

Web プログラミング基礎  
**Castle Datastore**  
マップから城を探せるアプリ

2024 年 1 月 30 日  
k23075 多田隆人

# 1 目的

このアプリケーションは、城好きに向けた城巡りの計画をサポートするための Web アプリケーションである。城好きは旅行や日々の移動の中で城に訪れる機会があり、その際は訪問予定の地域に良い城があるかを探す。しかし日本における城は約 3 万箇所存在するため、一括でまとめられたサイトがなく付近の城を探すのが大変である。また地域名で探しても実際に訪れる位置から離れている場合があり時間が掛かる。そこで本アプリケーションはマップ上で城の位置を示すことにより、簡単に城の位置を知られるようにすることを目的とする。

# 2 機能

このアプリケーションには、トップ画面・マップ画面・編集画面がある。また各画面のマップの表示位置・ズームレベルは同期しローカルストレージに保存されるため、ページ遷移やリロードによっても保持される。

## 2.1 トップ画面

トップ画面では、最初に訪れた人に対してアプリケーションの説明や情報を提供する。表示される内容にはアプリケーションの概要・マップ画面へのリンク・登録情報・マーカの説明・編集方法がある。登録情報は API を用いて動的に取得している。この画面を図 1 に示す。

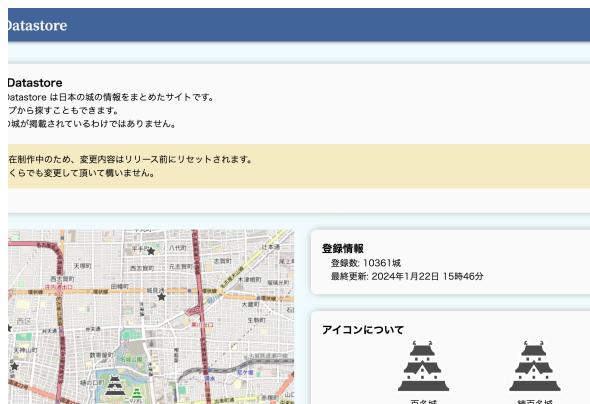


図 1: トップ画面(上部)



図 2: トップ画面(下部)

## 2.2 マップ画面

マップ画面では、マップ上に城の位置と規模を表示する。スクロールやズームにより表示範囲を変更することができる。マップ上にはマーカが表示され、クリックすることで城名が表示される。またマーカは規模によって異なるため視覚的に区別するこ

とができる。このマーカは API を使用し動的に取得することで、表示範囲内のマーカを適切な数だけ表示している。ズームにレベルによる表示の違いを図 3, 図 4 に示す。

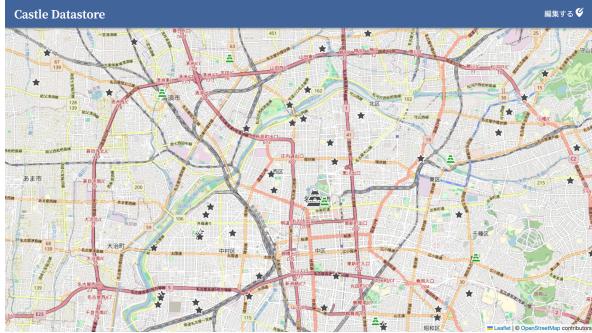


図 3: ズームレベル 15

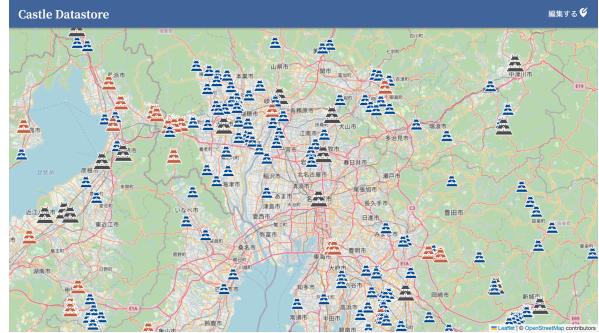


図 4: ズームレベル 10

### 2.3 編集画面

編集画面では、マーカの情報の編集・追加・削除ができる。マーカのポップアップのペンマークを押すことでそのマーカが選択される。また、マップ上でダブルクリックすることでマーカが追加される。編集中のマーカはアイコンが変わることで区別することができる。

マーカの情報はテキストボックスから編集することができる。また位置の編集はマーカをドラッグ・ドロップすることでもできる。編集内容は送信を押すことで反映される。

スマートフォンなどの画面幅が 700px 以下の端末ではマップが大きく表示されるため、編集画面を表示するためにスクロールが必要になる。しかしマップによりスクロールができない。そこで画面幅が 700px 以下の場合はマップの右下にスクロールボタンを表示することでスクロールを可能にしている。PC の画面を図 5 に、iPhoneXR においてスクロールボタンを押す前後の各画面を図 6, 図 7 に示す。



図 5: 編集画面



図 6: スクロールボタン押下前



図 7: スクロールボタン押下後

### 3まとめ

このアプリはマップから簡単に城を探せることを目的としている。そのため城の位置をマーカで示すことで見つけやすくした。また、全てのマーカを表示すると動作が重くなるため、表示範囲内のマーカのみを表示するようにした。これにより動作が軽くなり、ユーザエクスペリエンスの向上に成功した。またレスマートフォンなどの小さい画面でも利用できるようにスponシブ対応し、ユーザビリティの向上にも成功した。

また、マーカの追加・編集・削除という一連の操作を実装でき、中核となる機能を完成了。しかし、各城ごとの画面や各都道府県の城一覧画面がデータベースの構造上難しく実装できなかった。そしてマーカの追加・編集・削除の際には認証が必要なく、変更を追跡することができないといった問題がある。これらの問題に対しては次の節で述べる。

#### 3.1 改善点

今後の改善点として変更の追跡と各城・各都道府県ごとの画面の実装が挙げられる。

まず変更の追跡について述べる。現在は認証がないため、誰がいつ変更したかを追跡することができない。これはFirebase Authentication等を使用することで解決できる。認証情報を元に編集履歴を保存することで変更の追跡が可能になる。

次に各城・各都道府県ごとの画面の実装はデータベースの構造上難しく実装できなかった点について述べる。今回はリレーショナルデータベースであるCloudflare D1を使用した。一定範囲内のマーカを取得することは容易である一方、複雑な城の情報を取得することは難しく読み込み回数が増加する。この問題はFireStore等のNoSQLデータベースを使用することで複雑な情報を管理できる。このためマーカ

と城の情報をそれぞれのデータベースで管理し、NoSQL データベースからマーカ情報を取り・更新することで解決できる。

## 4 付録

この章では本アプリケーションの開発における特徴を述べる。

### 4.1 フレームワーク・ライブラリ

本アプリケーションでは複数画面を持ち、ヘッダー・フッター・マップなどの共通部分がある。そこで Next.js を使用することで共通部分をコンポーネント化し、開発を効率化とデザインの統一をした。また API 部分では Hono と Drizzle ORM を使用することで Cloudflare D1 との連携を容易にした。

### 4.2 API 通信部分

今回、API との通信に Fetch API を使用した。Fetch API を使用することでシンプルに実装することができる。また、頻繁に使用するコードは関数化することで再利用性を高めた。GET リクエストを行う関数を図 8 に示す。

```
1  export const commonGetFetch = async <T>(
2    path: string
3  ): Promise<T | undefined> => {
4    const url = new URL(path, BASE_URL).href;
5
6    return fetch(url)
7      .then((res) => res.json())
8      .catch((err) => {
9        console.error(err);
10       return undefined;
11     });
12   };

```

図 8: GET リクエストを行う関数

### 4.3 レスポンシブ対応

本アプリケーションのトップ画面ではブロックに分けて表示している。これらのブロックは画面幅が 700px 以下の場合は縦並びになるようにしている。このレイアウトの変更は 12 分割グリッドにより実装を容易にした。親要素、子要素の CSS をそれぞれ図 9、図 10 に示す。

```
1 .main {  
2   display: grid;  
3   grid-template-columns: repeat(12, 1fr);  
4   gap: 30px;  
5   padding-top: 40px;  
6 }
```

図 9: 親要素の CSS

```
1 .map {  
2   grid-column: span 6;  
3   grid-row: span 2;  
4  
5   @media (max-width: 700px) {  
6     grid-column: span 12;  
7   }  
8 }
```

図 10: 子要素の CSS

#### 4.4 デプロイ

今回このアプリケーションのフロントエンド部分は Cloudflare Pages に、バックエンド部分は Cloudflare Workers にデプロイした。全て Cloudflare を使用することで管理と連携を容易にした。また Cloudflare D1 では sqlite を使っているため、Docker 等の立ち上げが不要でローカル環境での開発が容易になった。

デプロイ先は下記 URL からアクセスできる。<https://castles.satooru.dev/>

#### 4.5 ローカルストレージ

ユーザはマップを移動・ズームすることで表示範囲を変更する。再度開いた際には前回の表示範囲を保持していると便利であるため、ローカルストレージに保存することで実装した。このコードの内、ローカルストレージからの読み込みを図 11、マップ設定の読み込みを図 12 に示す。

```
1 export function getLocalStorage<T>(key: LocalStorageKey, init: T): T {  
2   if (typeof window === "undefined") return init;  
3  
4   const valueStr = localStorage.getItem(key);  
5   if (valueStr === null) return init;  
6  
7   const value: T = JSON.parse(valueStr);  
8   return value;  
9 }
```

図 11: ローカルストレージからの取得

```
1 const mapSettings = atom<MapSettings>({  
2   key: recoilKeyHashSet.mapSettings,  
3   default: getLocalStorage<MapSettings>("mapSettings", init),  
4 });
```

図 12: マップ設定の読み込み