2025 年度　web プログラミング　レポート課題

25G1048　栗原　蓮

1　はじめに

本レポートは，作成した 3 つの Web アプリケーションに関する仕様を，「利用者」「管理者」「開発者」の 3 つの観点より使用方法を記述したものである．3 つのシステムとは「文明一覧システム」「紀元前の時代一覧システム」「世界の観光地一覧システム」である．全て GitHub にて全体の構造を公開している．

2　利用者向け仕様書

2.1　概要

本システムは，歴史や地理に関するデータをブラウザ上で手軽に管理・閲覧できる Web アプリケーションである． システムの使い方は 3 つのシステムを通してカラーリング以外ほとんど同じである．そのため 1 つのシステムを代表として説明を行う．文明一覧システムの操作を中心に説明する．

2.2　目的

膨大な歴史データや観光地の情報をテキストだけでなく画像と合わせて整理し，視覚的に分かりやすく管理することを目的としている．一覧表示による全体像の把握と，詳細画面による深い知識の確認を両立させることで，情報整理の効率化を目指した．また過去の文明を知ることによりより深くそれぞれの国の歴史や文化について知ってもらおうとする意図がある．

2.3　主な機能

一覧表示　登録データをリスト形式で確認する．
詳細表示　各データの詳細情報や大きな画像を閲覧できる．
新規登録　専用フォームから新しいデータを追加できる．
編集・削除　既存データの修正や不要なデータの消去ができる．

2.4　操作方法

2.4.1　アプリの起動とメニュー選択

ブラウザのアドレスバーに [http://localhost:8080] と入力すると，Web プログラミング課題メニュー（トップ画面）が表示される．ここから利用したいシステム（例：CIVILIZATION）のパネルをクリックして移動

する．

図 1　Google chrome **のアドレスバー**

図 2　**トップ画面の様子　それぞれの言葉をタップするとそのページに遷移する**

図 3 にてカラーが 3 つに分かれていてそれぞれのカラーの中に言葉があるが，それぞれのカラーをタップするとそれぞれの言葉に対応するシステムページに遷移する．今回は例として CIVILIZATION をクリックする．

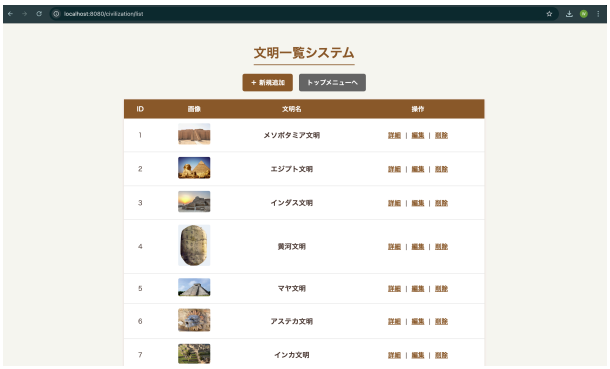

図 3　CIVILIZATION **をクリック後の画面　文明一覧システムの画面**

各システムのトップ画面では，現在登録されているデータが表形式で一覧表示される．詳細を知りたい場合は，データ名をクリックするか，操作列の詳細ボタンを押すことで詳細画面へ移動できる．トップメニューへというボタンをクリックすると図 3 の画面に戻ることができる．また新規追加したい場合は新規追加をクリックすることで図 4 の画面に遷移する．

図 4　**新規追加をクリックした後の画面　新規登録画面**

それぞれの情報を入力したのちに新規登録することができる．また任意で画像を追加することができ，画像を追加する際は任意のディレクトリに画像データを入れてそこまでのパスを入力することで追加することができる．一覧に戻るをクリックすると図 3 の画面に戻ることができる．なお編集するボタンをクリックすると図 4 と同じような画面に飛び，既存の登録してあるデータを自分の言葉で書き換えたり，画像を変更することができる．

3　管理者向け仕様書

3.1　概要

本システムは Node.js 上で動作するサーバーアプリケーションである．単一のサーバープロセスで 3 つの異なるアプリケーション（文明、時代、観光地）を一括して提供・管理する仕様となっている．なおこれから記載する内容は全て github 上より [webpro_06] をダウンロードしていることが前提となる．

3.2　動作環境と起動方法

3.2.1　実行方法

必要環境　Node.js がインストールされていること．

起動手順１　ターミナル（コマンドプロンプト）を開き，プロジェクトフォルダへ移動する．
起動手順２　コマンド npm install を実行する．
起動手順２　コマンド node app5.js を実行する．
起動手順３　コンソールに起動メッセージ（Example app listening on port 8080!）が表示されれば，ブラウザからのアクセスが可能となる．

3.3 データの運用と流れ

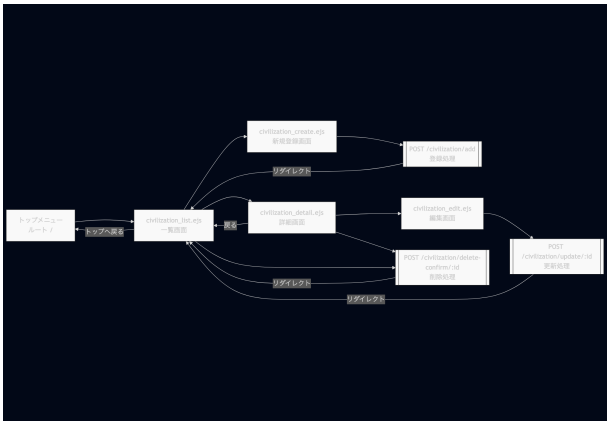

図 5　**データの遷移の流れ**

図 5 は，本システムにおける画面遷移とデータ処理の流れを示したものである．すべての操作はトップメニューから始まり，各システムの一覧画面へと遷移する．各システム一覧画面からは，トップメニューへ戻ることも可能である．一覧画面から任意のデータを選択すると詳細画面へ遷移し，詳しい情報を閲覧できる．詳細画面からは，再び一覧画面へ戻る導線が確保されている．データの変更を伴う操作（新規登録、編集、削除）は，入力フォームまたは確認画面を経て，サーバーへ POST リクエストを送信することで実行される．新規登録は新規登録画面で入力後，登録処理を行い一覧へ戻る．編集では編集画面で内容を修正後，更新処理を行い一覧へ戻る．削除では削除処理を実行後，直ちに一覧へ戻る．いずれの処理においても，サーバー側での処理完了後は自動的に一覧画面へリダイレクトされ，ユーザーは即座に最新のデータ状態を確認できる設計となっている．また全てのシステムにおいて使用感は変わらず，同じようにシステムの管理を行えるようになっている．

次にデータ保存の仕組みについて話す．簡易的な運用のためにデータベースサーバーを使用せず，プログラム内のメモリ（変数）上でデータを管理している．そのためサーバーを停止・再起動すると，稼働中に追加・編集されたデータはリセットされ，初期データ（ソースコードに記述されたサンプルデータ）に戻る仕様である．そのため永続的なデータ保存が必要な場合は，別途データベースの実装が必要となる．または初期のソースコードに追加する必要性がある．

3.4 エラー処理

接続できない場合はサーバーが起動しているかターミナルを確認する．ポート 8080 番が他のアプリで使用されていないか注意する．またはコマンドが間違っていないか，URL が間違っていないかを確かめる．追加した画像が表示されない場合，画像ファイルの配置場所が/public/images/以下の正しいフォルダにあるか，ファイル名が正しいかを確認する．画面に「404 Not Found」という文字だけが表示される場合は，URL が誤っているか，存在しないページにアクセスしようとしているため URL の確認をする．

4　開発者向け仕様書

4.1　アプリ 1

4.1.1　概要・機能

四大文明や中南米の文明など，古代文明の名称・成立場所・特徴を管理する Web システムである．テーマカラーには歴史・大地を想起させるブラウン (5d4037) を採用している．本システムでは一覧・作成・読み取り・更新・削除機能を実装している．

4.1.2　目的

歴史学習において，文字情報だけでなく場所や関連画像をセットで管理することで，知識の定着を助けることを目的とする．また，基本的な作成・読み取り・更新・削除機能を実装し，自身でも学習を勧められやすい環境を構築した．

4.1.3　データ構造

本システムではデータベースサーバーを使用せず，プログラム（app5.js）内の配列オブジェクトとしてデータを管理している．各アプリケーションで使用するデータモデルの定義は以下の通りである．

表 1　app1: 文明データ (civilizations)

プロパティ名	データ型	説明
id	Number	データを一意に識別するための管理 ID（自動採番）
name	String	文明の名称（例：メソポタミア文明）
river	String	成立した場所や河川名（例：チグリス・ユーフラテス川）
feature	String	文明の主な特徴や遺産（例：楔形文字、ハンムラビ法典）
image	String	画像ファイルのパス（例：`/public/images/...`）

4.1.4　遷移

本システムのページ遷移の流れ，また対応するメソッドとリソース名を示す．また使用する各ファイルの名前とその大まかな役割についてもまとめて示す．

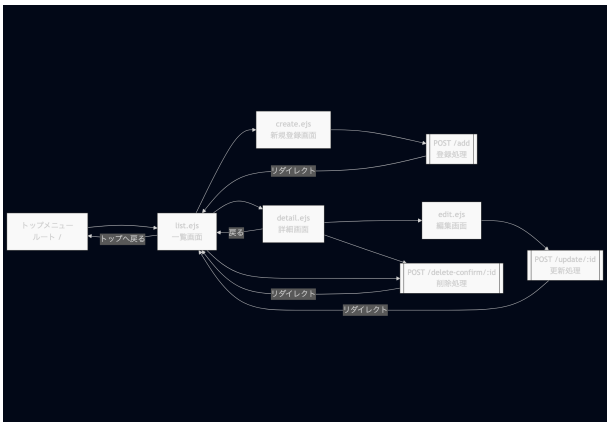

図 6　**データの遷移の流れ**

表 2　表 2 ファイル名と対応する役割一覧表

項目名	**詳細説明**
1. app5.js	Web サーバーのエントリーポイント．静的ファイルの配信に加え，各アプリ（文明・時代・観光地）へのリクエスト（GET, POST）を処理し，データの配列管理を行うメインプログラム．
2. views/civilization_list.ejs	文明一覧画面を表示する HTML テンプレート．文明データの配列を受け取り，画像付きのリスト形式で表示する．
3. views/civilization_detail.ejs	文明詳細画面を表示するテンプレート．選択された ID の詳細情報や大きな画像を表示し，編集・削除ボタンを配置している．
4. views/civilization_create.ejs	新規登録用の入力フォーム画面．必須項目の設定や POST 送信先（/add）が記述されている．
5. views/civilization_edit.ejs	編集用の入力フォーム画面．既存のデータ（名前や特徴など）を初期値として表示する処理が含まれる．
6. public/images	各アプリで使用する画像ファイルを格納するディレクトリ．サブフォルダ（civilization, era, spot）で分類されている．
7. package.json	プロジェクト全体の設定ファイル．サーバーを起動するための依存パッケージ（Express, EJS 等）の情報が記述されている．

表 3　表 3 HTTP メソッドとリソース名一覧

メソッド・リソース	機能・処理内容
GET /	トップメニュー（Web プログラミング課題メニュー）を表示する.
GET /civilization/list	文明一覧画面を表示する.
GET /civilization/detail/:id	指定された ID の文明詳細画面を表示する.
GET /civilization/create	新規登録フォームを表示する.
POST /civilization/add	フォームから送信されたデータを配列に追加し，一覧画面へリダイレクトする（登録処理）.
GET /civilization/edit/:id	指定された ID の編集フォームを表示する.
POST /civilization/update/:id	フォームから送信されたデータで既存情報を上書きし，一覧画面へリダイレクトする（更新処理）.
POST /civilization/delete-confirm/:id	指定された ID のデータを配列から削除し、一覧画面へリダイレクトする（削除処理）.

本システムにおける画面遷移とデータ処理の流れを示したものである．すべての操作はトップメニューから始まり，各システムの一覧画面へと遷移する．各システム一覧画面からは，トップメニューへ戻ることも可能である．一覧画面から任意のデータを選択すると詳細画面へ遷移し，詳しい情報を閲覧できる．詳細画面からは，再び一覧画面へ戻る導線が確保されている．データの変更を伴う操作（新規登録、編集、削除）は，入力フォームまたは確認画面を経て，サーバーへ POST リクエストを送信することで実行される．新規登録は新規登録画面で入力後，登録処理を行い一覧へ戻る．編集では編集画面で内容を修正後，更新処理を行い一覧へ戻る．削除では削除処理を実行後，直ちに一覧へ戻る．いずれの処理においても，サーバー側での処理完了後は自動的に一覧画面へリダイレクトされ，ユーザーは即座に最新のデータ状態を確認できる設計となっている．また全てのシステムにおいて使用感は変わらず，同じようにシステムの管理を行えるようになっている．

4.2　アプリ２

4.2.1　概要・機能

縄文時代や弥生時代，あるいは中国の春秋戦国時代など，紀元前の主要な時代区分を管理する Web システムである．テーマカラーには古代の自然や大地を想起させるグリーン (2e7d32) を採用している．本システムでは一覧・作成・読み取り・更新・削除機能を実装している．

4.2.2　目的

歴史の学習において，いつからいつまで続いたのか，どこの国の時代区分なのかを整理して可視化することを目的とする．異なる地域の時代を同じフォーマットで一覧表示することで，同時代の世界の動きを比較・把握しやすい環境を構築した．

4.2.3　データ構造

アプリ 1 と同様に，プログラム内の配列オブジェクト（eras）としてデータを管理している．データモデルの定義は以下の通りである．

表 4　app2: 時代データ (eras)

プロパティ名	データ型	説明
id	Number	データを一意に識別するための管理 ID（自動採番）
name	String	時代の名称（例：縄文時代）
period	String	時代の期間（例：BC14000 頃 - BC300 頃）
region	String	その時代区分が用いられる地域（例：日本，中国）
feature	String	時代の特徴や主な出来事
image	String	画像ファイルのパス（例：`/public/images/...`）

4.2.4　遷移

本システムのページ遷移の流れ，また対応するメソッドとリソース名を示す．ページ遷移図についてはアプリ 1 と共通の構造（図 6 参照）であるため省略するが，使用する各ファイルの名前とその大まかな役割について以下にまとめて示す．

表 5　表 4 ファイル名と対応する役割一覧表（アプリ 2）

項目名	詳細説明
1. views/era_list.ejs	時代一覧画面を表示する HTML テンプレート．時代データの配列を受け取り，期間や地域とともにリスト形式で表示する．
2. views/era_detail.ejs	時代詳細画面を表示するテンプレート．選択された ID の詳細情報を表示する．ヘッダー色は緑色に統一されている．
3. views/era_create.ejs	新規登録用の入力フォーム画面．期間の入力例（プレースホルダー）などが設定されている．
4. views/era_edit.ejs	編集用の入力フォーム画面．既存のデータを初期値として表示する．

表 6　表 5 HTTP メソッドとリソース名一覧（アプリ 2）

メソッド・リソース	機能・処理内容
GET /era/list	時代一覧画面を表示する．
GET /era/detail/:id	指定された ID の時代詳細画面を表示する．
GET /era/create	新規登録フォームを表示する．
POST /era/add	フォームから送信されたデータを配列に追加し，一覧画面へリダイレクトする（登録処理）．
GET /era/edit/:id	指定された ID の編集フォームを表示する．
POST /era/update/:id	フォームから送信されたデータで既存情報を上書きし，一覧画面へリダイレクトする（更新処理）．
POST /era/delete-confirm/:id	指定された ID のデータを配列から削除し，一覧画面へリダイレクトする（削除処理）．

4.3 アプリ３

4.3.1 概要・機能

世界遺産や有名な観光地のデータを管理する Web システムである．国名や遺産種別（文化遺産・自然遺産など）を登録できる．テーマカラーには空や海，旅行の楽しさを想起させるブルー (0277bd) を採用している．本システムでは一覧・作成・読み取り・更新・削除機能を実装している．

4.3.2 目的

地理や旅行に関する情報を，テキストだけでなく魅力的な画像とともにデータベース化することを目的とする．文字情報よりも視覚的な情報を重視し，ユーザーが見て楽しめる旅行ガイドブックのような UI を目指して構築した．

4.3.3 データ構造

プログラム内の配列オブジェクト（spots）としてデータを管理している．データモデルの定義は以下の通りである．

表 7　app3: 観光地データ (spots)

プロパティ名	データ型	説明
id	Number	データを一意に識別するための管理 ID（自動採番）
name	String	観光地・遺跡の名称（例：モン・サン・ミシェル）
country	String	所在する国名（例：フランス）
type	String	遺産の種別やカテゴリ（例：文化遺産，自然遺産）
description	String	観光地に関する説明文
image	String	画像ファイルのパス（例：`/public/images/...`）

4.3.4 遷移

本システムのページ遷移の流れ，また対応するメソッドとリソース名を示す．使用する各ファイルの名前とその大まかな役割について以下にまとめて示す．

表 8　表 6 ファイル名と対応する役割一覧表（アプリ３）

項目名	詳細説明
1. views/spot_list.ejs	観光地一覧画面を表示する HTML テンプレート．画像の表示領域を広く確保し，視覚的に楽しめるレイアウトになっている．
2. views/spot_detail.ejs	観光地詳細画面を表示するテンプレート．選択された ID の詳細情報を表示する．ヘッダー色は青色に統一されている．
3. views/spot_create.ejs	新規登録用の入力フォーム画面．国名や種別の入力欄が設定されている．
4. views/spot_edit.ejs	編集用の入力フォーム画面．既存のデータを初期値として表示する．

表 9　表 7 HTTP メソッドとリソース名一覧（アプリ 3）

メソッド・リソース	機能・処理内容
GET /spot/list	観光地一覧画面を表示する.
GET /spot/detail/:id	指定された ID の観光地詳細画面を表示する.
GET /spot/create	新規登録フォームを表示する.
POST /spot/add	フォームから送信されたデータを配列に追加し，一覧画面へリダイレクトする（登録処理).
GET /spot/edit/:id	指定された ID の編集フォームを表示する.
POST /spot/update/:id	フォームから送信されたデータで既存情報を上書きし，一覧画面へリダイレクトする（更新処理).
POST /spot/delete-confirm/:id	指定された ID のデータを配列から削除し，一覧画面へリダイレクトする（削除処理).

4.4　プログラムと CSS

本システムそれぞれの開発にあたり，授業で扱った基礎技術に加え，実用的な Web アプリケーション開発において利用される技術・手法を導入した．トップメニューの画面分割において，CSS3 の Flexible Box Layout Module を採用した．コンテナ要素に display: flex を指定し，子要素（各アプリのパネル）に flex: 1 を設定した．flex: 1 は flex-grow: 1・flex-shrink: 1・flex-basis: 0% のショートハンドである．これにより，ブラウザのウィンドウ幅がどのように変化しても，3 つのパネルが余白を均等に埋め尽くす計算が自動的に行われる．パネル内の文字配置には justify-content: center（主軸方向の中央揃え）と align-items: center（交差軸方向の中央揃え）を併用し，親要素の高さに関わらず常にコンテンツが上下左右の中央に来るよう制御している．従来の float プロパティや position: absolute による配置よりも，構造に依存しないレイアウト手法である．一覧画面における画像表示では，アスペクト比の保持とレイアウトの統一を両立させるために object-fit プロパティを使用した．img タグに固定の width と height を指定すると，元画像のアスペクト比と異なる場合に画像が歪んでしまう問題がある．object-fit: cover を適用することで，ブラウザは画像の短辺を表示領域に合わせて拡大・縮小し，長辺側を切り取るというレンダリング処理を行う．これにより，ユーザーがどのようなサイズの画像をアップロードしても，UI 上のサムネイル枠（例: 80px × 50px）に美しく収まるシステムを実現した．ユーザーの操作性を高めるため，擬似クラス（Pseudo-classes） を使用して動的なスタイルを適用した．tr:hover をテーブルの行（tr）に対して設定した．マウスカーソルが乗った瞬間に background-color を変更することで，現在選択しようとしている行を視覚的に強調する．一覧表示のテーブルにおいて，border-collapse: collapse を採用し，シンプルにした．また，th（ヘッダーセル）と td（データセル）に適切な padding を設定し，情報の密度を調整している．

データの操作において，従来の for ループではなく高階関数を用いた宣言的な記述を行った．Array.prototype.find() では詳細画面などで ID 検索を行う際，配列の中から条件に一致する最初の要素を返すというロジックを直感的に記述できる．Array.prototype.filter() では削除処理において，指定 ID 以外を抽出するというフィルタリング処理を 1 行で実装でき，元の配列を破壊せず，新しい配列を生成する操作を行っている．Array.prototype.map() では特定のプロパティ（ID 一覧など）を抽出する際に使用した．新規データ登録時の ID 自動採番ロジックにおいて，const newId = Math.max(...data.map(d =¿ d.id)) + 1; の

記述を用いた．Math.max() 関数は配列を直接引数に取れない仕様である．そこで，スプレッド構文 (...) を使用して配列を展開し，個別の引数として関数に渡している．これにより，apply メソッドなどを使用する古い手法よりも高速で処理できるコードを実現した．画像ファイルへのアクセス制御において，Express の組み込みミドルウェア関数を使用した．app.use(”/public”, express.static(...)) を定義することで，Web サーバー内の特定のディレクトリをクライアントに公開している．これにより，個別の画像リクエストに対してルーティング処理を書く必要がなくなり，非同期 I/O による高速なファイル配信を実現している．また文字列操作において＋演算子ではなく ‘を用いたテンプレートリテラルを使用した．これにより改行を含む HTML コードをそのまま記述することを可能にした．