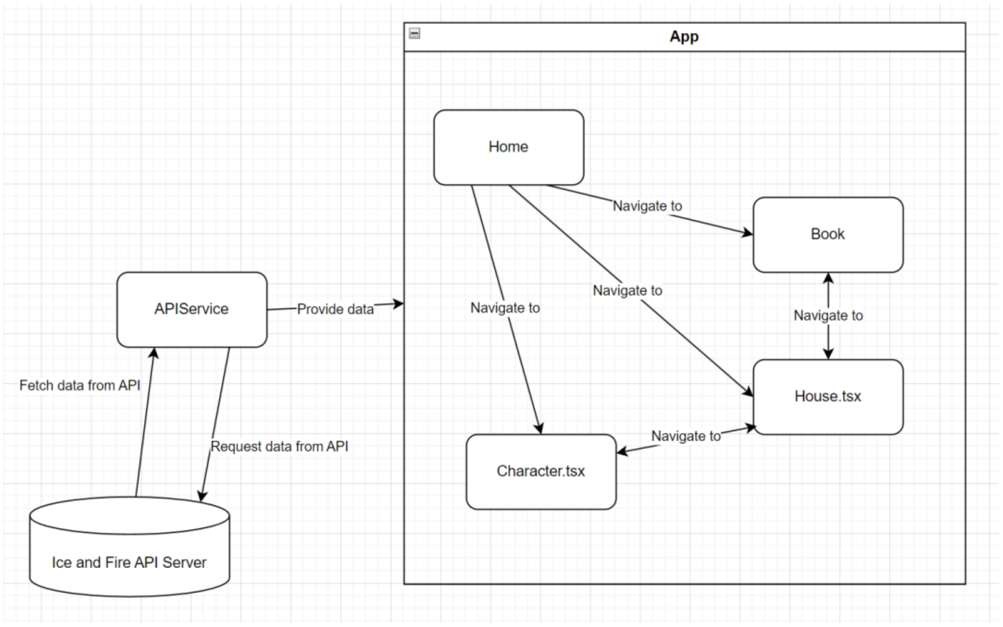
Game Of Thrones

1. Introduction of the Application

Game of Throne is a web application that created using React framework with TypeScript. This website allows the users to browse the information about Game of Thrones book series with the data provided by a public API called Ice and Fire API (https://anapioficeandfire.com/api). The user can search the name of their desired entity which is books, characters, or houses of Game of Thrones series and they will also be able to check the detail description in the detail page of each book, character, or house by clicking on one of the results that returned after searching using the keyword. Inside this detail page the user can also click on the related entity which will direct them to the detail page of the clicked entity.

2. Description of Architecture

This project is built based on the architecture which similar to Angular Project Architecture that consists of Functional components which used for the User Interface, Service component which responsible for fetching and processing the data for the API, and Utility component for storing reusable functions.



3. List of classes with Description

The Web application is consisting of 8 components such as:

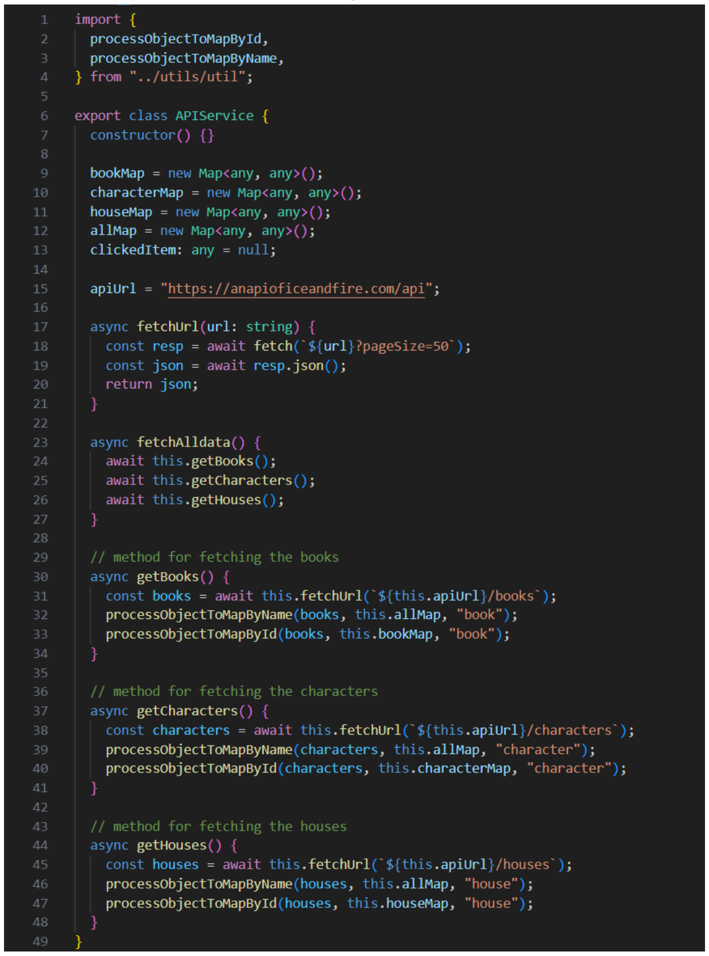
1. Home.tsx
   This component is responsible for searching the data from that being fetched from the API such as books, characters, or houses name in Game of Thrones Series.
2. Book.tsx
   This component is responsible for showing the detail description of the book that was selected by the user from the Home page or any other page that contain the book lists. The user may also find some other information like characters that were in this book and it can be selected to navigate to the detail page of that character.
3. Character.tsx
   This component is responsible for showing the detail description of the character that was selected by the user from the Home page or any other page that contain the character lists. The user may also find some other information like books that this character is in, and it can be selected to navigate to the detail page of that book.
4. House.tsx
   This component is responsible for showing the detail description of the house that was selected by the user from the Home page or any other page that contain the house lists. The user may also find some other information like the sworn members which is the character that is sworn to be the member of the house and it can be selected to navigate to the detail page of that character.
5. APIService.tsx
   This component is responsible for fetching all the data from the Ice and Fire API and provide these data to the application for displaying in the User Interface.
6. util.ts
   This component contained all the reusable functions such as
   - processObjectToMapByName() which assigning the object into the map while using the name as the key, this was used when we fetch all the data and assign it into one map.
   - processObjectToMapById() which assigning the object into the map while using the Id as the key, this was used when we fetch the data of books, characters, or houses and assign it into different maps for it specific type.

- grabId() which extracting the id from the URL by splitting the part with id. Ex: house/23

7. <u>App.tsx</u>
   This component is responsible for set up the Route and render all components by calling all the component. It allows the user to navigate from one page to another.
8. <u>index.css</u>
   This component is responsible for designing and decorating the user interface provide the spacing, alignment, effect, font properties (style, color, size,...etc), shadow and more to the website user interface.

4. Description of Client-Side Communication

Communication of this web application is done in the APIService component. This service will send HTTP requests (book, character, house) to the Ice and Fire API.



```ts
import {
  processObjectToMapById,
  processObjectToMapByName,
} from "../utils/util";

export class APIService {
  constructor() {}

  bookMap = new Map<any, any>();
  characterMap = new Map<any, any>();
  houseMap = new Map<any, any>();
  allMap = new Map<any, any>();
  clickedItem: any = null;

  apiUrl = "https://anapioficeandfire.com/api";

  async fetchUrl(url: string) {
    const resp = await fetch(`${url}?pageSize=50`);
    const json = await resp.json();
    return json;
  }

  async fetchAlldata() {
    await this.getBooks();
    await this.getCharacters();
    await this.getHouses();
  }

  // method for fetching the books
  async getBooks() {
    const books = await this.fetchUrl(`${this.apiUrl}/books`);
    processObjectToMapByName(books, this.allMap, "book");
    processObjectToMapById(books, this.bookMap, "book");
  }

  // method for fetching the characters
  async getCharacters() {
    const characters = await this.fetchUrl(`${this.apiUrl}/characters`);
    processObjectToMapByName(characters, this.allMap, "character");
    processObjectToMapById(characters, this.characterMap, "character");
  }

  // method for fetching the houses
  async getHouses() {
    const houses = await this.fetchUrl(`${this.apiUrl}/houses`);
    processObjectToMapByName(houses, this.allMap, "house");
    processObjectToMapById(houses, this.houseMap, "house");
  }
}
```

- fetchUrl() is a reusable function that take in a string parameter which is the url of the endpoint and is used for fetching the data from that specific URL by sending the HTTP request to the API and then it will store the response and it get from the fetch() function to the constant variable called resp after that it will parse this resp variable into a JSON format using the json() function then store this json format into a constant variable called json and return this variable.

5. API Usage

Whenever the user starts the application before the component is loaded, the app makes a HTTP request to the server then store the data that it received from the API in the map. Once the data are stored inside the map, the component will render and display the user interface. For the searching, when the user start typing in the search, the application will filter the data and left only the data that match the keyword that was input by the user. Once the user selects on any data that listed down, it will navigate to the detail page of that data.

```
import { useContext, useEffect, useState } from "react";
import { AppContext } from "../App";
import { Link, useNavigate } from "react-router-dom";
import { grabId } from "../utils/util";

export function Home() {
  const apiService = useContext(AppContext);
  const [data, setData] = useState(new Map());
  const navigate = useNavigate();
  const [searchTerm, setSearchTerm] = useState("");

  useEffect(() => {
    window.scrollTo(0, 0);
    apiService.fetchAlldata().then(() => {
      setData(apiService.allMap);
    });
  }, []);

  const handleChange = (event: any) => {
    setSearchTerm(event.target.value);
  };

  const handleClick = (itemData: any) => {
    apiService.clickedItem = itemData;
    const typeAndId = grabId(itemData.url).split("/");
    navigate(`/api/${typeAndId[0]}?${typeAndId[1]}`);
  };

  return (
    data.size > 0 && (
      <div>
        <div id="welcome">
          <Link to={"/"} className="title-link">
            <h1>Game Of Thrones</h1>
          </Link>
        </div>
        <div className="search-wrapper">
          <input
            type="search"
            id="search"
            placeholder="Search"
            value={searchTerm}
            onChange={handleChange}
          />
        </div>
        {searchTerm.length && (
          <div>
            {Array.from(data.entries())
              .filter(([itemName]) =>
                itemName.toLowerCase().includes(searchTerm.toLowerCase())
              )
              .map(([itemName, itemData], index) => (
                <div key={index} className="card" onClick={() =>
handleClick(itemData)}>
                  <a>{itemName}</a>
                </div>
              ))}
          </div>
        )}
      </div>
    )
  );
}
```