Web アプリケーション開発仕様書

(In-Memory Data System)

開発チーム

2025 年 12 月 5 日

目次

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　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。

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。

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　ディレクトリ構成

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。

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
];

// 次のIDを管理する変数
let nextId = 48;
```

Listing 2　routes/prefectures.js の実装例

4　API エンドポイントと処理フロー

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。

5 拡張・変更ガイド

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。

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。

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

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