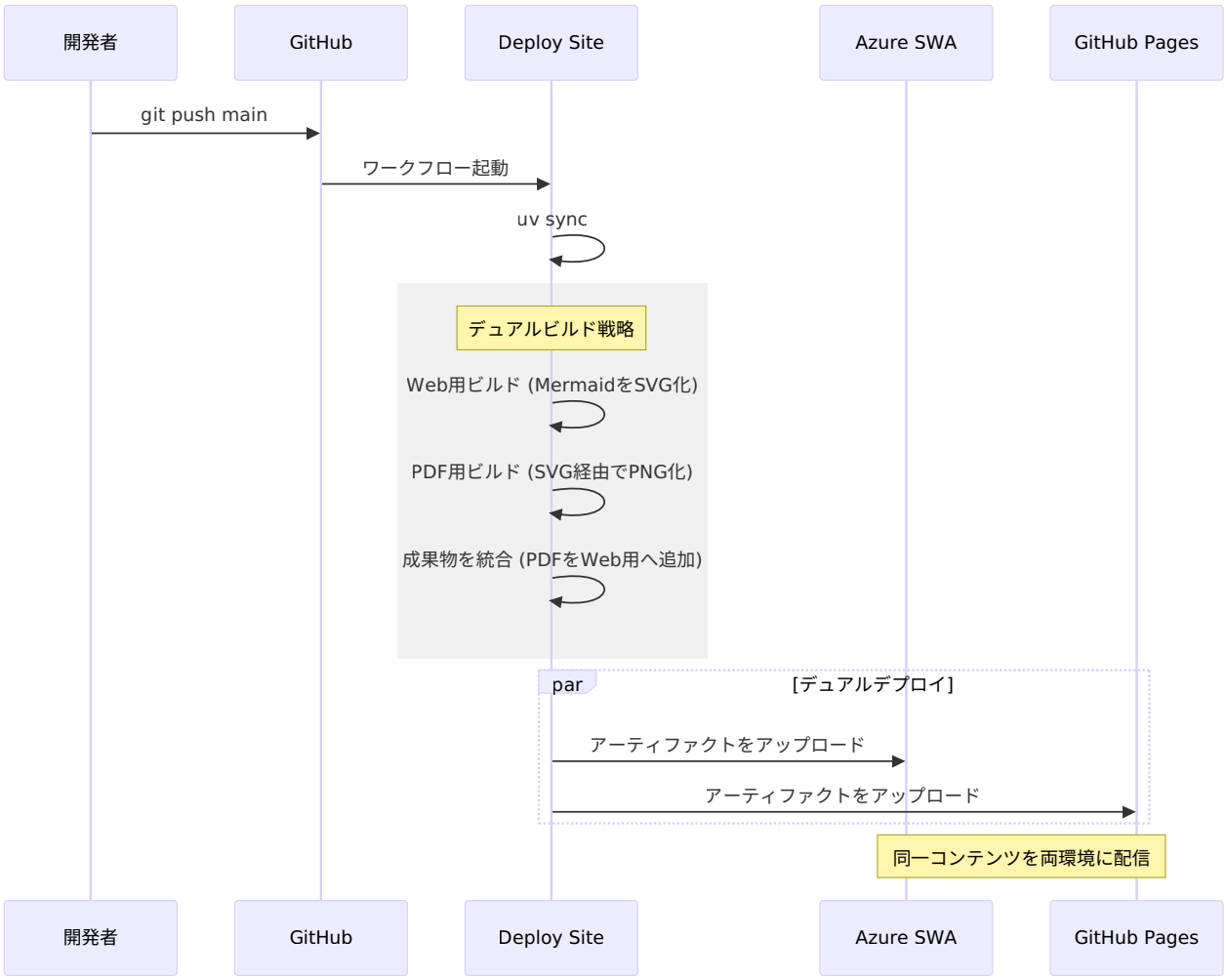
リソース	用途
リポジトリ	Markdown ソース、MkDocs 設定、ワークフロー定義を格納
Deploy Site	CI/CD パイプライン。ビルド・デプロイを実行
Sync Role	ロール同期ジョブ。閲覧権限の管理を自動化
GitHub Pages	静的サイトのホスティング。Private リポジトリ + Enterprise で認証制御可能
GitHub Discussions	SWA の招待管理。Sync Role が生成した招待 URL を登録
GitHub App	Discussions API への書き込み権限を持つアプリケーション

Azure 側リソース

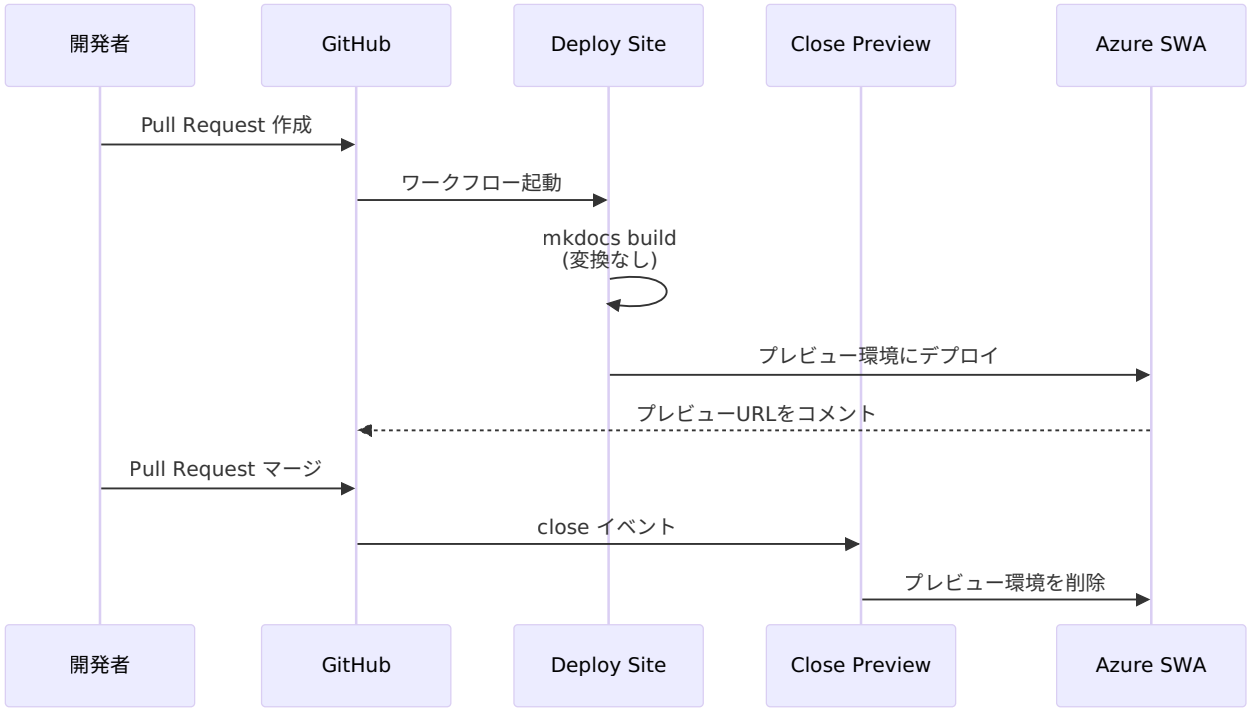
リソース	用途
Resource Group	関連リソースをグルーピング
Static Web App	静的サイトのホスティング。認証・認可機能を内蔵
Managed Identity	GitHub Actions が Azure を操作するための ID
Federated Credential	OIDC による GitHub Actions との信頼関係
RBAC	Managed Identity に SWA への Contributor 権限を付与

3.2.4 デプロイフロー

本番デプロイ（main ブランチ）



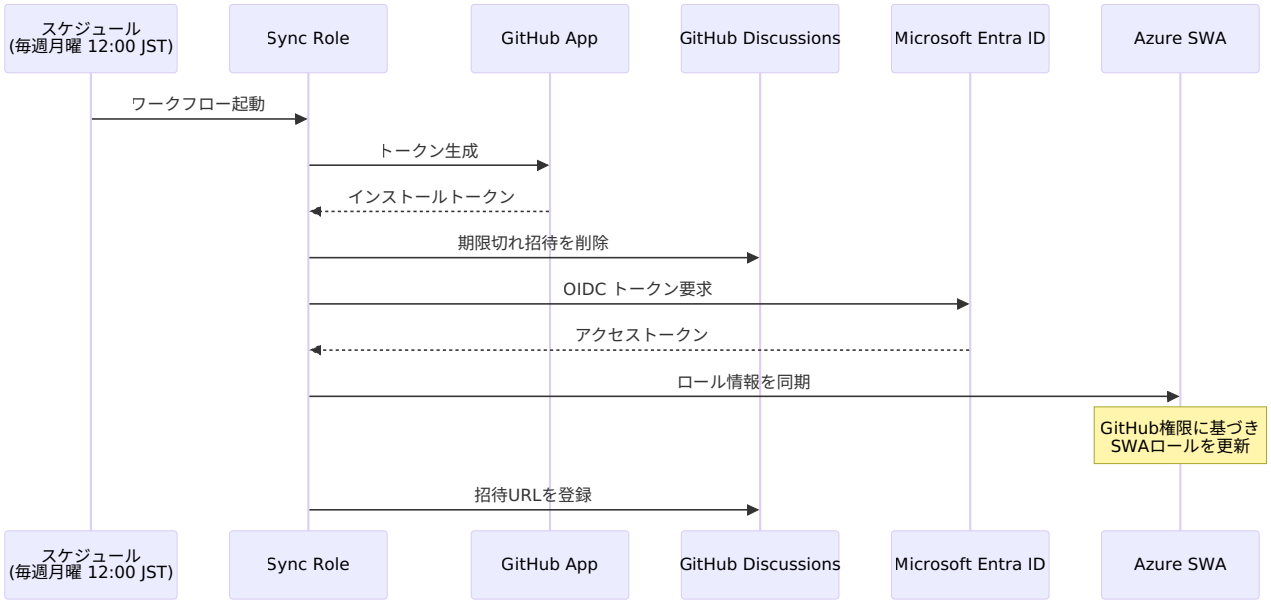
PR プレビュー



注意:

- fork からの Pull Request では `AZURE_SWA_API_TOKEN` が利用できず、SWA のプレビュー作成/削除に失敗する可能性がある。SWA プレビューを有効にする場合は「同一リポジトリ内 PR」を前提とし、必要に応じてワークフロー側で fork PR をスキップする。

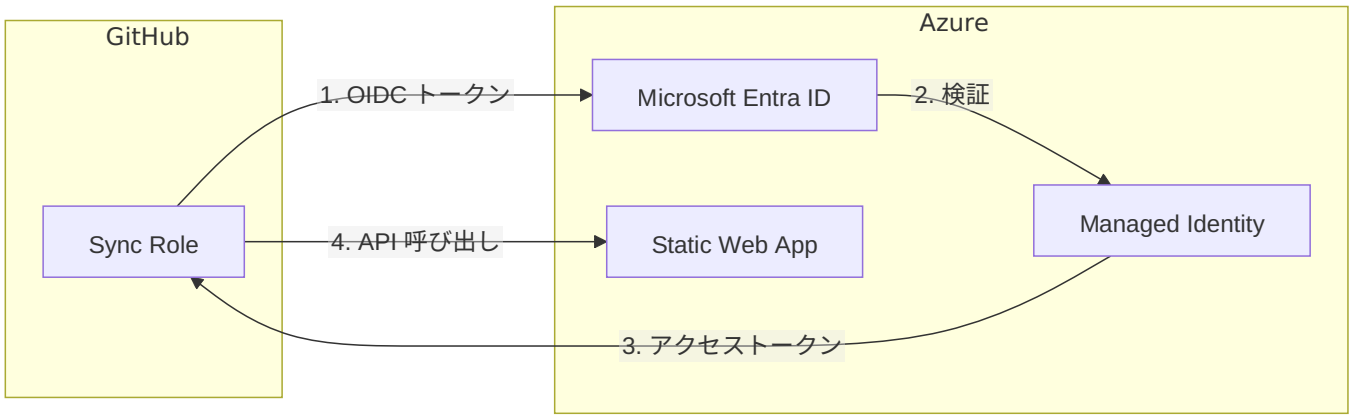
ロール同期



3.2.5 認証・認可モデル

OIDC フェデレーション

Sync Role の Azure への認証には OIDC（OpenID Connect）を使用する。これにより Azure 資格情報（クライアントシークレット等）の長期保存が不要となる。



SWA デプロイの認証

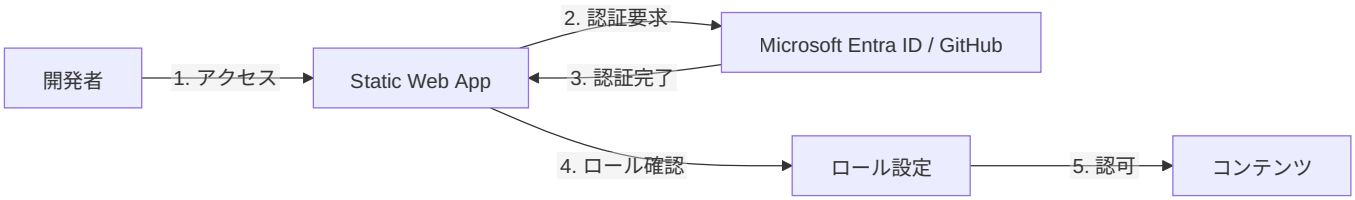
SWA へのデプロイと PR プレビューの削除には `AZURE_SWA_API_TOKEN`（GitHub Secrets）を使用する。

信頼関係の設定:

- **Issuer**: `https://token.actions.githubusercontent.com`
- **Subject**: `repo:{owner}/{repo}:ref:refs/heads/main`
- **Audience**: `api://AzureADTokenExchange`

SWA 認証フロー

Azure Static Web Apps は組み込みの認証機能を提供する。



3.2.6 環境構成の比較

項目	GitHub Pages	Azure Static Web Apps
用途	**正式公開（マージ後）**	**PRプレビュー / 開発用**
URL	`{owner}.github.io/{repo}`	`*.azurestaticapps.net`
認証	リポジトリ権限（Enterprise）	Microsoft Entra ID / GitHub / カスタム
閲覧人数制限	**制限なし**（リポジトリ権限に従う）	**最大 25 人**（カスタムロール使用時）
PRプレビュー	なし	あり（自動生成）
カスタムドメイン	可能	可能
料金	無料	Free / Standard

3.2.7 関連ドキュメント

- ワークフロー アーキテクチャ - GitHub Actions ワークフローの詳細
- クラウド環境構築 - Azure / GitHub リソースの構築手順
- テキスト校正 - textlint による品質管理

3.3 テキスト校正

このリポジトリーでは、textlintの設定と依存関係を単一の基点に集約し、VS Code・CLI・CIの3経路で同一ルールの校正を実現している。

3.3.1 アーキテクチャーの要点

- ルールの単一ソース化（`.textlintrc.json`）
- 除外対象の一元管理（`.textlintignore`）
- 依存関係の固定（`package.json` と `pnpm-lock.yaml`）
- 実行経路の分離とルール共通化

3.3.2 共通ルールの中核

textlint本体と各種ルールセットは `package.json` に集約し、実行時は `.textlintrc.json` を必ず読み込む設計である。これにより、エディター内とCLI/CIでの差分を作らない。

```
ルール定義: .textlintrc.json
除外設定: .textlintignore
依存固定: package.json + pnpm-lock.yaml
```

3.3.3 利用シーン別の実行経路

1. VS Codeのリアルタイム校正

VS Code拡張は `.vscode/settings.json` で `.textlintrc.json` を明示参照している。入力時（`onType`）にtextlintが走り、ルールはローカルの `node_modules` と設定ファイルから解決される。

- 設定参照の明示（`textlint.configPath: ".textlintrc.json"`）
- 実行タイミングの統一（`textlint.run: "onType"`）
- 自動修正の無効化（`textlint.autoFixOnSave: false`）

2. CLIによるローカル全文書の校正

`pnpm run lint:text` がtextlintを起動し、`"**/*.md"` を対象に校正する。CLIも `.textlintrc.json` と `.textlintignore` を同じく参照するため、VS Codeと同一ルールで検出される。自動修正を適用する場合は `pnpm run lint:text:fix` を使う。

```
pnpm run lint:text      # 校正のみ
pnpm run lint:text:fix  # 自動修正を適用
```

3. CI（GitHub Actions）による校正

`.github/workflows/textlint.yml` がPR/Push時にtextlintを実行する。`pnpm install --frozen-lockfile` で依存を固定し、`pnpm run lint:text` でローカルと同一コマンドを実行するため、CIでも同じルールが保証される。

```
pnpm install --frozen-lockfile
pnpm run lint:text
```

3.3.4 同一ルールを維持する設計

3つの経路はいずれも `.textlintrc.json` を参照し、依存は `pnpm-lock.yaml` で固定される。変更点を一か所に集約することで、エディター・CLI・CIの結果が一致する構成になっている。

4. サンプル

4.1 サンプル

このセクションでは、本プロジェクトで使用する各種ツールの利用方法とサンプルを紹介する。

4.1.1 ツールの使い分け

Mermaid（推奨）

テキストベースで図を記述するツールである。以下の理由から、基本的にはMermaidの使用を推奨する。

- AIが解釈・生成しやすい
- バージョン管理が容易（差分が見やすい）
- コードレビューがしやすい

Draw.io（限定的に使用）

VS CodeのDraw.io Integration拡張を使用する。

使用する場面

- Mermaidでは表現が難しい複雑な図
- 接続線の位置を細かく制御したい場合
- 自由なレイアウトが必要な場合

注意

AIの解釈が困難になるため、どうしても必要な理由があるときのみ使用すること。

Marp

プレゼンテーション資料を作成するツールである。MkDocsと統合してWebで表示し、PDFはMkDocsのto-pdfプラグインで統合的に出力される。

4.1.2 サンプルページ

- Mermaid - フローチャート、シーケンス図、状態遷移図などのサンプル
- Draw.io - VS Code拡張を使ったSVG図の作成方法
- Marp - プレゼンテーション資料の作成方法

4.2 Draw.io

Draw.ioはGUIベースで図を作成できるツールである。VS CodeのDraw.io Integration拡張を使用して、エディター内で直接編集できる。

4.2.1 使用する場面

Draw.ioは以下のような場面で使用する。

- Mermaidでは表現が難しい複雑な図
- 接続線の位置を細かく制御したい場合
- 自由なレイアウトが必要な場合
- アイコンやイラストを含む図

4.2.2 注意事項

Draw.ioはどうしても必要な理由があるときのみ使用すること。

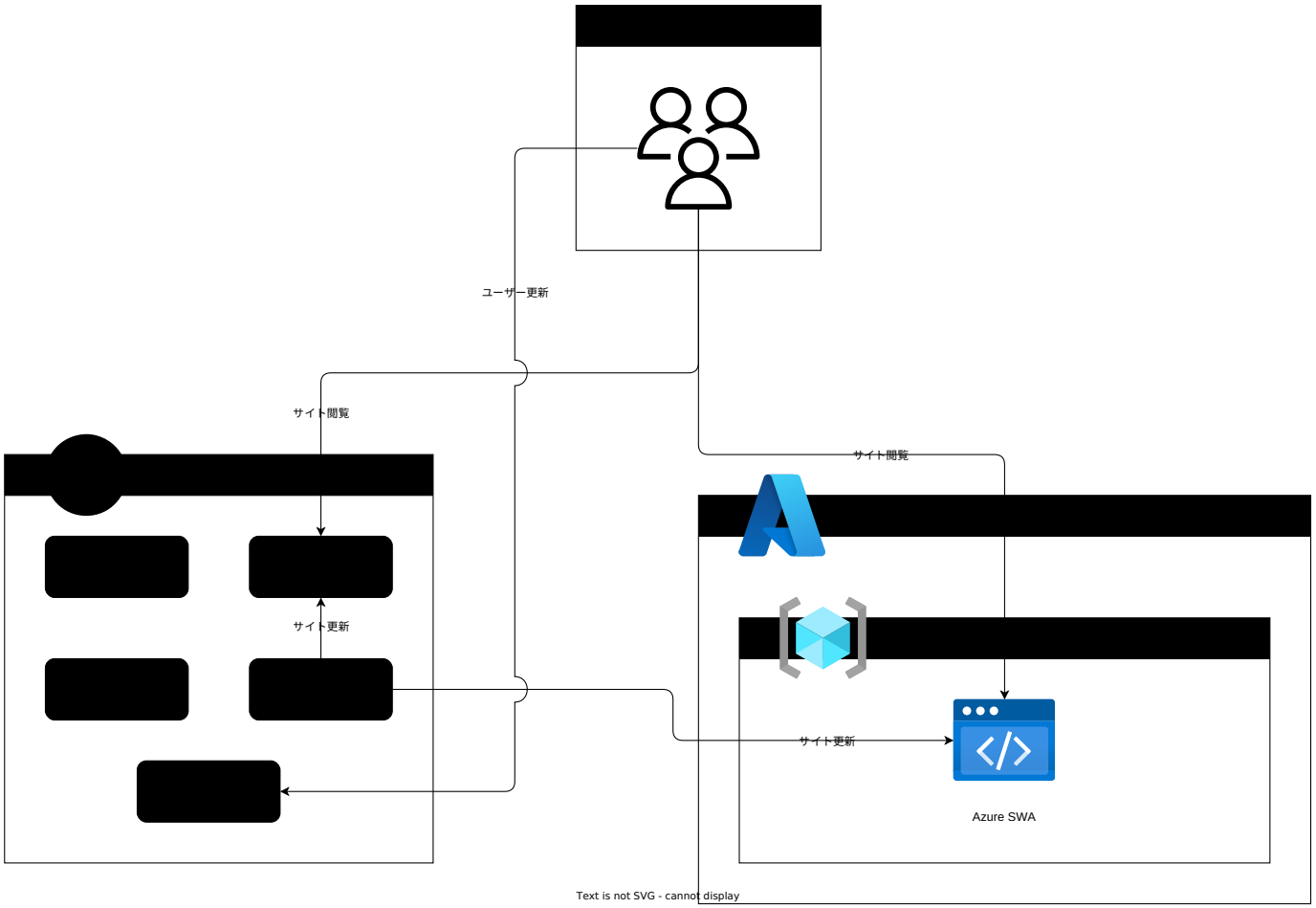
理由：

- バイナリ形式に近いため、AIによる解釈・生成が困難
- 差分の確認が難しい
- テキストベースのMermaidと比較してバージョン管理が複雑

基本的な図（フローチャート、シーケンス図、状態遷移図など）はMermaidを使用すること。

4.2.3 サンプル

以下はDraw.ioで作成した図の例である。



4.2.4 セットアップ

VS Code拡張のインストール

1. VS Codeを開く
2. 拡張機能マーケットプレイスで「Draw.io Integration」を検索
3. 「Draw.io Integration」（hediet.vscode-drawio）をインストール

4.2.5 ファイル形式

Draw.ioで作成するファイルは `.drawio.svg` 形式を使用する。

SVG形式を使用する理由：

・ベクター形式のため、Webで閲覧しやすく拡大縮小が可能

・PDF化する際に、MkDocsのsvg-to-pngプラグインでPNGに変換できる

4.2.6 使い方

新規作成

1. 対象の文書と同じディレクトリに `.drawio.svg` ファイルを作成
2. 例： `overview.md` の図なら `overview-architecture.drawio.svg`
3. VS Codeでファイルを開くと、Draw.ioエディターが起動
4. 図を作成・編集

5. 保存（Ctrl+S）

Markdownでの参照

同じディレクトリにあるため、相対パスで簡潔に参照できる。

```
![アーキテクチャ図](./overview-architecture.drawio.svg)
```

4.2.7 ファイル配置のルール

Draw.ioファイルは、関連する文書と同じディレクトリに配置する。

```
docs/
├── section-a/
│   ├── overview.md
│   ├── overview-architecture.drawio.svg
│   └── detail/
│       ├── api-design.md
│       └── api-design-flow.drawio.svg
```

命名規則

`{文書名}-{図の内容}.drawio.svg`

- 文書名： 関連するMarkdownファイルの名前（拡張子なし）
- 図の内容： 図が表す内容を簡潔に表現
- 例： `api-design-flow.drawio.svg`, `system-architecture.drawio.svg`

この配置方式の理由

1. **近接性**: 文書と関連図が同じディレクトリにあり、関係が明確
2. **移動・削除の容易さ**: 文書を移動・削除する際、関連ファイルも一緒に扱える
3. **命名による関連付け**: ファイル名で所属する文書が明確
4. **スケーラビリティ**: 階層が深くなっても管理が破綻しない
5. **git追跡可能**: `docs/assets/images/` はMkDocs生成画像用でgitignore対象のため、文書と同じ場所に配置

4.2.8 参考リンク

- Draw.io公式サイト
- VS Code Draw.io Integration

4.3 Marp

Marpはマークダウンからプレゼンテーション資料を作成するツールである。本プロジェクトではMkDocsと統合してWebで表示する。

4.3.1 概要

MarpはMarkdown形式でスライドを記述し、HTML/PDF/PPTXなどに変換できるツールである。

本プロジェクトでの特徴：

- MkDocsのページとしてWeb上で閲覧可能
- PDF出力はMkDocsのto-pdfプラグインで統合的に行う
- スライド単体のPDF出力も可能

4.3.2 基本構文

スライドの区切り

スライドは `---`（水平線）で区切る。

```
---
marp: true
---

# スライド1

最初のスライドの内容

---

# スライド2

2枚目のスライドの内容

---

# スライド3

3枚目のスライドの内容
```

フロントマター

スライドの先頭にフロントマターを記述して設定を行う。

```
---
marp: true
theme: default
paginate: true
header: 'ヘッダーテキスト'
footer: 'フッターテキスト'
---
```

主な設定項目：

設定	説明
`marp: true`	Marpスライドとして認識させる
`theme`	テーマ（default, gaia, uncover）
`paginate`	ページ番号を表示
`header`	ヘッダーテキスト
`footer`	フッターテキスト
`size`	スライドサイズ（16:9, 4:3）

4.3.3 スタイリング

ディレクティブ

スライドごとにスタイルを変更できる。

```
---

<!-- _class: lead -->
# タイトルスライド

中央寄せのリードスライド

---

<!-- _backgroundColor: #123 -->
<!-- _color: white -->
# 背景色を変更

このスライドは背景が暗い

---
```

画像の配置

```
---

# 画像の配置

![width:300px](../assets/images/my-diagram.drawio.svg)

<!-- 背景画像として使用 -->
![bg right:40%](../assets/images/background.png)

---
```

画像の指定方法：

- `width:300px` - 幅を指定
- `height:200px` - 高さを指定
- `bg` - 背景画像として使用
- `bg right:40%` - 右側40%に背景画像

4.3.4 サンプルスライド

marp-sample.htmlは、以下のテキストをMarkdownファイルとして保存し、Marpで作成したサンプルスライドをHTML形式に変換したものである。

```
---
marp: true
theme: default
paginate: true
---

# プロジェクト概要
```

```
生成AI時代のドキュメント基盤

---

## 技術スタック

- MkDocs + Material for MkDocs
- Mermaid（ダイアグラム）
- Draw.io（複雑な図）
- Marp（プレゼンテーション）

---

## まとめ

- テキストベースでドキュメント管理
- バージョン管理が容易
- 自動ビルド・デプロイ
```

4.3.5 VS Code拡張

Marp for VS Code拡張を使用すると、エディター内でプレビューを確認できる。

1. VS Codeを開く
2. 拡張機能で「Marp for VS Code」を検索
3. インストール

4.3.6 PDF出力

MkDocs統合でのPDF

MkDocsサイト全体をPDF化する場合は、to-pdfプラグインを使用する。

```
MKDOCS_PDF=1 uv run mkdocs build
```

スライド単体のPDF

Marp CLIを使用してスライド単体をPDF化できる。

```
# Marp CLIのインストール
npm install -g @marp-team/marp-cli

# PDFに変換
marp slides.md --pdf
```

HTMLに変換

```
marp slides.md --html
```

4.3.7 ファイル配置

Marpスライドはdocsディレクトリ内の任意の場所に配置できる。

```
docs/
├── presentations/
│   ├── project-overview.md
│   └── technical-design.md
└── samples/
    └── marp.md
```

4.3.8 参考リンク

- Marp公式サイト
- Marp for VS Code

- Marpit Markdown

5. スライド

5.1 スライド

本セクションでは、Marpで作成したプレゼンテーション資料を掲載する。

5.1.1 スライド一覧

- Marp [Slide/Markdown] - Marpの基本構文、スタイリング、VS Code連携、出力方法のガイド
- 生成AI時代のドキュメント基盤 [Slide/Markdown] - Markdown中心のドキュメント管理と基盤構築についての発表資料

5.2 生成AI時代のドキュメント基盤

5.2.1 ドキュメントの重要性

生成AIの活用が進み、ドキュメントの重要性の再認識

5.2.2 生成AI活用にも注目

- AWS - Kiro
- GitHub - Spec Kit
- Fission-AI - OpenSpec
- gotalab - cc-sdd

5.3 新たな課題の出現

5.3.1 ドキュメントの2つの課題

- ドキュメント体系
- ドキュメント基盤

5.3.2 ドキュメント体系

ソフトウェア開発プロセスの中で

- どのタイミングで
- 何を文章化し
- どうコードとコラボレーションするか

5.3.3 ドキュメント基盤

- どのように作成し
- どのようにレビューし
- どのように閲覧し
- どのように配布するか？

5.3.4 2つの視点

- ドキュメント体系 : どう構成し、どのように役割分担するか
- **ドキュメント基盤** : どのように作成・レビューし、どう閲覧し、どう配布するか

後者が本発表のスコープです。

5.3.5 本ドキュメント基盤の発端

・7年前、MS Officeから脱却を決意

5.3.6 本ドキュメント基盤の発端

・7年前、MS Officeから脱却を決意
・WordやExcelで「きれいに書く事」に時間をかける事が負担
・複数人で並列作業する事の難しさ
・コードと文書のバージョン同期の難しさ
・平行作業時のレビューの難しさ

5.3.7 6時間かけて書いたWordが崩壊してキレた

5.3.8 Markdownベース文書への移行を決意

5.3.9 本ドキュメント基盤の実績

・大手金融3社のSIプロジェクト
・Markdownベース
・800ページ超
・最長7年の持続性

これらの事例とノウハウを紹介します。

5.3.10 自己紹介

- T.B.D
- T.B.D

5.3.11 ドキュメント基盤要件

- AIフレンドリー - 単純な図の簡便な作成 - 複雑な図を正確に描画 - 表の簡便な作成 - 単一PDF出力
- git管理 - PRベースレビュー - PR時のステージング - CI/CD
- 検索を含めた高い閲覧性 - 高い拡張性 - 十分なセキュリティ - 低コスト - 保守ステージのコスト0

5.4 デモンストレーション

5.4.1 デモの流れ

1. Markdown ファイルの作成・閲覧

2. Mkdocs によるサイト閲覧・PDF出力
3. Mermaid / draw.io による作図
4. Copy To Markdown Excel アドインによる 表の管理
5. ソース と ドキュメント の一元管理
6. レビュー と ステージング環境
7. CI / CD
8. おすすめ３機能（MkDocs の検索 / Csv の埋め込み / Marp によるスライド作成）

5.4.2 PDF 出力の流れ

ドキュメンテーション戦略.pdf

5.5 Marp

5.5.1 概要

MarpはMarkdown形式でスライドを記述し、HTML/PDF/PPTXなどに変換できるツール。

本プロジェクトでの特徴：

・MkDocsのページとしてWeb上で閲覧可能

・PDF出力はMkDocsのto-pdfプラグインで統合的に行う

・スライド単体のPDF出力も可能

5.5.2 基本構文

スライドの区切り

スライドは `---`（水平線）で区切る。

```
# スライド1

最初のスライドの内容

---

# スライド2

2枚目のスライドの内容
```

フロントマター

スライドの先頭にフロントマターを記述して設定を行う。

```
---
marp: true
theme: default
paginate: true
header: 'ヘッダーテキスト'
footer: 'フッターテキスト'
---
```

主な設定項目：

設定	説明
`marp: true`	Marpスライドとして認識させる
`theme`	テーマ（default, gaia, uncover）
`paginate`	ページ番号を表示
`header`	ヘッダーテキスト
`footer`	フッターテキスト
`size`	スライドサイズ（16:9, 4:3）

5.5.3 スタイリング

ディレクティブ

スライドごとにスタイルを変更できる。

```
---

<!-- _class: lead -->
# タイトルスライド

中央寄せのリードスライド

---
```

```
<!-- _backgroundColor: #123 -->
<!-- _color: white -->
# 背景色を変更

このスライドは背景が暗い

---
```

画像の配置

```
![width:300px](../assets/images/my-diagram.drawio.svg)

<!-- 背景画像として使用 -->
![bg right:40%](../assets/images/background.png)
```

画像の指定方法：

- `width:300px` - 幅を指定
- `height:200px` - 高さを指定
- `bg` - 背景画像として使用
- `bg right:40%` - 右側40%に背景画像

5.5.4 VS Code拡張

Marp for VS Code拡張を使用すると、エディター内でプレビューを確認できる。

1. VS Codeを開く
2. 拡張機能で「Marp for VS Code」を検索
3. インストール

5.5.5 PDF出力

MkDocs統合でのPDF

MkDocsサイト全体をPDF化する場合は、to-pdfプラグインを使用する。

```
MKDOCS_PDF=1 uv run mkdocs build
```

スライド単体のPDF

Marp CLIを使用してスライド単体をPDF化できる。

```
# Marp CLIのインストール
npm install -g @marp-team/marp-cli

# PDFに変換
marp slides.md --pdf
```

HTMLに変換

```
marp slides.md --html
```

5.5.6 参考リンク

- Marp公式サイト
- Marp for VS Code
- Marpit Markdown

5.6 Mermaid

Mermaidはテキストベースで図を記述できるツールである。本プロジェクトでは推奨ツールとして位置づけている。

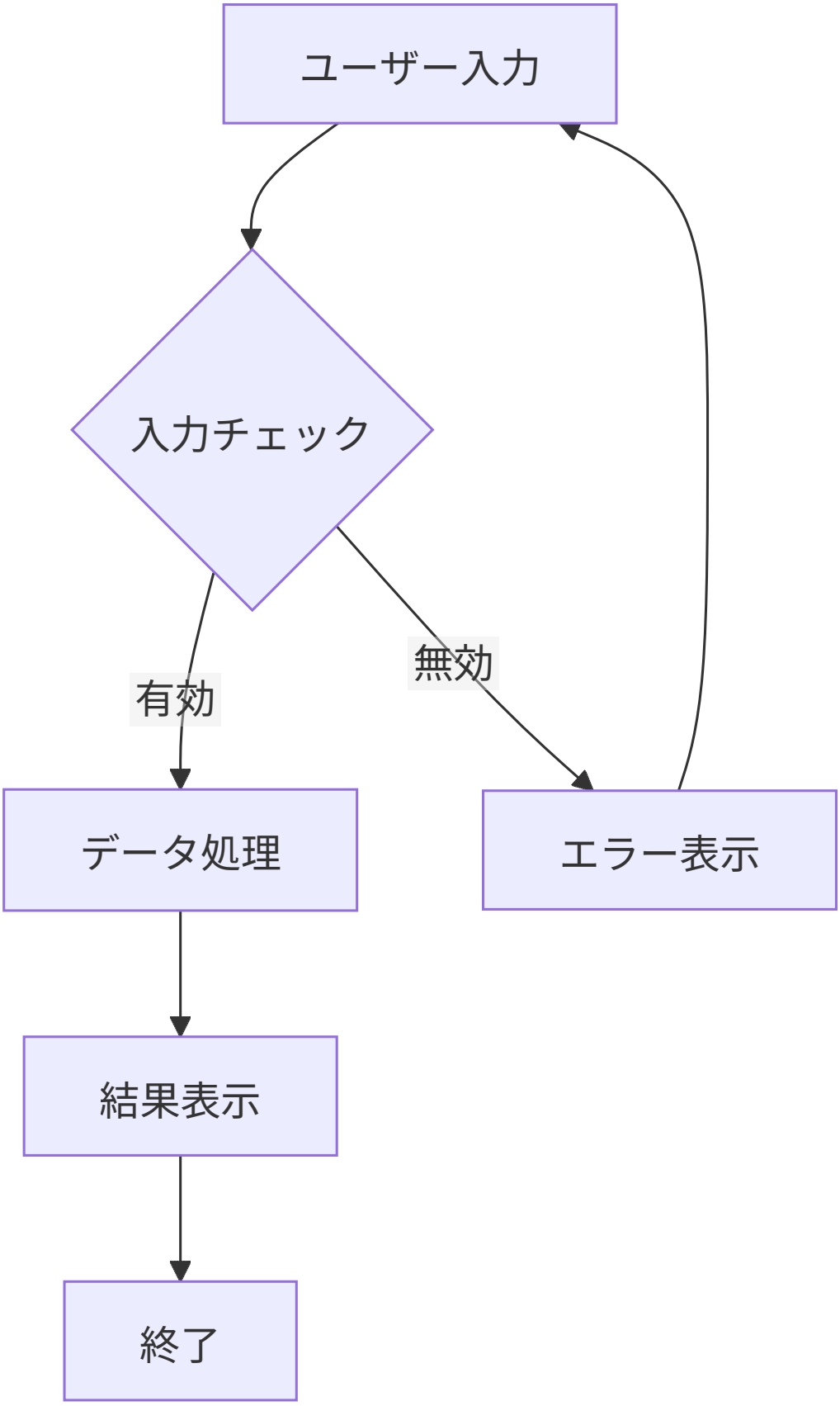
5.6.1 基本的な使い方

Markdownファイル内で、以下のようにコードブロックを記述する。

````
```mermaid
graph TD
    A[開始] --> B[処理]
    B --> C[終了]
```
````

5.6.2 フローチャート

処理の流れを表現する基本的な図である。

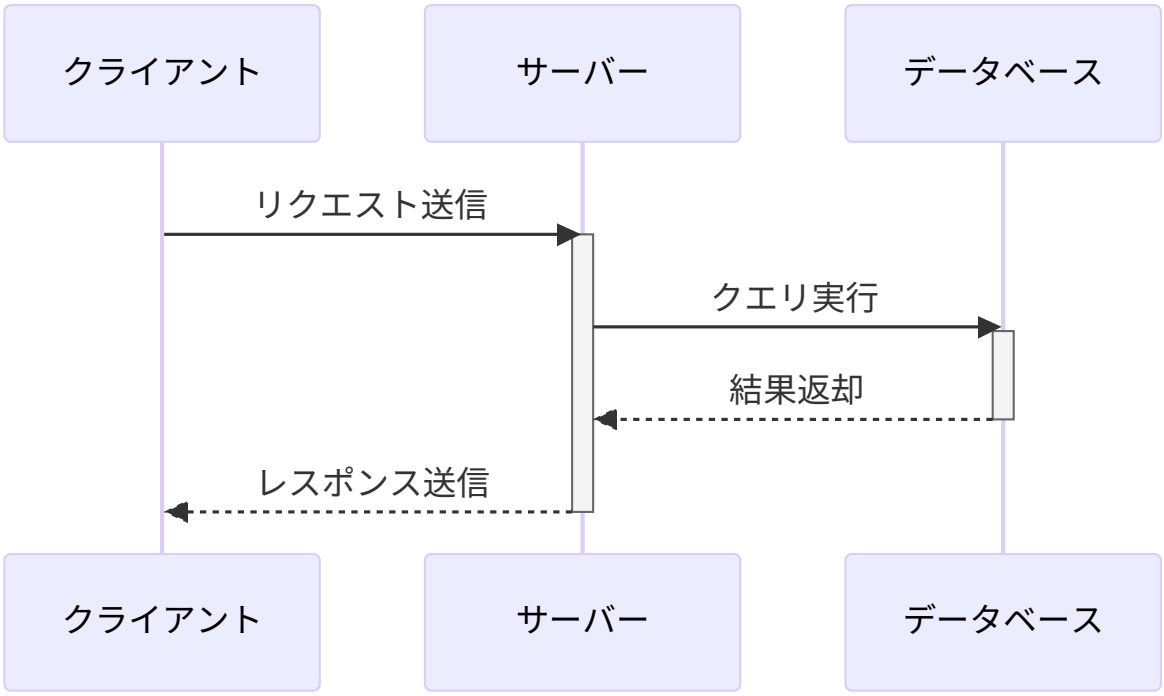


記法のポイント

- `graph TD`: 上から下へ流れる図（Top to Down）
- `graph LR`: 左から右へ流れる図（Left to Right）
- `[テキスト]`: 四角形のノード
- `{テキスト}`: ひし形のノード（条件分岐）
- `-->`: 矢印
- `-->|ラベル|`: ラベル付き矢印

5.6.3 シーケンス図

オブジェクト間のやり取りを時系列で表現する。

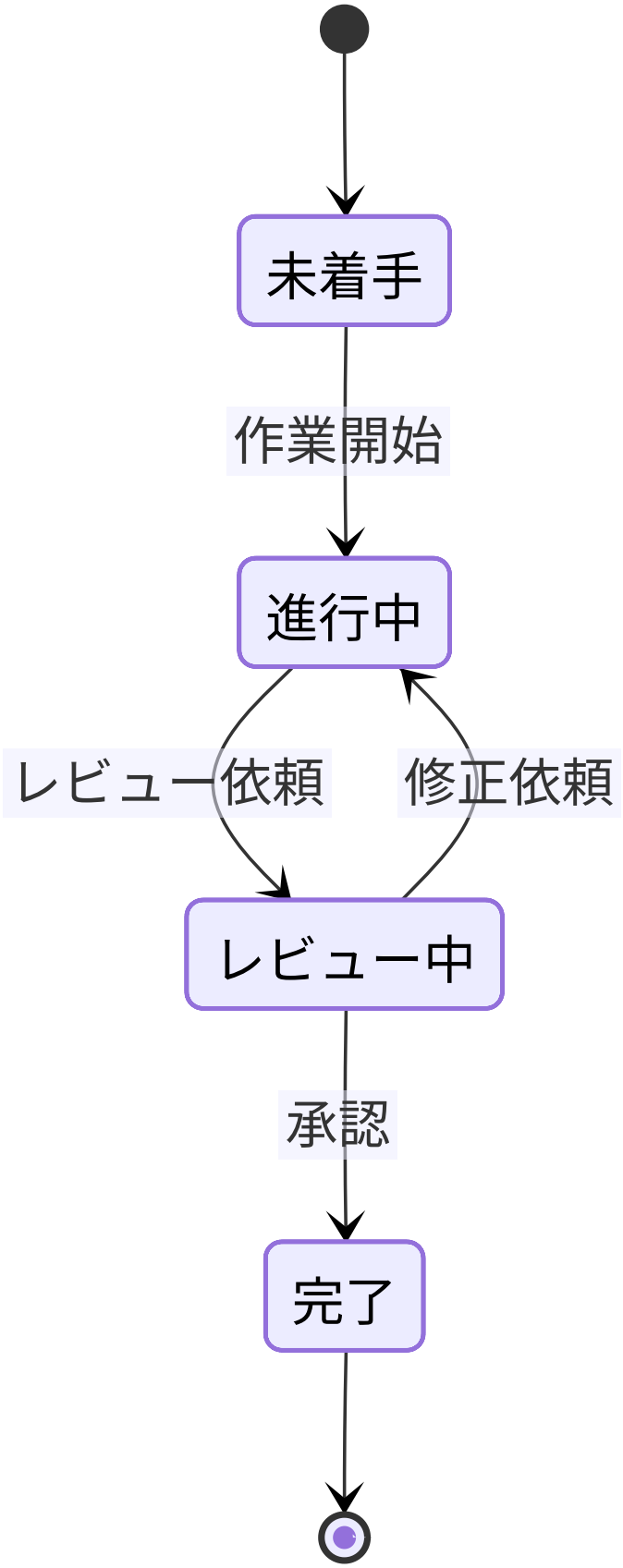


記法のポイント

- `participant`: 参加者を定義
- `->>`: 同期メッセージ
- `-->>`: 応答メッセージ
- `activate/deactivate`: 活性化バーの表示

5.6.4 状態遷移図

システムやオブジェクトの状態の変化を表現する。

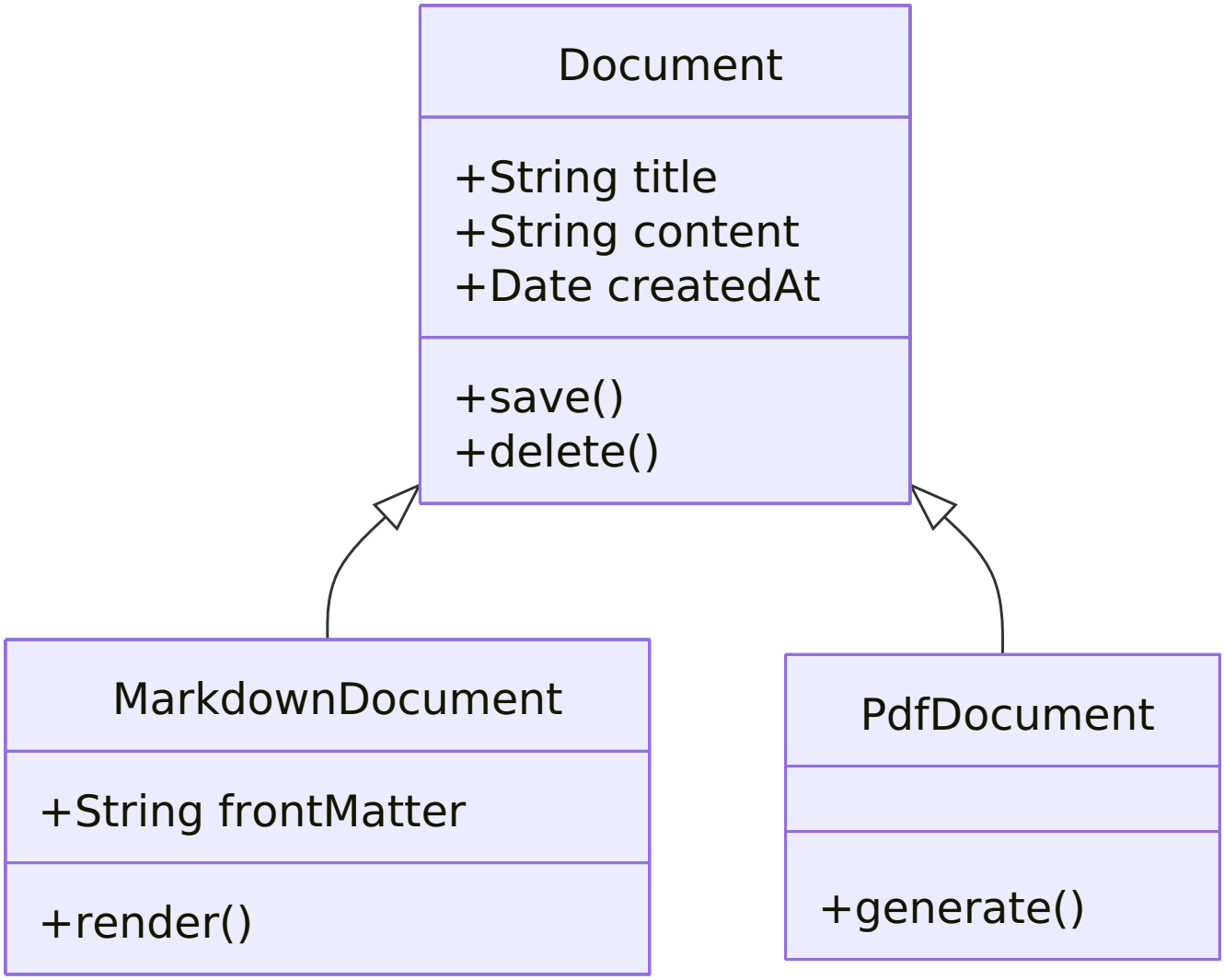


記法のポイント

- `[*]`: 開始・終了状態
- `状態名`: 状態を定義
- `-->`: 遷移

5.6.5 クラス図

クラスの構造と関係を表現する。

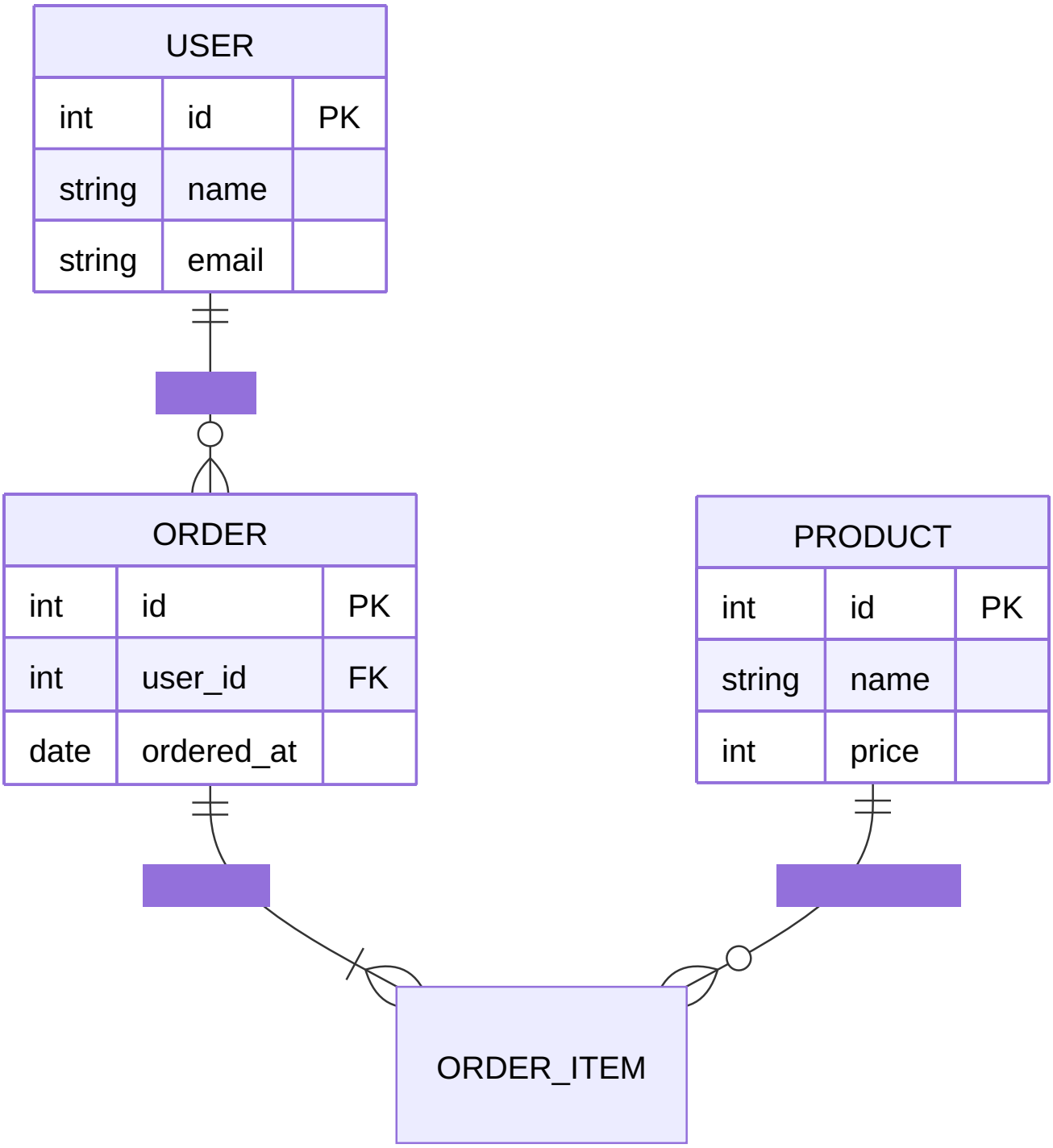


記法のポイント

- `class クラス名`: クラスを定義
- `+`: public
- `-`: private
- `<|--`: 継承関係

5.6.6 ER図

データベースのエンティティと関係を表現する。

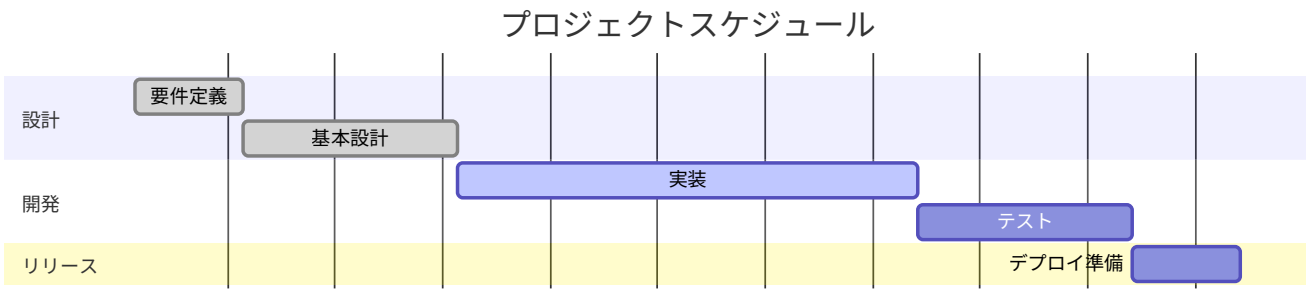


記法のポイント

- `||--o{` : 1対多の関係
- `||--|{` : 1対1以上の関係
- `PK` : 主キー
- `FK` : 外部キー

5.6.7 ガントチャート

プロジェクトのスケジュールを表現する。



5.6.8 参考リンク

- Mermaid公式ドキュメント
- Mermaid Live Editor