基本的なデータ表示の Web アプリ開発 仕様書

25G1016 入江晟太

2025 年 12 月 24 日

GitHub リポジトリ URL

本仕様書で説明する 3 つの Web アプリケーションのソースコード一式は，次の GitHub リポジトリで管理している．

`https://github.com/jota92/wpro2025`

リポジトリ直下には，以下の 3 つのディレクトリを用意し，それぞれ独立した Web アプリケーションとして実装した．

- `tasks_app`：授業課題・テスト管理アプリ
- `subs_app`：サブスク管理アプリ
- `todos_app`：ToDo リストアプリ

1　利用者向け仕様

この節では，実際に Web アプリケーションを利用するユーザの視点から，3 つのアプリケーションの目的，画面構成，主な操作を説明する．

1.1　授業課題・テスト管理アプリ（tasks_app）

目的

大学の授業ごとに課題やテストが複数存在するため，「どの授業で，いつまでに，どの課題やテストがあるか」を一覧で把握できるようにすることを目的とする．本アプリケーションでは，授業名，課題名，種別，締切日，状態をまとめて管理し，締切が近い課題を見落とさないようにする．

画面構成と画面遷移

主な画面は次の 4 つである．

- 一覧画面：登録済みの課題・テストを表形式で一覧表示する．
- 詳細画面：1 件の課題・テストの内容を表示する．
- 新規登録画面：新しい課題・テストを追加するフォームを表示する．
- 編集画面：既存の課題・テストの内容を変更するフォームを表示する．

画面遷移の概要を図 1 に示す．

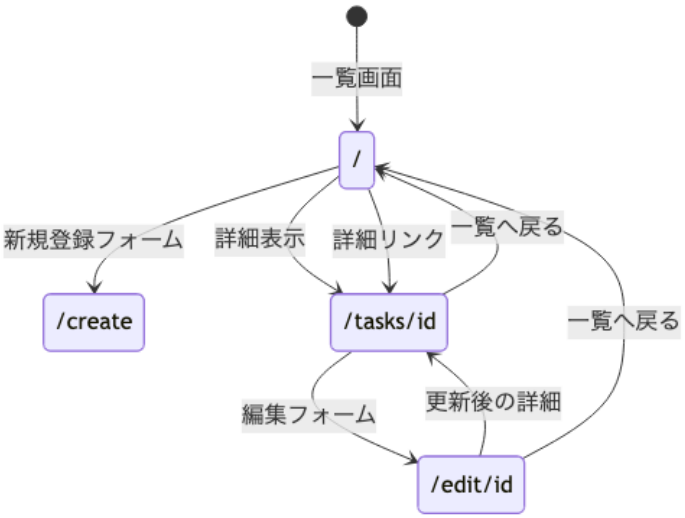

図 1　授業課題・テスト管理アプリの画面遷移図

主な操作

利用者は次のような操作を行う．

- 一覧画面で，登録済みの課題・テストを確認する．
- 一覧画面から新規登録画面に移動し，新しい課題・テストを登録する．
- 詳細画面から編集画面に移動し，状態や締切日などを更新する．
- 詳細画面の削除ボタンで，不要になった課題・テストを削除する．

1.2　サブスク管理アプリ（subs_app）

目的

動画配信サービスや音楽配信サービスなどのサブスクリプションが増えると，どのサービスにいくら支払っているのか把握しづらくなる．本アプリケーションは，契約中のサブスクを一覧管理し，サービス名，カテゴリ，料金，課金サイクルなどを整理して確認できるようにすることを目的とする．

画面構成と画面遷移

主な画面は次の 4 つである．

- 一覧画面：登録済みサブスクを表形式で表示する．
- 詳細画面：1 件のサブスクについて，料金や課金サイクルなどを表示する．
- 新規登録画面：新しいサブスクを登録するフォームを表示する．
- 編集画面：既存のサブスク情報を変更するフォームを表示する．

画面遷移の概要を図 2 に示す．

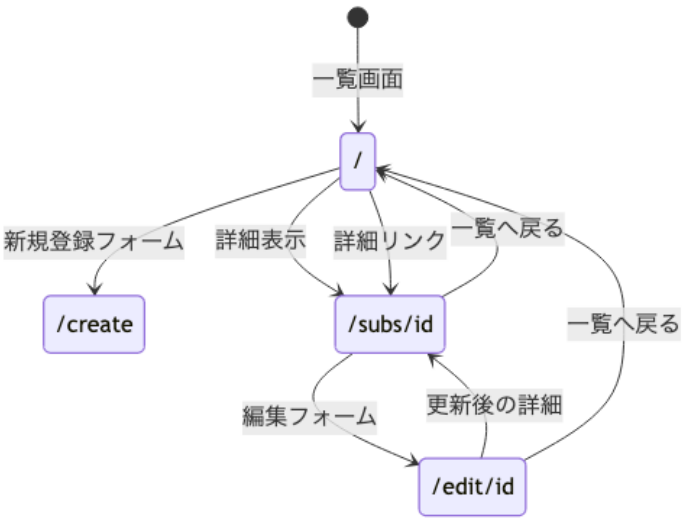

図 2　サブスク管理アプリの画面遷移図

主な操作

- 一覧画面で，契約しているサブスク一覧と月額料金を確認する．
- 新規登録画面で，新たに契約したサブスクを登録する．
- 詳細画面から編集画面に移動し，料金変更やメモの追記を行う．
- 解約したサブスクを削除し，一覧を整理する．

1.3　ToDo リストアプリ（todos_app）

目的

日々の作業ややるべきことを簡単に整理できる ToDo リストを提供する．タスクのタイトル，優先度，完了状況などを一覧で管理し，重要なタスクを見落とさないようにすることを目的とする．

画面構成と画面遷移

主な画面は次の 4 つである．

- 一覧画面：登録済みの ToDo を表形式で表示する．
- 詳細画面：1 件の ToDo について，優先度や完了状態などを表示する．
- 新規登録画面：新しい ToDo を登録するフォームを表示する．
- 編集画面：既存の ToDo の内容を変更するフォームを表示する．

画面遷移の概要を図 3 に示す．

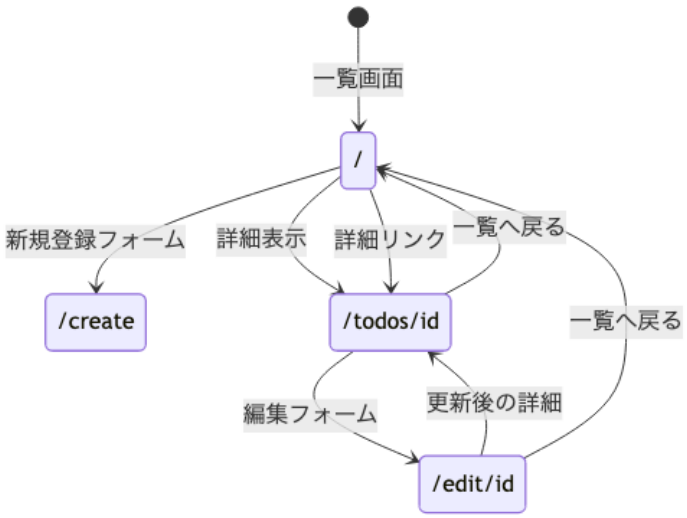

図 3　ToDo リストアプリの画面遷移図

主な操作

- 一覧画面で，登録済み ToDo の一覧を確認する．
- 新規登録画面で，新しい ToDo を登録する．
- 詳細画面から編集画面に移動し，優先度や完了状態を更新する．
- 完了済みまたは不要な ToDo を削除する．

2　管理者向け仕様

この節では，データ構造や入力ルール，サーバの起動・停止方法など，運用・管理の観点から 3 つのアプリケーションを説明する．いずれのアプリケーションも，データはサーバプログラム内の配列変数として保持し，サーバを再起動すると初期データに戻る．

2.1 授業課題・テスト管理アプリのデータ仕様

課題・テストは，表 1 に示す項目を持つオブジェクトとして `tasks` 配列で管理する．

表 1　課題・テストのデータ項目

項目名	型	概要
id	数値	課題・テストの ID
title	文字列	タイトル
course	文字列	科目名
type	文字列	種別（レポートなど）
dueDate	文字列	締切日（YYYY-MM-DD）
status	文字列	状態（未提出など）
memo	文字列	メモ

入力・運用ルールを以下に示す．

- `title`, `course`, `type`, `dueDate`, `status` は必須項目とする．
- `status` は「未提出」「勉強中」「提出済」のいずれかを選択する．
- `dueDate` は HTML の日付入力欄から `YYYY-MM-DD` 形式で入力する．
- サーバ再起動時には `tasks` 配列が初期状態に戻るため，長期保存したい場合は別途バックアップを行う．

2.2 サブスク管理アプリのデータ仕様

サブスクリプション情報は，表 2 に示す項目を持つオブジェクトとして `subscriptions` 配列で管理する．

表 2　サブスクのデータ項目

項目名	型	概要
id	数値	サブスクの ID
name	文字列	サービス名
category	文字列	カテゴリ（動画配信など）
fee	数値	月額料金（円）
cycle	文字列	課金サイクル
memo	文字列	備考

運用ルールは次のとおりである．

- name, category, fee, cycle は必須項目とする．
- fee は 0 以上の整数とし，フォームでは数値入力欄を用いる．
- 解約したサブスクは削除処理で一覧から除去する．

2.3 ToDo リストアプリのデータ仕様

ToDo データは，表 3 に示す項目を持つオブジェクトとして todos 配列で管理する．

表 3　ToDo のデータ項目

項目名	型	概要
id	数値	ToDo の ID
title	文字列	タイトル
priority	文字列	優先度（高・中・低）
done	真偽値	完了フラグ
memo	文字列	メモ

運用ルールは次のとおりである．

- title は必須項目とし，空のまま登録しない．
- priority は「高」「中」「低」から選択させる．
- done はチェックボックスの ON/OFF を真偽値として保持する．

2.4 サーバの起動・停止手順

管理者がサーバを運用する際の共通手順を以下に示す．

前提条件

- Node.js がインストールされていること．
- 上記 GitHub リポジトリをクローン済みであること．

授業課題・テスト管理アプリ（tasks_app）の起動・停止

1. ターミナルで wpro2025/tasks_app ディレクトリに移動する．
2. 初回のみ npm install を実行して依存パッケージをインストールする．
3. npm start または node app.js を実行し，ポート 8080 でサーバを起動する．
4. ブラウザから http://localhost:8080/ にアクセスし，一覧画面を表示する．
5. 停止するときは，ターミナルで Ctrl+C を押してサーバを終了する．

サブスク管理アプリ（subs_app）の起動・停止

1. ターミナルで wpro2025/subs_app ディレクトリに移動する．
2. 初回のみ npm install を実行して依存パッケージをインストールする．
3. npm start または node app.js を実行し，ポート 8080 でサーバを起動する．
4. ブラウザから http://localhost:8080/ にアクセスし，一覧画面を表示する．
5. 停止するときは，ターミナルで Ctrl+C を押してサーバを終了する．

ToDo リストアプリ（todos_app）の起動・停止

1. ターミナルで wpro2025/todos_app ディレクトリに移動する．
2. 初回のみ npm install を実行して依存パッケージをインストールする．
3. npm start または node app.js を実行し，ポート 8080 でサーバを起動する．
4. ブラウザから http://localhost:8080/ にアクセスし，一覧画面を表示する．
5. 停止するときは，ターミナルで Ctrl+C を押してサーバを終了する．

複数のアプリケーションを同時に利用する場合は，ポート番号が競合しないように，一度サーバを停止してから別のアプリケーションを起動することとする．

3 開発者向け仕様

この節では，実装者向けにプログラム構造と処理内容を説明する．3 つのアプリケーションはいずれも共通の構成を持ち，app.js をエントリポイントとして views ディレクトリ内の EJS テンプレートを描画する．

3.1 共通の実装構成

各アプリケーションの package.json では，メインファイルを app.js とし，依存モジュールとして express と ejs を利用している．app.js 冒頭の構成は共通であり，下記のように設定している．

- const express = require('express'); で Express を読み込む．
- app.set('view engine', 'ejs'); によりテンプレートエンジンを EJS に設定する．
- app.use("/public", express.static(__dirname + "/public")); で静的ファイルを配信する．
- app.use(express.urlencoded({ extended: true })); で HTML フォームの POST データを解釈できるようにする．
- 各 app.js の先頭に "use strict"; を記述し，厳格モードで実行する．

テンプレートは views ディレクトリ内に list.ejs，detail.ejs，form.ejs を配置し，CSS ファイルは public/style.css にまとめた．

3.2 授業課題・テスト管理アプリ（tasks_app）のプログラム構造

データ定義と検索関数

app.js では，初期データを持つ tasks 配列を定義している．各要素は表 1 で示した項目を持つオブジェクトであり，サンプルとして Web プログラミングのレポートや線形代数の小テストなどを登録している．

個々の課題を取得するため，次のような関数を定義する．

```
function findTaskById(id) { return tasks.find(t => t.id === id); }
```

ルーティングと処理内容

ルーティングは以下のように実装している．

- GET /：list.ejs を用いて課題一覧を表示する．
- GET /tasks/:id：findTaskById で該当課題を取得し，detail.ejs に渡して表示する．見つからない場合は 404 を返す．
- GET /create：form.ejs に空の課題データと /create を渡し，新規登録フォームを表示する．
- POST /create：フォーム送信値から新しい課題オブジェクトを作成し，tasks 配列の末尾に追加する．ID は最後の要素の ID に 1 を足して採番する．
- GET /edit/:id：編集対象の課題を取得し，form.ejs に渡して編集フォームを表示する．
- POST /edit/:id：フォーム送信値で既存の課題の各項目を上書きする．
- POST /delete/:id：tasks 配列から該当 ID の要素を除外し，一覧画面へリダイレクトする．

3.3 サブスク管理アプリ（subs_app）のプログラム構造

データ定義と検索関数

subs_app/app.js では，サブスク情報を持つ subscriptions 配列を定義している．各要素は表 2 で示した項目を持ち，Netflix や Spotify などのサンプルデータを登録している．

個々のサブスクを取得するため，次の関数を定義する．

```
function findSubById(id) { return subscriptions.find(s => s.id === id);
}
```

ルーティングと処理内容

主なルート定義は次のとおりである．

- `GET /`：`list.ejs` を用いてサブスク一覧を表示する．
- `GET /subs/:id`：`findSubById` で該当サブスクを取得し，`detail.ejs` に渡して表示する．
- `GET /create`：新規登録フォームを `form.ejs` で表示する．
- `POST /create`：フォーム送信値から新しいサブスクを生成し，`subscriptions` 配列に追加する．料金は `Number` で数値化する．
- `GET /edit/:id`：編集対象を取得し，`form.ejs` に渡して表示する．
- `POST /edit/:id`：フォーム送信値で既存のサブスクを更新する．
- `POST /delete/:id`：`subscriptions` 配列から該当サブスクを削除する．

3.4 ToDo リストアプリ（todos_app）のプログラム構造

データ定義と検索関数

`todos_app/app.js` では，ToDo 情報を持つ `todos` 配列を定義している．各要素は表 3 の項目を持ち，レポートの要件整理や部屋の片付けなどのサンプルデータを登録している．

個々の ToDo を取得するため，次の関数を定義する．

```
function findTodoById(id) { return todos.find(t => t.id === id); }
```

ルーティングと処理内容

主なルート定義は次のとおりである．

- `GET /`：`list.ejs` を用いて ToDo 一覧を表示する．
- `GET /todos/:id`：`findTodoById` で該当 ToDo を取得し，`detail.ejs` に渡して表示する．
- `GET /create`：新規登録フォームを `form.ejs` で表示する．
- `POST /create`：フォーム送信値から新しい ToDo を生成し，`todos` 配列に追加する．優先度の未指定時は「中」，完了フラグはチェックボックスの ON/OFF から真偽値を設定する．
- `GET /edit/:id`：編集対象を取得し，`form.ejs` に渡して表示する．
- `POST /edit/:id`：フォーム送信値で既存の ToDo を更新する．
- `POST /delete/:id`：`todos` 配列から該当 ToDo を削除する．

3.5 エラー処理とサーバ起動

各アプリケーションの末尾では，存在しないパスに対して 404 を返すためのミドルウェアを定義している．

- `app.use((req, res, next) => { res.status(404).send("ページが見つかりません"); });`

さらに，`app.listen(8080, () => { ... });` でポート 8080 番を待ち受けるサーバを起動し，コンソールに起動メッセージを出力するようにしている．