

AI AGENTで変わる、 これからの業務アプリ開発

アプリケーション開発プロジェクトに、AI Agentチームを。

AIで業務を変える。その仕組みを作る

アプリケーション開発
プロジェクトに、
AI Agentチームを。



会社名

株式会社ビルドシステム

設立

平成7年11月1日（1995年）

代表者

秦 一雄 代表取締役社長

役員

貝瀬 丈次 取締役

三塩 陽介 取締役（非常勤）

従業員数

54名（2024年7月現在）

所在地

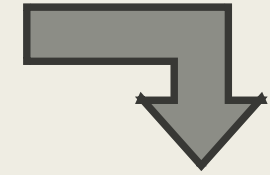
〒101-0022 東京都千代田区神田練塀町3番地 インテージ秋葉原ビル

AI は個々に役立っている？

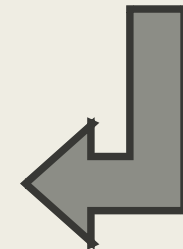
Understand
既存システム解析
Reverse Engineering
現状把握・構造整理



Design
要件整理
仕様・設計案
業務フロー整理



Build
画面・ロジック
API・データ連携
アプリ実装



Test
テスト観点
レビュー・修正
品質確認



Document
仕様書
設計資料
引継ぎドキュメント

生成AI サービス

BUSINESS

AI Agentで 業務を変える

AI Agentが既存システムやデータにつ
ながり、調査・整理・回答案作成などの
仕事を支援します。

「在庫を確認して、納期回答案
を作って」

在庫

受注

顧客

↓ 3つの情報を参照

納期回答案を作成



DEVELOPMENT

AI Agentで その仕組みを作る

要件整理・設計・実装・テスト・ドキュ
メント。複数のAI Agentが役割分担
し、人と一緒に開発を進めます。

CAT
ANALYST

CAT
ARCHITECT

CAT
BUILDER

CAT
QA

CAT
DEVOPS

CAT
WRITER

BUSINESS

AI Agentで 業務を変える

AI Agentが既存システムやデータにつ
ながり、調査・整理・回答案作成などの
仕事を支援します。

「在庫を確認して、納期回答案
を作って」

在庫

受注

顧客

↓ 3つの情報を参照

納期回答案を作成

DEVELOPMENT

AI Agentで その仕組みを作る

要件整理・設計・実装・テスト・ドキュ
メント。複数のAI Agentが役割分担
し、人と一緒に開発を進めます。

CAT
ANALYST

CAT
ARCHITECT

CAT
BUILDER

CAT
QA

CAT
DEVOPS

CAT
WRITER



**AIで業務を変える。
その仕組みを作るのが、私たちの役割で
す。**

algo-Cats | AI Agentが役割分担する



Human Control | AIに丸投げしない



App構築デモ

「簡単なお天気アプリを構築する」

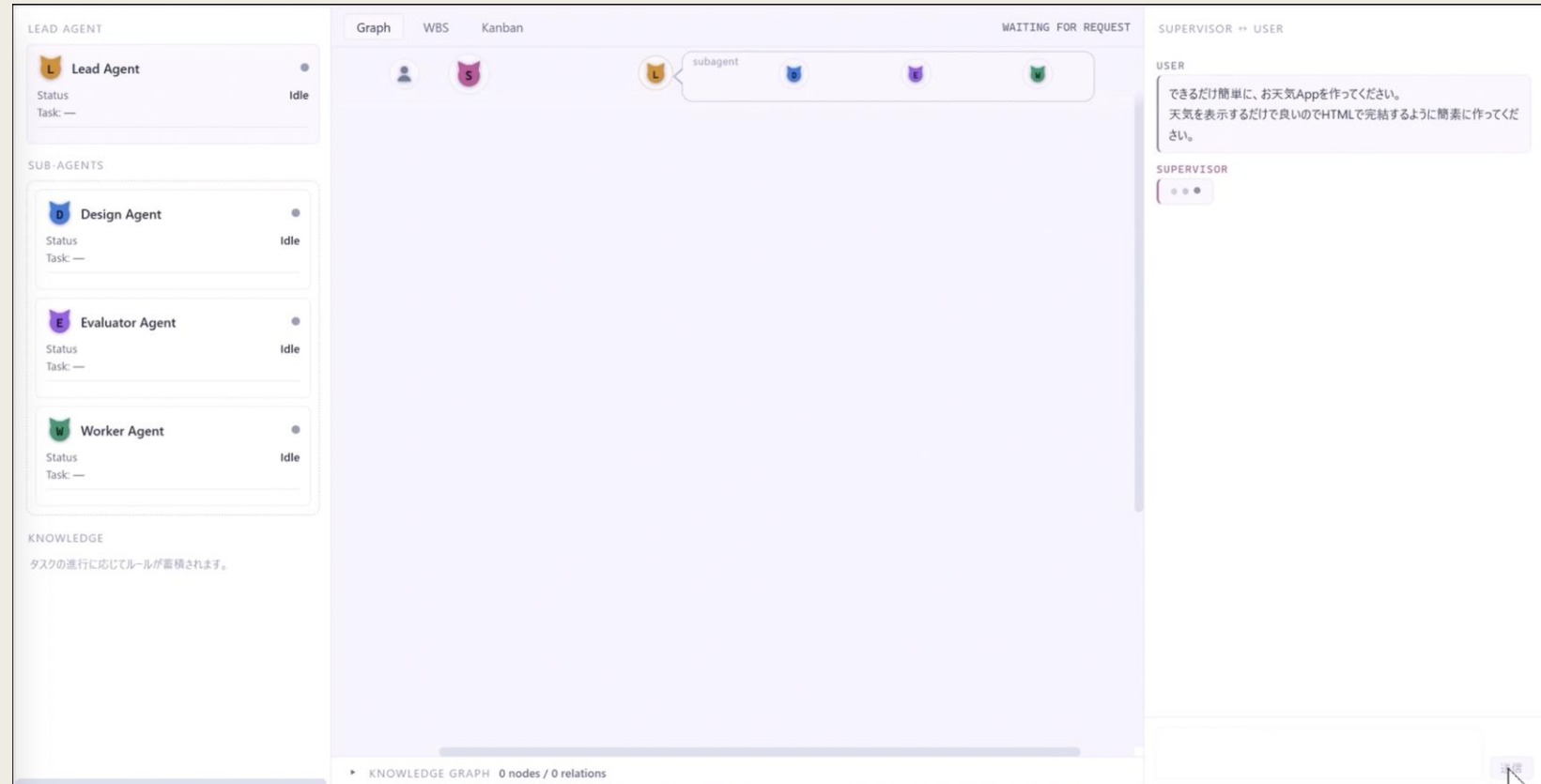
要件 → 設計 → 戦略 → 実装 → 評価 → 完成



Demo 実施

App構築デモの振り返り

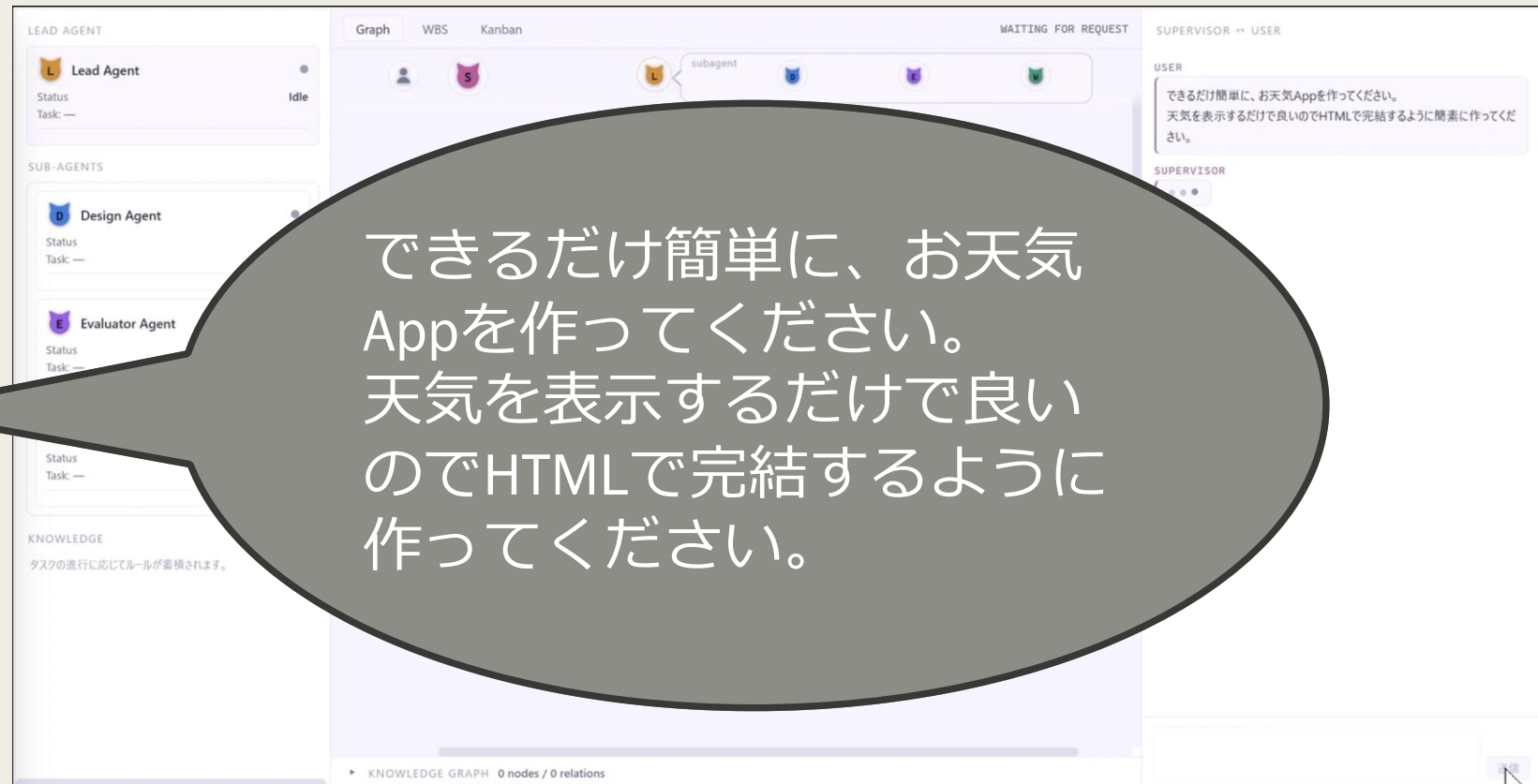
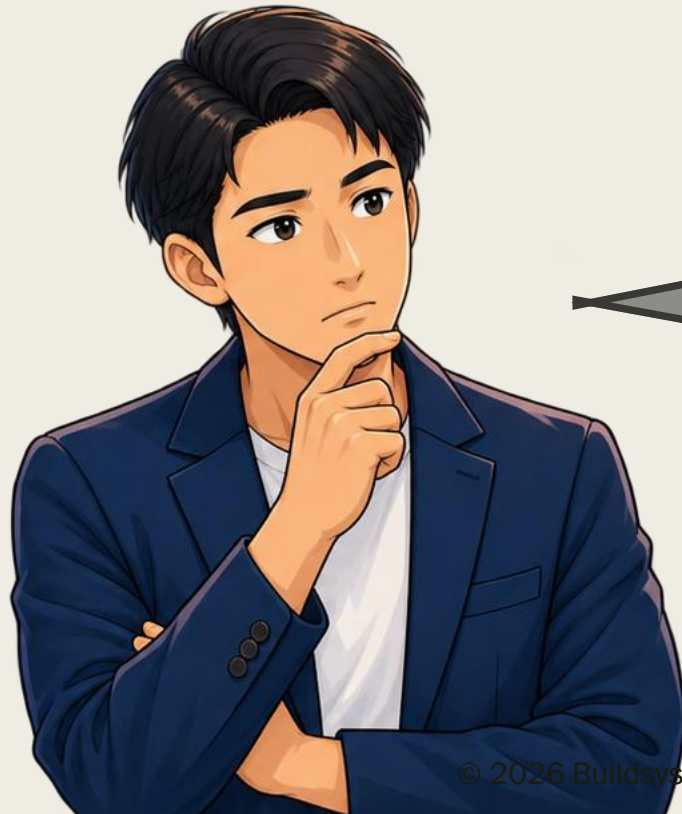
「簡単なお天気アプリを構築する」



まず、要件を AI agent に伝えます。

App構築デモ

「簡単なお天気アプリを構築する」



まず、要件を AI agent に伝えます。

要件定義書

Weather App Project / 生成: Agentic Graph Engineering Demo / 承認: Lead Agent

1. 目的

ユーザ要求「天気情報を表示するWeb Appを作って」に基づき、現在天気・予報を高速に閲覧できるWebアプリケーションを構築する。気象データは [Open-Meteo API](#) (APIキー不要・CORS対応) から実際に取得する。

2. 機能要件

ID	要件	優先度
FR-1	現在天気表示 — 気温 / 天候 / 湿度 / 風速 / 体感温度を表示する	高
FR-2	5日間予報 — 日次の最高/最低気温・天候・降水確率を一覧表示する	高
FR-3	地点検索 — 任意の地点名 (日本語 / 英語) で世界中の天気を検索できる	中
FR-4	お気に入り地点 — 複数地点を登録・切替できる	中
FR-5	自動更新 — 表示中の天気を定期的に再取得する (5分間隔)	低

3. 非機能要件

ID	要件	目標値 / 方式
NFR-1	応答性能	キャッシュヒット時は即描画。取得時はスケルトン表示で1秒以内の体感応答
NFR-2	オフライン耐性	取得失敗・オフライン時は最終取得値をキャッシュ (TTL 10分) から表示
NFR-3	モバイル対応	360px幅までレスポンス表示

4. 前提・制約

- 単一HTMLファイル・ブラウザ内で完結 (サーバ不要)。ネットワーク接続時のみ実データを取得。
- データ提供元: Open-Meteo (ジオコーディング / 天気予報)。利用にあたり出典表記を行う。



アーキテクチャ設計書

Weather App Project / 生成: Agentic Graph Engineering Demo / 承認: Lead Agent

1. 全体構成 (ブラウザ内3層 + 外部API)

UI層	現在天気画面 / 予報画面 (3状態: 通常 / エラー / オフライン)	
calls		
API層	WeatherClient (apiFetch / fetchCity / normalize)	
uses	WMOコード変換・単位正規化	
データ層	Weather / Forecast モデル + Cache (TTL 10分, in-memory)	
HTTPS (fetch)		
外部	Open-Meteo Geocoding API / Forecast API (キー不要・CORS許可)	

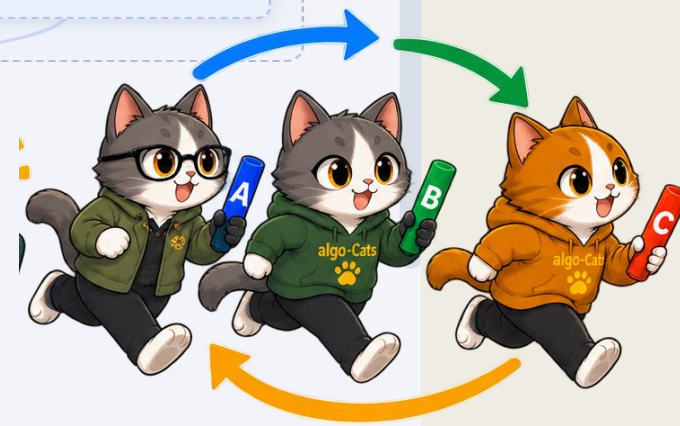
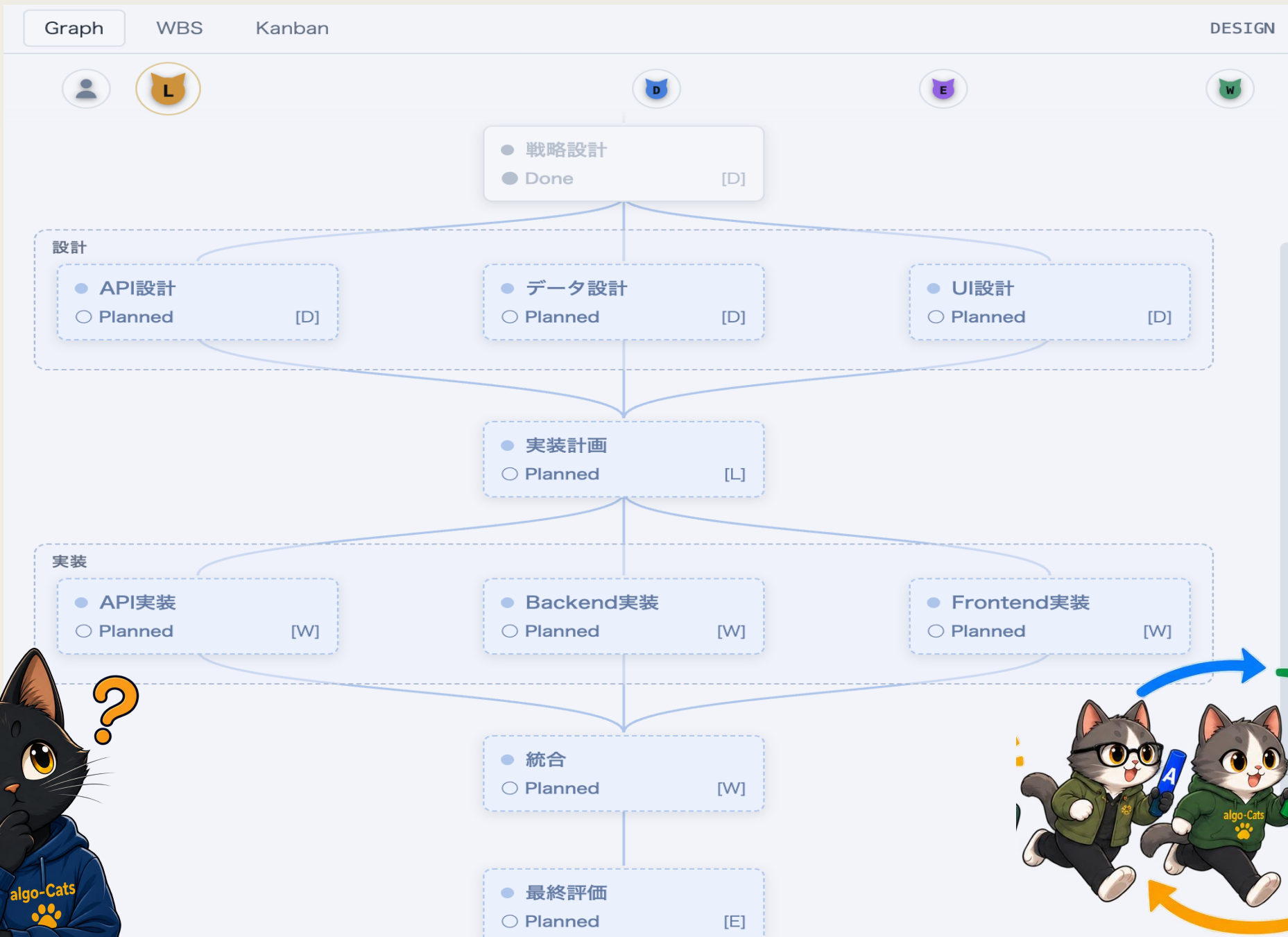
2. 設計方針

- サーバレス構成: 3層すべてを単一HTML内のブラウザJSで実装し、外部依存はOpen-Meteoのみとする。
- プロバイダ差し替えを想定し、Open-Meteo固有形式 (weather_code等) は `normalize()` でアプリ内モデルへ変換してからデータ層に載せる。
- データ層にTTL10分のキャッシュを置き、同一地点の連続表示・オフライン時の表示を吸収する (NFR-1 / NFR-2)。
- データフローは **UI** → `WeatherClient` → `Open-Meteo` → `正規化` → `キャッシュ` → **UI反映** の一方向とする。

3. コンポーネント責務

層	コンポーネント	責務
API層	apiFetch	HTTPアクセス・共通エラー変換・指数バックオフリトライ (最大3回)
API層	fetchCity	ジオコーディング → 予報取得の直列オーケストレーション
API層	normalize / wmoToCond	WMO天候コード・単位のアプリ内モデルへの正規化
データ層	Weather / Forecast	正規化済みモデル (詳細はデータ設計書)
データ層	Cache	地点名キーのTTL管理・ヒット率カウント・期限切れ値のstale返却
UI層	load / render	3状態制御・スケルトン・キャッシュフォールバック・自動更新 (5分)





API設計書

Weather App Project / 生成: Agentic Graph Engineering Demo / 承認: Lead Agent

1. 利用する外部API (Open-Meteo)

API	エンドポイント	用途
Geocoding	GET geocoding-api.open-meteo.com/v1/search?name={地名}&count=1&language=ja&format=json	地点名 → 緯度経度・正式名称。日本語地名がヒットしな「サフィックス除去 (大阪府→大阪) → ローマ字対応表 (→Tokyo)」の順に候補を再検索する
Forecast	GET api.open-meteo.com/v1/forecast latitude / longitude current=temperature_2m, relative_humidity_2m, apparent_temperature, weather_code, wind_speed_10m hourly=temperature_2m daily=weather_code, temperature_2m_max, temperature_2m_min, precipitation_probability_max timezone=auto / forecast_days=5 / wind_speed_unit=ms	現在天気・時間別気温・5日間予報を1リクエストで取得

いずれもAPIキー不要・CORS許可 (Access-Control-Allow-Origin: *) のため、ブラウザから直接呼び出す。

2. 共通エラースキーマ (Evaluator指摘により追加 / クライアント側で変換)

<pre>{ "code": "LOCATION_NOT_FOUND", "message": "指定された地点が見つかりません", "retriable": false}</pre>			
code	発生条件	retriable	挙動
LOCATION_NOT_FOUND	Geocodingの結果0件	false	エラー画面 + 「東京に戻る」
NETWORK_UNAVAILABLE	ネットワーク断 / HTTP 5xx	true	指数バックオフで最大3回リトライ → 失敗時はキャッオールバック
評価			
APIエンドポイント定義			NG — エラーレスポンス定義
エラーレススキーマ・リトライ指針を追加			合格
Open-Meteo) 接続版へ更新。自前サーバ案を廃しブラウザ直接呼出に変更			合格

データ設計書

Weather App Project / 生成: Agentic Graph Engineering Demo / 承認: Lead Agent

1. Weather モデル (現在天気)

フィールド	型	由来 (Open-Meteo)	説明
name	string	geocoding name + admin1	表示用地点名 (例: 大阪 (大阪府))
temp	number	current.temperature_2m	気温 (°C)
hum	number	current.relative_humidity_2m	湿度 (%)
wind	number	current.wind_speed_10m (m/s指定)	風速 (m/s)
feels	number	current.apparent_temperature	体感温度 (°C)
cond	enum	current.weather_code → 変換	正規化済み天候 (下表)
hourly	(temps[12], hours[12])	hourly.temperature_2m	現在時刻から2時間刻み12点
observedAt	ISO8601	current.time	観測時刻 (地点ローカル)

2. Forecast モデル (日次 × 5)

フィールド	由来	説明
date	daily.time[i]	対象日
hi / lo	temperature_2m_max / min	最高 / 最低気温 (四捨五入)
cond	daily.weather_code[i] → 変換	天候enum
pop	precipitation_probability_max[i]	降水確率 (%) — 提供なしの地点は null

3. WMO weather_code → 天候enum 変換表

WMOコード	enum	表示
0	sunny	晴れ
1-2	partly	晴れ時々
3, 45, 48 (霧含む)	cloudy	曇り
51-67, 80-82, 95-99 (雷雨含む)	rain	雨
71-77, 85-86	snow	雪

4. Cache

- キー: 検索クエリ (地点名) / 値: 正規化済みWeatherモデル + 取得時刻
- TTL: 10分。期限内はネットワークを介さず即描画。期限切れ値は state フラグ付き (NFR-2)。



UI設計書

Weather App Project / 生成: Agentic Graph Engineering Demo / 承認: Lead Agent

1. 画面構成

現在天気画面

23°C

☀ 晴れ

東京 / 湿度48% / 風速3.4m/s

時間別推移 (折れ線)



予報画面

今日 ☀ 10% 24 / 15°C

明日 ☁ 30% 22 / 14°C

水曜 ☂ 80% 19 / 13°C

木曜 ☁ 40% 21 / 14°C

金曜 ☀ 10% 25 / 16°C

2画面はタブで相互遷移する (⇔)。ヘッダに地点検索・お気に入りチップを常設。

2. 表示状態 (3状態)

状態	表示
ローディング	スケルトン表示。1秒以内の応答を想定 (NFR-1)
エラー	共通エラーレスポンスの <code>message</code> と再試行ボタン
オフライン	キャッシュ値 + 「最終更新 hh:mm」 バッジ

3. ナビゲーション

現在天気 ⇔ 予報はタブ遷移。地点検索・お気に入りはヘッダから呼び出す。360px幅までレスポンシブ対応。



お天気App



● オンライン

地点名で検索 (例: 大阪 / Paris)

検索

東京

大阪

現在天気

5日間予報

東京都 ★

22°C

雨



観測 01:00 (現地時刻) / 自動更新中

湿度

98%

風速

0.51 m/s

体感

27°C

時間別推移 (約24h)

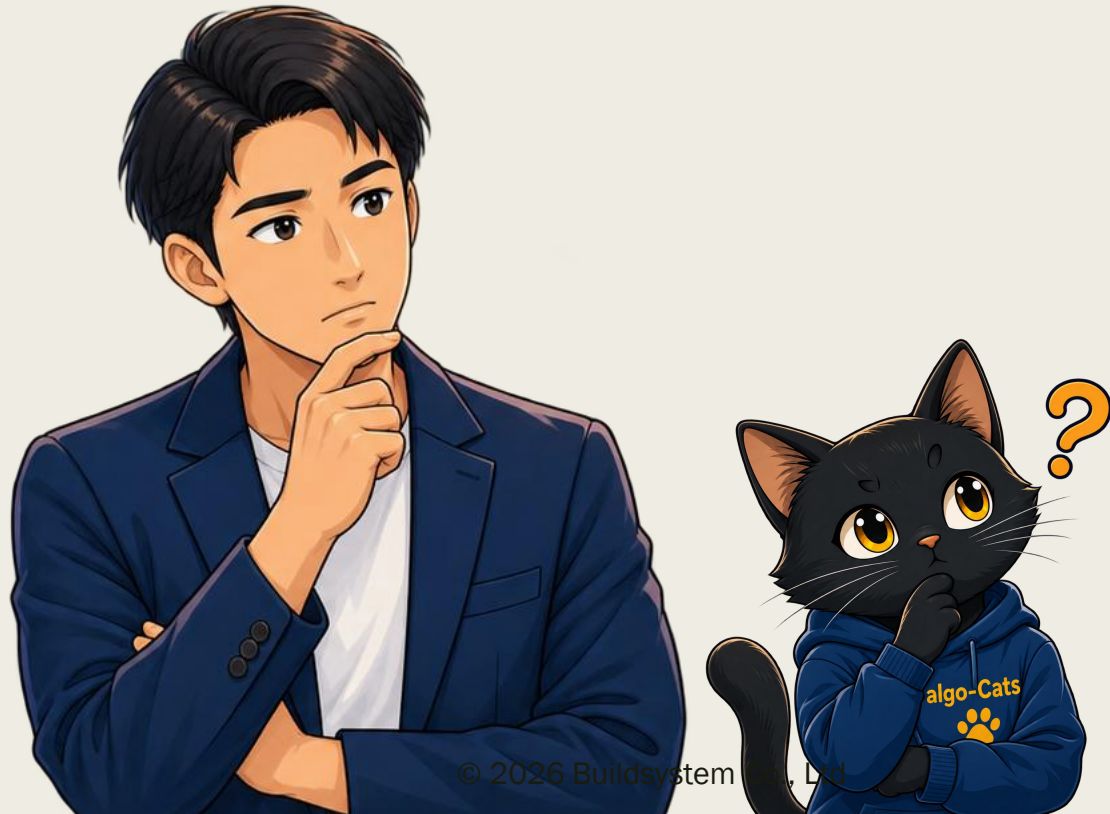


自動更新: 5分ごと / キャッシュ TTL 10分 / データ: Open-Meteo

[← UIに戻る](#)



AI Agentは、新しく作るだけではない。
既存システムを理解する。



Demo 実施

速さだけでは終わらせない。 人と仕組みで守る、品質向上が命。

OUR PRINCIPLE

AI Agentに “丸投げ”しない。

AI Agentは大量の作業を高速に、かつ抜け漏れなく正確に進められます。一方、業務の意味、設計上のトレードオフ、セキュリティ、最終品質には人の判断と責任が必要です。人がマネジメントし、AI Agentが網羅的・正確に実行する。そこに開発品質を高める価値があります。

- 1 業務要件 | 何を実現すべきか
- 2 設計判断 | どの構成を採用するか
- 3 セキュリティ | 守るべき条件は何か
- 4 テスト | 業務で使える品質か
- 5 最終レビュー | 責任を持って判断する

企業アプリケーションのモデルケース

Mendix + Maia

AI Agentで網羅的・正確に開発を進めても、企業システムとして安全に使い続けられなければ 意味がありません。 **Mendixは、開発・実行・運用・管理を一体で支える企業向けアプリケーション基盤**です。



Mendix
アプリを安全に開発・実行・運用できるクラウド開発基盤

開発標準
統一されたモデルと開発

運用・保守
リリース後も継続的に改善

ガバナンス
権限・変更・レビューをルール管理

AI Agentは、開発だけを速くするのではない。

要件整理から設計、開発、テスト、ドキュメントまで。
開発プロジェクト全体の生産性向上を目指します。

	要件・仕様	設計	開発	テスト	ドキュメント
Mendix	人を中心に 進行	人を中心に 進行	×2 開発工程の 生産性向上	人を中心に 進行	人を中心に 進行
AI Agent	×2 要件整理	×2 設計支援	×2 実装	×2 テスト支援	×2 文書化

M

Mendix
開発工程の生産性を高める

AI

AI Agent
開発プロジェクト全体の
生産性を高める

※ AI Agent活用による生産性は従来比約2倍を想定。
対象業務・要件・既存環境等により効果は異なります。

まず、相談から。



「AI Agentに興味はある。でも、
どこから始めればいい？」——その
段階で大丈夫です。自社の業務
に合うテーマと一緒に整理し、
必要に応じてPoCへつなげます。



小間番号AI1-6

にお越しく下さい。