Web ページでのデータ一覧管理システムの仕様書

24G1142 米原亙

2025 年 12 月 28 日

1　ソースコード

report に関するものがまとめられた github のディポジトリの url を以下に添付する．
https://github.com/wyonehara/report.git

2　利用者向け仕様書（野球選手一覧システム）

2.1　概要

Web ページ上で野球選手の一覧を閲覧，編集できるシステムを作成した．以下に，ユーザーが Web 上で行う操作方法を説明する．

2.2　操作方法

2.2.1　ホームページへのアクセス

まず野球選手一覧へのアクセス方法は，図 1 のように http://localhost:8080/players をお使いの web ブラウザの検索ボックスに入力し，ホームページにアクセスします．ホームページにアクセスすると図 2 のように，野球選手一覧の画面に選手の名前，背番号が表示される．

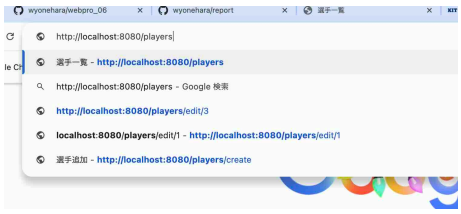

図 1　ホームページへのアクセス

2.2.2　選手詳細

トップページの選手の名前のリンクをクリックすると，図 3 のように選手の詳細ページに飛ぶことができる．一覧に戻りたいときは，"一覧へ戻る"をクリックすることで戻ることができる．

図 2　トップページ

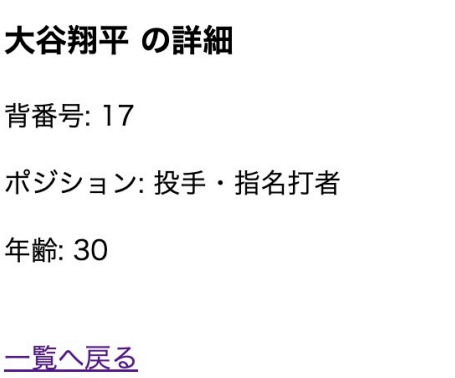

図 3　選手詳細ページ

2.2.3　編集

選手の詳細を編集したい場合は，トップページの選手の名前の右にある”編集”をクリックすると図 4 のページに飛ぶことができる．そして，編集したい項目のチャットボックスを変更し，”更新”をクリックすることで変更が可能である．大谷翔平の年齢を 31 に変更後，詳細ページに飛ぶと図 5 のように大谷の詳細が変更している．編集をやめたい場合は”戻る”をクリックすることで，トップページに戻る．

図 4　編集ページ

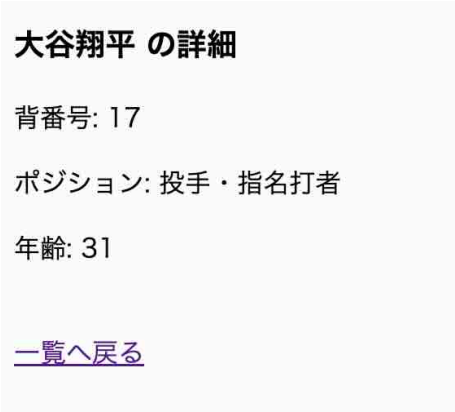

図 5　編集後の詳細ページ

2.2.4　選手の削除

選手を削除したい場合は選手の名前の右にある”削除”をクリックすると図 6 のように削除してよいか確認がされるので”OK”を押すことで図 7 のように削除される．削除をやめたい場合には”キャンセル”をクリックする．

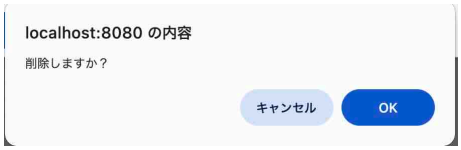

図 6　削除確認

図 7　削除後のトップページ

2.2.5　選手の追加

選手を追加したい場合は，トップページの”選手追加”をクリックすることで図 8 のように選手追加ページに飛ぶことができる．すべてのチャットボックスを埋め，登録ボタンをクリックすることで図 9 のように選手が追加される．選手の追加をやめたい場合は戻るボタンをクリックするとトップページに戻ることができる．

図 8　追加ページ

図 9　追加後のトップページ

3　管理者向け仕様書

3.1　サーバの立ち上げ

ターミナルやコマンドプロンプトで rep.js を配置したディレクトリで図 10 のように”node rep.js”を実行し，サーバーが起動する．ポート番号 8080 で listen され，起動に成功した場合メッセージが表示される．サー

バを終わらせたい場合は，ターミナルで control+c を押して終了させる．

```
yonewata@yoneharamac report % node rep.js
Example app listening on port 8080!
```

図 10　サーバ立ち上げ時のターミナル

4　開発者向け仕様書

野球選手一覧, 世界都市システム, やることリストの 3 つを nodejs を用いて作成する．開発言語は Javascript で，各システムはローカルネットワーク上で動作し，ブラウザからアクセスすることで利用可能である．データベースは使用せず，すべてのデータはサーバプログラム内の変数として管理している．

4.1　野球選手一覧システム

野球選手一覧を作成する．本システムは野球選手の情報を管理し，ユーザーがブラウザからアクセスすることで，野球選手の一覧表示，詳細表示，データの追加，編集，削除を行うことができる．本仕様書では，開発者向けに本システムの構成について説明する．また，ページ遷移図は図 11 のようになっている．

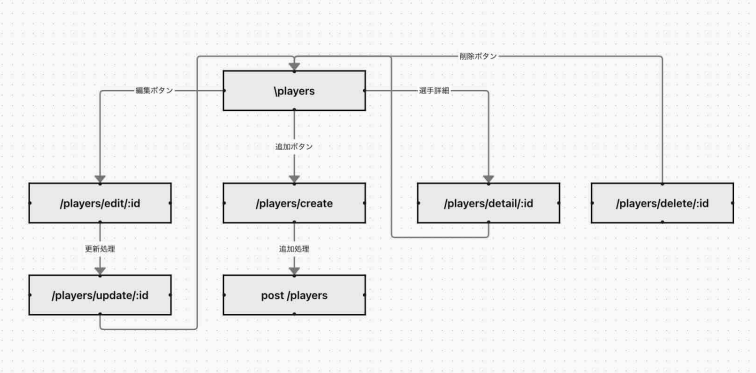

図 11　野球選手一覧のページ遷移図

4.1.1　データ管理

野球選手の情報は，{id: Number（一意な ID），name: String（選手名），number: Number（背番号），position: String（ ポジション），age: Number（年齢）} の形式のオブジェクトとして管理する．これらのオブジェクトを配列 players に格納し，簡易的なデータベースとして扱っている．ID は各選手を一意に識別するための値であり，選手追加時に自動で採番されるよう実装している. 本システムでは，ID と配列の index を混同しないようにして実装を行った．URL では必ず ID を使用し，サーバ側では find や findIndex を用いて対象データを取得することで，データの不整合が発生しないようにしている．

4.1.2　一覧表示（GET /players）

選手一覧の表示では，GET メソッドで/players にアクセスすると，サーバ側で管理しているすべての選手データを取得し，players.ejs に render し，表形式で一覧表示している．各選手名は詳細画面へのリンクとなっており，リンクをクリックすることで/players/detail/:id にページ遷移し，該当する選手の詳細情報を閲覧することができる．また，各行には編集ボタンおよび削除ボタンが配置されており，それぞれ/players/edit/:id, /players/delete/:id に遷移する．一覧の下には新規選手追加のボタンがあり，/players/create に遷移する．

4.1.3　選手追加（GET /players/create）

新規選手の追加は，/players でボタンを押し GET メソッドで/players/create にアクセスすると，HTML ファイル players_add.html を表示する．この画面では，選手名，背番号，ポジション，年齢を入力するフォームを表示する．フォーム送信時には POST メソッドで players にデータが送信される．

4.1.4　選手追加処理（POST /players）

追加処理では，フォームから送信されたデータを受け取り，新しい選手データを作成する．既存の選手データの中から最大の ID を取得し，その値に 1 を加えたものを新しい選手の ID として設定している．これにより，途中で選手が削除された場合でも ID が重複しないようにしている．追加後は一覧画面へリダイレクトし，更新された選手一覧を表示する

4.1.5　詳細表示（GET /players/detail/:id）

詳細表示では，URL パラメータとして受け取った ID をもとに，players 配列から該当する選手データを検索する．取得したデータは player_detail.ejs に渡され，選手の名前，背番号，ポジション，年齢といった詳細情報を表示する．また，一覧へ戻るリンクを表示し，/players に戻れる仕様にしている．

4.1.6　編集（GET /players/edit/:id）

選手情報の編集では，編集対象の ID をもとに選手データを取得し，取得したデータは player_edit.ejs に渡され，既存の情報を入力欄に表示した編集画面を生成する．ユーザーは各項目を編集することができ，フォーム送信時には更新処理へ遷移する．

4.1.7　更新処理（GET /players/update/:id）

更新処理では，対象となる選手の ID を保持したまま，名前や背番号などの項目のみを更新するようにしている．これにより，選手の識別情報が変更されることなく，内容のみを編集できる構成となっている．更新後は一覧画面へリダイレクトする．

4.1.8　削除（GET /players/delete/:id）

削除処理では，指定された ID を持つ選手を配列から削除し，処理完了後に一覧画面へリダイレクトし，削除後の一覧を表示する．削除後もシステムが正しく動作するよう，一覧表示や詳細表示では常に ID を基準としてデータを取得する設計としている．

4.2　世界都市システム

世界各地の都市情報を一覧で表示する Web アプリケーションを作成する．本システムは，世界の都市の情報を管理し，ユーザーはブラウザからアクセスすることで，都市情報の一覧表示，詳細表示，データの追加，編集，削除を行うことができる．本システムも野球選手一覧システムと同様に，Node.js を用いて実装している．また，ページ遷移図は図 12 のようになっている．

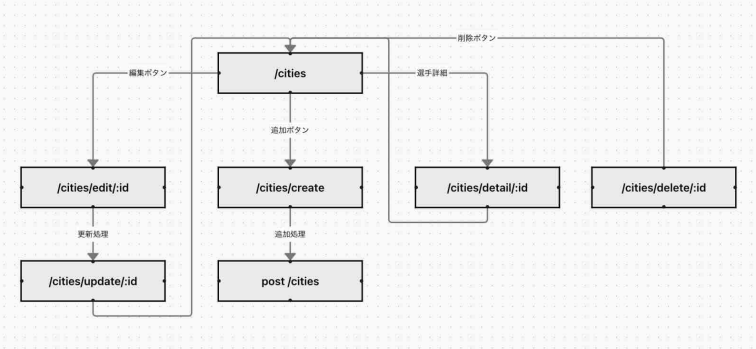

図 12　追加後のトップページ

4.2.1　データ管理

都市情報は，{id: Number（一意な ID），name: String（都市名），country: String（国名），population: Number（人口），industry: String（主要産業）} の形式を持つオブジェクトとして管理する．これらのオブジェクトを配列 `cities` に格納し，簡易的なデータベースとして扱っている．

ID は都市を一意に識別するための値であり，ユーザーが指定して追加できるように実装している．そのため，都市の追加と更新処理では，既存の都市データと ID が重複しないかをチェックし，重複する場合は処理を中断する設計にしている．一覧表示時には，ID 順に並び替えて表示することで，見やすい画面構成としている．

4.2.2　一覧表示（GET /cities）

一覧表示では，GET メソッドで/cities にアクセスすると，すべての都市データを取得する．取得したデータは ID 順にソートされた後，cities.ejs にレンダーされ，表形式で表示される．各都市名は都市詳細へのリンクとなっていて，クリックすると/cities/detail/:id に遷移する．また，各行には編集ボタンおよび削除ボタンが配置されており，それぞれ/cities/edit/:id，/cities/delete/:id に遷移する．一覧の下には都市追加のボタンがあり，/cities/create に遷移する．

4.2.3　都市追加（GET /cities/create）

都市追加画面では，GET メソッドで/cities/create にアクセスすると，都市情報入力用の HTML ファイル cities_add.html を表示する．この画面では，ID，都市名，国名，人口，主要産業を入力することができるフォームを表示する．

4.2.4　都市追加処理（POST /cities）

追加処理では，フォームから送信された都市情報を受け取り，新しい都市情報を cities 配列に追加する．このとき，入力された ID が既存の都市と重複していないかを確認し，重複がある場合はエラーメッセージを表示する．追加完了後は一覧画面へリダイレクトする．

4.2.5　詳細表示（GET /cities/detail/:id）

詳細表示では，受け取った ID をもとに都市データを検索し，該当する都市の詳細情報を表示する．取得したデータは cities_detail.ejs に渡され，都市名，国名，人口，主要産業といった詳細情報を表示する．また，一覧へ戻るリンクを表示し，/cities に戻れる仕様にしている．

4.2.6　編集（GET /cities/edit/:id）

編集画面では，指定された ID の都市データを取得し，取得したデータは cities_edit.ejs に渡され，既存の情報を入力欄に表示したフォームを生成する．ユーザーは都市情報および ID を変更することが可能である．

4.2.7　更新処理（POST /cities/update/:id）

更新処理では，編集画面から送信されたデータをもとに都市情報を更新する．ID を変更する場合は，ほかの都市と重複しないことを確認した上で更新を行う．更新後は一覧画面へリダイレクトする．

4.2.8　削除（GET /cities/delete/:id）

削除処理では，指定された ID を持つ都市データを配列から削除し，一覧画面へリダイレクトする．削除後も一覧表示は常に ID を基準として行われる．

4.3　やることリストシステム

やることリストシステムは，タスクを優先順位付きで管理することを目的とした Web アプリケーションである．ユーザーはタスクの一覧表示，詳細表示，追加，編集，削除を行うことができる．本システムも他のシステムと同様に Node.js を用いて実装している．また，ページ遷移図は図 13 のようになっている．

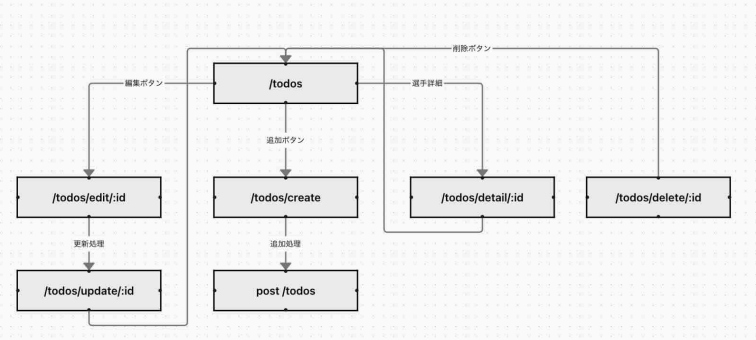

図 13　追加後のトップページ

4.3.1　データ管理

タスク情報は，id: Number（一意な ID），priority: Number（優先順位），title: String（タイトル），detail: String（詳細）の形式を持つオブジェクトとして管理する．これらのオブジェクトを配列 todos に格納している．ID はタスク追加時に自動で採番される．また，優先順位はユーザーが指定できるように実装しており，順位が重複しないように制御を行っている．

4.3.2　一覧表示（GET /todos）

一覧表示では，GET メソッドで/todos にアクセスすると，cities.ejs にレンダーされ，すべてのタスクを取得し，優先順位順に表示する．各タスクには編集および削除の操作リンクが配置されている．

4.3.3　タスク追加（GET /todos/create）

タスク追加では，GET メソッドで/todos/create にアクセスすると，やることリストを入力するの HTML ファイル todos_add.html を表示する．画面ではタイトル，詳細，優先順位を入力するフォームを表示する．ユーザーは新しいタスクを任意の優先順位に挿入することができる．

4.3.4　タスク追加処理（POST /todos）

追加処理では，新しいタスクの優先順位を受け取り，既存のタスクのうち同じまたはそれ以上の優先順位を持つものを繰り下げる．その後，新しいタスクを指定された優先順位で追加する．この処理により，優先順位が重複しない状態を維持している．

4.3.5　詳細表示（GET /todos/detail/:id）

詳細表示では，指定された ID をもとにタスクを find 検索し，todo_detail.ejs にレンダーしてタイトル，詳細，優先順位を表示する．

4.3.6　編集（GET /todos/edit/:id）

編集画面では，指定された ID の都市データを取得し，取得したデータは todo_edit.ejs に渡され，既存のタスク情報を入力欄に表示し，ユーザーが内容および優先順位を変更できるようにしている．

4.3.7　更新処理（POST /todos/update/:id）

更新処理では，変更前と変更後の優先順位を比較し，他のタスクの優先順位を必要に応じて繰り上げまたは繰り下げる．これにより，編集後も優先順位が重複しない状態を保っている．

4.3.8　削除（GET /todos/delete/:id）

削除処理では，指定された ID のタスクを削除する．削除後は，削除されたタスクより下位の優先順位を持つタスクを繰り上げることで，優先順位の連続性を維持している．