

Tasks: 気温予測アプリケーション

Input: Design documents from [spec/001-weather-forecast-app.js](#). Prerequisites: plan.md, spec.md, research.md, data-model.md, quickstart.md, contracts/open-meteo-api.yaml. Tests: テストはオプション（仕様で明示的に要求された場合のみ生成） Organization: ユーザーストーリー単位でタスクを分類し、独立実装・独立テストを実現

フォーマット: [ID] [P#] [Story] Description

- [P]: 並列実行可能（異なるファイル、依存関係なし）
- [Story]: このタスクが属するユーザーストーリー（例: US1, US2, US3）
- ファイルパスは絶対パスではなく、リポジトリルートからの相対パスで記述

バス規則

本プロジェクトは静的WebアプリケーションのSPA構成:

- ソースコード: [open-meteo/](#) ディレクトリ
 - [index.html](#) - メインHTML
 - [style.css](#) - サイバーパンク風ベーススタイル
 - [script.js](#) - API連携・グラフ描画ロジック
- テスト: [tests/](#) ディレクトリ
 - [tests/unit/](#) - ユニットテスト
 - [tests/integration/](#) - E2Eテスト
- ドキュメント: [docs/](#), [specs/](#), [.specify/](#)

Phase 1: Setup (共通インフラ)

目的: プロジェクト初期化と基本構造の構築

- T001 package.jsonのdevDependencies確認（ Jest 30.x, Puppeteer 24.x, http-server 14.x）
- T002 [P] ignore/node_modules/.coverage/、DS_Storeを追加
- T004 [P] start-app.ps1の動作確認（ポート8080でhttp-server起動）
- T004 ESLint設定ファイル(eslintrc.json)を作成し、ES6+ルールを適用

Phase 2: Foundational (プロトotyping前提条件)

目的: すべてのユーザーストーリーが依存するコアインフラの完成

⚠ CRITICAL: このフェーズが完了するまで、ユーザーストーリー実装は開始不可

- T005 open-meteo/index.htmlの基本構造を作成（DOCTYPE, meta, Chart.js CDN読み込み）
- T006 [P] open-meteo/style.cssでサイバーパンク風ベーススタイル定義（ダーク基調、緑ネオン #39ff14、マゼンタ #f6bd9d）
- T007 [P] open-meteo/script.jsでAPISelect.js初期化（canvas要素 -id="temperatureChart" をカード風コンテナ内に配置）
- T008 open-meteo/script.jsでChart.js初期化（要素取得、グラフインスタンス生成、animation: false）
- T009 [P] open-meteo/script.jsでprocessWeatherData(data)関数を実装（APIレスポンスから過去/未来データ分離、日付をMM/dd形式に変換）
- T010 [P] open-meteo/script.jsでエラーハンドリング基盤（showError関数、ユーザーへのエラーメッセージ表示）

Checkpoint: 基盤完成 - ユーザーストーリー実装を並列開始可能

Phase 3: User Story 1 - 都市選択と気温表示 (Priority: P1) MVP

Goal: ユーザーが世界の主要都市を選択し、過去・未来の気温データをグラフで確認できる

Independent Test: ドロップダウンから都市を選択し、グラフが表示されることを確認。API通信とグラフ描画の基本機能を検証できる。

User Story 1 実装

- T011 [P] [US1] open-meteo/index.htmlにドロップダウン（id="citySelect"）と10都市のoptionタグを追加
- T012 [P] [US1] open-meteo/index.htmlにcanvas要素（id="temperatureChart"）をカード風コンテナ内に配置
- T013 [US1] open-meteo/script.jsでfetchWeatherData(city)関数を実装（Open-Meteo APIへのGETリクエスト、latitude、longitude/timezone/hourly/past_days/futureDaysパラメータ）
- T014 [US1] open-meteo/script.jsでbuildApiUrl(city, pastDays, futureDays)関数を実装（URLSearchParams使用、パラメータ検証）
- T015 [US1] open-meteo/script.jsでprocessWeatherData(data)関数を実装（APIレスポンスから過去/未来データ分離、日付をMM/dd形式に変換）
- T016 [US1] open-meteo/script.jsでupdateChart(pastData, futureData, cityName)関数を実装（Chart.jsデータセット構築、過去/緑ネオン実線、未来/マゼンタ破綻、凡例文字を同色に設定）
- T017 [US1] open-meteo/script.jsでcitySelect.addEventListener('change')でAPI再リクエスト実装（都市選択時にfetchWeatherData呼び出し）
- T018 [US1] open-meteo/script.jsでローディングインジケーター実装（API通信中はスピナー表示、opacity: 0.3でグラフ半透明化）
- T019 [US1] open-meteo/script.jsでAPIエラー時のメッセージ表示（Network Error、タイムアウト、500エラー等）
- T020 [US1] open-meteo/style.cssでグラフコンテナのスタイル定義（max-width: 1600px, max-height: 900px, 淡い青グラデーション背景）
- T021 [US1] open-meteo/style.cssでlegend.labels.generateLabels()でcolor dataset.borderColor設定、v4仕様準拠
- T022 [US1] open-meteo/script.jsでChart.js例カスタマイズ（legend.labels.generateLabels()でcolor dataset.borderColor設定、v4仕様準拠）
- T023 [US1] open-meteo/script.jsでconsole.log出力追加（APIレスポンス、グラフ描画時間をperformance.measure()で計測）

Checkpoint: ユーザーストーリー1完成 - 都市選択から気温グラフ表示までの基本機能が独立して動作

Phase 4: User Story 2 - 表示期間の調整 (Priority: P2)

Goal: ユーザーが過去・未来の表示期間ボタンで調整し、より長期間または短期間のデータを確認できる

Independent Test: 期間調整ボタンをクリックし、グラフの表示範囲が変更されることを確認。API再リクエストとグラフ再描画を検証。

User Story 2 実装

- T024 [P] [US2] open-meteo/index.htmlに過去期間調整ボタン3つ追加（-1日、-7日、-14日、data-past属性でpast_days値を保持）
- T025 [P] [US2] open-meteo/index.htmlに未来期間調整ボタン3つ追加（+1日、+7日、+14日、data-future属性でfutureDays値を保持）
- T026 [US2] open-meteo/script.jsで期間調整ボタンのイベントナレッジナーチャー実装（button.dataset.past/futureを取得、currentPastDays/currentFutureDaysを更新）
- T027 [US2] open-meteo/script.jsでCITIES定義（API再リクエスト実装）
- T028 [US2] open-meteo/script.jsでAPIエラー時の状態の視覚化（エラーハンドリング）
- T029 [US2] open-meteo/style.cssで期間調整ボタンのスタイル定義（border-radius: 50%）
- T030 [US2] open-meteo/script.jsで期間調整ボタンのアクション実装（pastDays/futureDaysが1, 7, 14以外の場合はconsole.errorでエラー出力）

Checkpoint: ユーザーストーリー1と2が両方独立して動作、期間調整機能が完全に機能

Phase 5: User Story 3 - 温度単位の切り替え (Priority: P3)

Goal: ユーザーが氏名（℃）と華氏（°F）を切り替えて気温を確認できる

Independent Test: トグルスイッチをクリックし、グラフの温度表示と軸ラベルが変換されることを確認。API再リクエストとグラフ再描画を検証。

User Story 3 実装

- T031 [P] [US3] open-meteo/index.htmlに単位切り替えトグルスイッチ追加（label+inputタグ、type="checkbox"、id="unitToggle"）
- T032 [US3] open-meteo/script.jsでcelsiusToFahrenheit(celsius)関数を実装（公式: $T(F) = T(C) \times 9/5 + 32$ 、小数点以下1桁）
- T033 [US3] open-meteo/script.jsでfahrenheitToCelsius(fahrenheit)関数を実装（公式: $T(C) = (T(F) - 32) \times 5/9$ 、小数点以下1桁）
- T034 [US3] open-meteo/script.jsでupdateChartUnit()関数を実装（currentUnit単位替り、グラフデータを変換、Y軸ラベル更新）
- T035 [US3] open-meteo/script.jsでunitToggle.addEventListener('change')を実装（チェック状態でcurrentUnit更新、updateChartUnit呼び出し）
- T036 [US3] open-meteo/script.jsで単位切り替え時のAPI再リクエスト実装（クライアント側計算のみ、既存データを変換）
- T037 [US3] open-meteo/style.cssでトグルスイッチのスタイル定義（サイバーパンク風デザイン、ON時マゼンタ、OFF時緑ネオン）
- T038 [US3] open-meteo/script.jsでChart.js例カスタマイズ（options.scales.title.text = '気温(℃)' or '気温(°F)'）

Checkpoint: 全ユーザーストーリー（US1, US2, US3）が独立して機能、完全な気温予測アプリとして動作

Phase 6: Polish & Cross-Cutting Concerns

目的: 複数ユーザーストーリーに影響する改善

- T039 [P] README.mdの更新（技術スタッフ、セットアップ手順、スクリーンショット追加）
- T040 open-meteo/script.jsのコード整理（関数を单一責任原則に従って50行以内にリファクタリング、複雑度15以下）
- T041 open-meteo/script.jsのAPI再リクエスト実装（関数を单一責任原則に従って50行以内にリファクタリング、複雑度15以下）
- T042 [P] open-meteo/style.cssのコード整理（BEM命名規則適用、変数定義でカラーマネジメント）
- T043 パフォーマンス検証（Chrome DevToolsでAPI 200ms以下、グラフ描画100ms以下、メモリ100MB以下を確認）
- T044 ブラウザ互換性テスト（Chrome, Firefox, Safari, Edgeで動作確認）
- T045 quickstart.mdの最終確認（GitHub Pagesデプロイ手順が正確か検証）
- T046 T001-T010の各ストーリー完了時にGitHub Pagesデプロイ手順が正確か確認
- T047 セキュリティチェック（XSS脆弱性、HTTPS通信、機密データ非保存の確認）

Dependencies & Execution Order

フェーズ依存関係

- Setup (Phase 1): 依存なし - 即ち从属開始可能
- Foundational (Phase 2): Setup完了後 - すべてのユーザーストーリーをブロック
- User Story 1 (Phase 3): Foundational完了後 - すべてのユーザーストーリーを統合
- User Story 2 (Phase 4): Foundational完了後 - US1と統合するが独立テスト可能
- User Story 3 (Phase 5): Foundational完了後 - US1/US2と統合するが独立テスト可能
- Polish (Phase 6): すべてのユーザーストーリー完了後

ユーザーストーリー依存関係



- User Story 1 (P1): Foundational完了後に開始可能 - 他ストーリーへの依存なし、MVP
- User Story 2 (P2): Foundational完了後に開始可能 - US1と統合するが独立テスト可能
- User Story 3 (P3): Foundational完了後に開始可能 - US1/US2と統合するが独立テスト可能

ストーリー内のタスク依存関係

- HTML要素追加 → JavaScript実装（DOM要素が必要なため）
- API関数実装 → イベントリスナー実装（関数を呼び出すため）
- データ取得 → グラフ描画（データが必要なため）
- 基本実装 → スタイル適用（要素が存在する必要があるため）

並列実行機会

- Phase 1: T002, T003は並列可（異なるファイル）
- Phase 2: T006, T007, T008は並列可（異なるHTML要素）
- Phase 4: T024&T025は並列可（異なるHTML要素）
- Phase 5: T032とT033は並列可（独立した関数）
- Phase 6: T039, T042, T045は並列可（異なるファイル）

Parallel Example: User Story 1

```
# User Story 1 の並列実行例:  
Task T011: "open-meteo/index.htmlにドロップダウン追加"  
Task T012: "buildApiUrl()関数実装"  
(同時実行 - 同じ関数にparam, @returns, @throws記述)  
Task T013: "open-meteo/index.htmlにcanvas要素追加"  
(同時実行 - fetchWeatherData()関数呼び出し)  
Task T014: "open-meteo/script.jsでAPI再リクエスト実装"  
(同時実行 - fetchWeatherData()関数呼び出し)  
Task T015: "open-meteo/script.jsでprocessWeatherData(data)関数実装"  
(APIレスポンスから過去/未来データ分離)  
Task T016: "open-meteo/script.jsでupdateChart(pastData, futureData, cityName)関数実装"  
(Chart.jsデータセット構築、過去/緑ネオン実線、未来/マゼンタ破綻)
```

Implementation Strategy

MVP First (User Story 1のみ)

- Phase 1: Setup完了
- Phase 2: Foundational完了
- Phase 3: User Story 1完了
- ストップ & バリデーション: User Story 1を独立してテスト
- デプロイ/モード準備完了

段階的デリバリー

- 最小限の機能で価値提供
都市選択 → グラフ表示のみ

- 全ユーザーストーリー統合
期間調整・単位切り替え含む

- 全機能完成

- Phase 6

- Github Pages公開

1. Setup + Foundational → 基盤完成
2. User Story 1 → 独立してテスト → デプロイ/モード (MVP)
3. User Story 2 → 独立してテスト → デプロイ/モード
4. User Story 3 → 独立してテスト → デプロイ/モード
5. 各ストーリーが独立完成後、統合テスト

Task Summary

タスク統計

- 総タスク数: 47タスク
- User Story 1タスク数: 13タスク (T011-T023)
- User Story 2タスク数: 7タスク (T024-T030)
- User Story 3タスク数: 8タスク (T031-T038)
- Setup+Foundational: 10タスク (T001-T010)
- Polish: 9タスク (T039-T047)

推奨MVP範囲

通常はUser Story 1 (P1) のみでMVPとして十分:

- 都市選択機能（10都市）
- Open-Meteo API連携
- Chart.jsグラフ描画（過去7日+未来7日）
- サイバーパンク風デザイン

これだけで「世界の都市の気温を視覚化するアプリ」として価値提供可能。

Notes

- [P]タスク: 異なるファイル、依存関係なし
- [Story]タスク: 異なるファイルで実装のユーザーストーリーに紐付け、トレーサビリティ確保
- DOD非採用: 本仕様でAPI連携開発が明示的に要求されないため、テストスケジュール省略
- コットン粒度: 各タスクまたは論理グループごとに独立して実装（APIレスポン