一覧表示システム
Webアプリケーション仕様書

25G1034 恩田隼士

2025年12月20日

目 次

第1章　仕様書テンプレート

1.1　システム概要

本システムは、Node.js (Express) を用いたサーバーサイドレンダリング (SSR) Web アプリケーションである。3 つの異なるデータセット（都道府県、元素記号、88 星座）に対し、RESTful な設計に基づいた CRUD（作成、読み取り、更新、削除）機能を提供する。

1.1.1　特徴と制約

- **データ永続化なし**: データベースを使用せず、**サーバーサイドの変数（メモリ上の配列）**にデータを保持する。サーバーを再起動すると、データは初期状態にリセットされる。
- **テンプレートエンジン**: EJS を使用し、HTML を動的に生成する。
- **拡張性**: ルーティングとビューを追加することで、容易に新しいテーマ（例：商品管理、書籍リストなど）を追加可能である。

1.2　アーキテクチャ構成

システムは MVC（Model-View-Controller）モデルに近い構成をとるが、簡易化のため Model 部分は変数操作として実装する。

1.2.1　ディレクトリ構成案

開発者が直感的にファイルを配置できるよう、以下の構成を推奨する。

```
root/
|-- app.js                # エントリーポイント
|-- routes/               # ルーティング定義 (Controller相当)
|   |-- prefectures.js    # 都道府県用ルーター
|   |-- elements.js       # 元素記号用ルーター
|   `-- constellations.js # 88星座用ルーター
|-- views/                # EJSテンプレート (View)
|   |-- index.ejs         # トップページ
|   |-- prefectures/      # 都道府県用ビューフォルダ
|   |   |-- index.ejs     # 一覧表示
|   |   |-- show.ejs      # 詳細表示
|   |   |-- new.ejs       # 新規作成フォーム
|   |   `-- edit.ejs      # 編集フォーム
|   |-- elements/         # (構成は同上)
|   `-- constellations/   # (構成は同上)
|-- public/               # 静的ファイル (css, images)
`-- package.json          # 依存関係定義
```

Listing 1.1: ディレクトリ構成

1.3 データモデル仕様 (Model)

データは各ルーターファイル (例: `routes/prefectures.js`) 内のローカル変数として定義する。

1.3.1 変数定義のルール

- **型**: オブジェクトの配列 (`Array<Object>`)
- **初期データ**: サーバー起動時に配列に格納されるデフォルトデータ。
- **ID 管理**: 新規追加時は、既存の最大 ID + 1 を付与するか、単純なインクリメントロジックを実装する。

```
// データストア (変数)
let prefectures = [
    { id: 1, name: '北海道', region: '北海道', capital: '札幌市' },
    { id: 2, name: '青森県', region: '東北', capital: '青森市' },
    // ...初期データ
];
```

```
7
8 // 次のIDを管理する変数
9 let nextId = 48;
```

Listing 1.2: routes/prefectures.js の実装例

1.4 APIエンドポイントと処理フロー

1.4.1 実装すべきルート処理詳細

各テーマ共通で実装するURLパターンと処理内容は以下の通りである。

URL パターン	メソッド	処理概要と注意点
/	GET	**一覧表示** 配列全体を views/index.ejs に渡す。
/new	GET	**新規作成画面** 空のフォームを表示する。
/	POST	**データ作成** req.body から値を取得し、ID を付与して配列に push する。その後一覧へリダイレクト。
/:id	GET	**詳細表示** req.params.id と一致する要素を配列から find で検索し表示。見つからない場合は 404。
/:id/edit	GET	**編集画面** 対象データを検索し、フォームの初期値 (value) として渡して表示。
/:id	POST	**データ更新** 対象データを検索し、req.body の内容でプロパティを上書きする。
/:id/delete	POST	**データ削除** req.params.id と一致する要素を配列から除外 (filter 等) する。

1.5 拡張・変更ガイド

1.5.1 ケース A：データの項目を増やしたい

例：都道府県に「名産品 (product)」を追加する場合。

1. **データの変更**: routes/prefectures.js の初期データ配列に product プロパティを追加する。
2. **新規作成処理の変更**: 同ファイルの POST / ルート内で、req.body.product を受け取るように修正する。
3. **更新処理の変更**: 同ファイルの POST /:id ルート内で、更新ロジックに product を追加する。
4. **ビューの変更**:
   - index.ejs: 表に見出しと列を追加。
   - show.ejs: 表示項目を追加。
   - new.ejs / edit.ejs: <input name="product"> タグを追加。

1.5.2 ケース B：新しいテーマ（例：戦国武将）を追加したい

1. **ルーター作成**: routes/warriors.js を作成し、CRUD ルートとデータ配列（warriors）を記述する。
2. **ビュー作成**: views/warriors/ フォルダを作成し、4 つの ejs ファイルを作成する。
3. **アプリ登録**: app.js に以下を追記する。

```
1 const warriorsRouter = require('./routes/warriors');
2 app.use('/warriors', warriorsRouter);
```

1.6 技術スタック要件

- **Runtime**: Node.js (v14 以上推奨)
- **Framework**: Express
- **Template Engine**: EJS
- **Body Parser**: Express 標準 (`express.urlencoded`) ※ POST データ受け取りに必須

第2章　開発者向け仕様書：都道府県一覧表示システム

2.1　概要

本ドキュメントは，Node.js およびテンプレートエンジン EJS を用いた「都道府県管理 Web アプリケーション」の設計仕様書である．本システムは，サーバーサイドで都道府県データを管理し，EJS を用いて動的に HTML を生成・返却する．データベースの利用は行わず，サーバープロセスのメモリ上（変数）でデータを保持・操作することを前提とする．

2.2　システム要件

2.2.1　技術スタック

- **ランタイム**: Node.js
- **Web フレームワーク**: Express
- **テンプレートエンジン**: EJS
- **データ保存先**: サーバーサイドメモリ（配列変数）

2.2.2　データ構造 (Schema)

サーバー内の配列変数で管理するオブジェクト構造は以下の通り．

表 2.1: 都道府県データスキーマ

プロパティ名	データ型	説明
id	Number	一意な識別子．登録時に自動採番（最大値+1等）．
name	String	都道府県名（例: 大阪府）．
capital	String	県庁所在地（例: 大阪市）．
region	String	地方区分（例: 近畿）．

2.3 ディレクトリ・ページ構成

Express の標準的な構成に基づき，ビュー（EJS）ファイルを配置する．

```
project-root/
    |- app.js           (メインロジック・データ変数保持)
    |- public/          (CSS，画像などの静的ファイル)
    |- views/           (EJS テンプレート)
        |- index.ejs    (一覧表示画面)
        |- show.ejs     (詳細表示画面)
        |- new.ejs      (新規登録フォーム)
        |- edit.ejs     (編集フォーム)
```

2.4 HTTP メソッドとルーティング

RESTful な設計に基づき, 各 URL と HTTP メソッド, および対応する処理を定義する. ※ブラウザの HTML フォーム標準仕様では PUT/DELETE が使用できないため，method-override ライブラリを使用するか，POST メソッドで代用する想定とする（本仕様では論理的なメソッドを記載）．

表 2.2: ルーティング一覧

機能	メソッド	パス (URL)	対応ビュー/処理
一覧表示	GET	`/prefectures`	`views/index.ejs`
新規作成フォーム	GET	`/prefectures/new`	`views/new.ejs`
詳細表示	GET	`/prefectures/:id`	`views/show.ejs`
編集フォーム	GET	`/prefectures/:id/edit`	`views/edit.ejs`
新規データ作成	POST	`/prefectures`	(処理後一覧へリダイレクト)
データ更新	PUT	`/prefectures/:id`	(処理後詳細へリダイレクト)
データ削除	DELETE	`/prefectures/:id`	(処理後一覧へリダイレクト)

2.5 ページ遷移図

画面間の遷移フローを以下に示す.

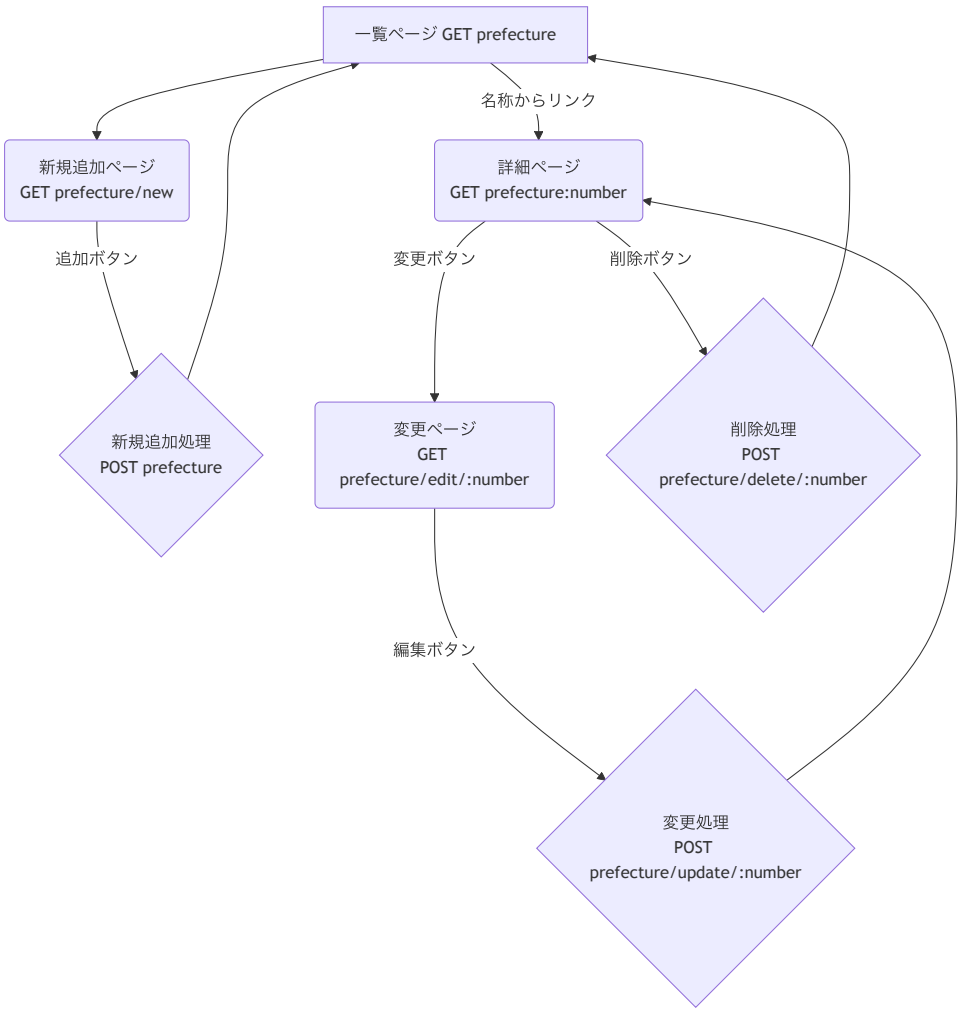
一覧ページ GET prefecture
名称からリンク
新規追加ページ
GET prefecture/new
詳細ページ
GET prefecture:number
追加ボタン
変更ボタン
削除ボタン
新規追加処理
POST prefecture
変更ページ
GET
prefecture/edit/:number
削除処理
POST
prefecture/delete/:number
編集ボタン
変更処理
POST
prefecture/update/:number

図 2.1: 画面遷移フロー

2.6　各機能・リソース詳細

2.6.1　1. 一覧機能 (Index)

- **URL**: `GET /prefectures`
- **処理**: サーバー変数の全データを EJS に渡し，`forEach` 文を用いてテーブル形式でレンダリングする．
- **要素**:「新規登録ボタン」，各行ごとの「詳細リンク」．

2.6.2　2. 新規作成機能 (New / Create)

- **フォーム (GET /prefectures/new)**:
  - 都道府県名，県庁所在地，地方の入力フィールドを表示．
  - 送信先は `POST /prefectures`．
- **作成処理 (POST /prefectures)**:
  - リクエストボディから値を取得．
  - 新しい ID を採番し，サーバー変数（配列）に push する．
  - 処理完了後，一覧画面 (`/prefectures`) へリダイレクトする．

2.6.3　3. 詳細表示機能 (Show)

- **URL**: `GET /prefectures/:id`
- **処理**: URL パラメータの ID に基づき配列を検索（`find`）し，対象データを表示する．
- **要素**:「編集ボタン」,「削除ボタン」(form タグによる POST/DELETE 送信),「一覧に戻るボタン」．

2.6.4 4. 編集・更新機能 (Edit / Update)

- **フォーム (GET /prefectures/:id/edit)**:
  - 対象データを検索し，value 属性に現在の値を埋め込んで表示する．
  - 送信先は PUT /prefectures/:id.
- **更新処理 (PUT /prefectures/:id)**:
  - ID に基づき配列内の該当インデックスを特定.
  - リクエストボディの値でプロパティを上書きする．
  - 詳細画面 (/prefectures/:id) へリダイレクトする．

2.6.5 5. 削除機能 (Destroy)

- **URL**: DELETE /prefectures/:id
- **処理**: ID に基づき配列から要素を取り除く（filter 等を使用）.
- **挙動**: 削除完了後，一覧画面 (/prefectures) へリダイレクトする．