1.設計

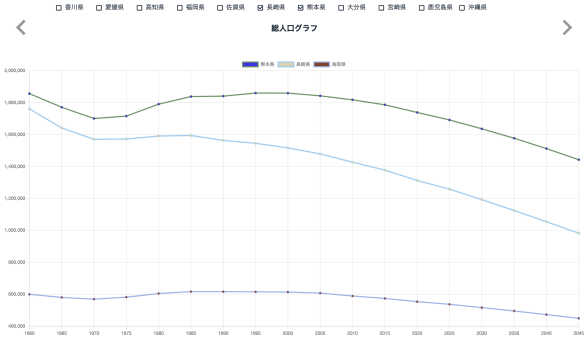
1-1.利用技術

本アプリケーションは主に以下の技術を利用している.

- Next.js (14.0.1)
- chart.js (4.4.0)
- jest (29.7.0)
- playwright (1.40.0)
- sass (1.69.5)

1-2.画面

本アプリケーションは一つの画面のみであり、以下の通りである.



1-3.ディレクトリ構成

ディレクトリ構成は以下の通りである.
UI コンポーネントのみを扱う `ui-domain` と UX コンポーネントのみを扱う `ux-domain` を分けることで、コンポーネントの再利用性を高め、責務を分離、ソースコードの可読性を高めることを意識した.

```
.
├── __test__
│   ├── api
│   │   ├── populationRoute.test.ts
│   │   └── prefecturesRoute.test.ts
│   ├── const
│   │   ├── population.ts
│   │   └── prefectures.ts
│   └── e2e
│       └── index.spec.ts
├── app
│   ├── api
│   │   ├── fetcher
│   │   ├── population
│   │   ├── prefectures
│   │   └── utils
│   ├── hooks
│   ├── styles
│   ├── ui-domain
│   │   ├── ArrowButton
│   │   ├── Chart
│   │   ├── CheckBox
│   │   ├── Header
│   │   ├── PopulationCards
│   │   │   └── Card
│   │   └── Spinner
│   └── ux-domain
│       └── SelectPopulationChart
│           ├── index.tsx
│           └── logic-ui
│               ├── PopulationChart
│               └── SelectPrefecture
├── const
├── fetcher
├── types
└── utils
```

`__test__`

- フロントエンドのテストコードが入るディレクトリ

. `api`

  - API Route のテストコードが入るディレクトリ

. `const`

  - テストコードで扱う定数が入るディレクトリ

. `e2e`

  - E2E テストコードが入るディレクトリ

`app`

- コンポーネント、フック、スタイルなど、フロントエンドに関する処理が入るディレクトリ

- `api`
  - API routes に関する処理が入るディレクトリ
  - 今回は RESAS API との通信に関する処理が入り、複数回叩くなどの処理がある
  - `fetcher`
    - RESAS API を叩く際に使う fetcher を定義するディレクトリ
  - `population`
    - `api/population` のエンドポイントに関する処理が入るディレクトリ
  - `prefectures`
    - `api/prefectures` のエンドポイントに関する処理が入るディレクトリ
  - `utils`
    - 型確認と型作成に関する処理が入るディレクトリ
- `hooks`
  - フックスを定義するディレクトリ
  - エンドポイントに関するフックスが入っている
- `styles`
  - グローバルに扱うスタイルが入るディレクトリ
- `ui-domain`
  - アプリケーションで扱う UI コンポーネントが入るディレクトリ
- `ux-domain`
  - アプリケーションで発生するユーザーの操作に関する処理が入るディレクトリ

`const`

- アプリケーションで扱う定数が入るディレクトリ

`fetcher`

- API Routes を叩く際に使う fetcher を定義するディレクトリ

`types`

- アプリケーションで扱う型が入るディレクトリ

`utils`

- アプリケーションで扱うユーティリティ関数が入るディレクトリ

2.仕様

2-2.概要

本アプリケーションは RESAS API（https://opendata.resas-portal.go.jp/）が提供するの都道府県データと人口予測データを元に時系列による折れ線グラフを表示する.
都道府県のチェックボックスを選択することでそれに応じた折れ線グラフが表示される.また、グラフの種類を選択することが可能であり、選択したグラフの種類に応じた折れ線グラフが表示される.

2-2.機能

本アプリケーションは都道府県のチェックボックスに入力された値を元に以下の 4 種類のグラフを表示する.

- 総人口グラフ
- 年少人口グラフ
- 生産年齢人口グラフ
- 老年人口グラフ

2-3.データ加工

利用した RESAS API のデータのエンドポイントは以下の通りである.

- 都道府県データ：`GET  api/v1/prefectures`
- 人口予測データ：`GET  api/v1/population/composition/perYear`

RESAS API が提供する人口予測データは複数の都道府県を元に検索を行うことができない.そのため人口予測データのエンドポイントからを表示したい都道府県の数だけ取得する必要がある.しかし、取得の為の処理を UI 描写に記述すると余計なロジックが追加されることにより、ソースコードの可読性が落ちてしまう.よって Next.js の機能の一つである API Routes を中間 API として利用し、クライアントと RESAS API の間に存在するデータの加工処理を一任することで対処した.

2-3.テスト

本アプリケーションは E2E テストと API Routes で処理した結果が正しいかのテストを行っている. E2E テストではレンダリング時からデータのフェッチ、グラフの描写までの処理をテストしている.具体的にはフェッチ結果とタイトルが画面上に描写されているかをテストしている.また、API Routes でのテストは RESAS API から取得したデータを元に加工した結果が正しいかをテストしている.

3.工夫と反省

3.1.工夫

本アプリケーションでは可読性の向上と保守性の向上を意識して開発を行った.そのためディレクトリ構成を意識し、コンポーネントの再利用性を高めることを意識した.また、コンポーネントの再利用性を高めるために UI コンポーネントと UX コンポーネントを分けることで責務を分離した.さらに、UX コンポーネントは UI コンポーネントを組み合わせて作成している.これにより UX コンポーネントは UI コンポーネントのみを扱うことができ、UX コンポーネントの再利用性を高めることができた.

3.2 反省

今回の開発では可読性の高いソースコードを意識していた為、一度完成するまでにおおよそ 2 週間かかってしまった.また、開発中にテストコードを書くことを意識していなかった為、テストコードを書くことに時間がかかってしまった.今後は開発を行う際にテストコードを書くことを意識し、開発を行っていきたい.