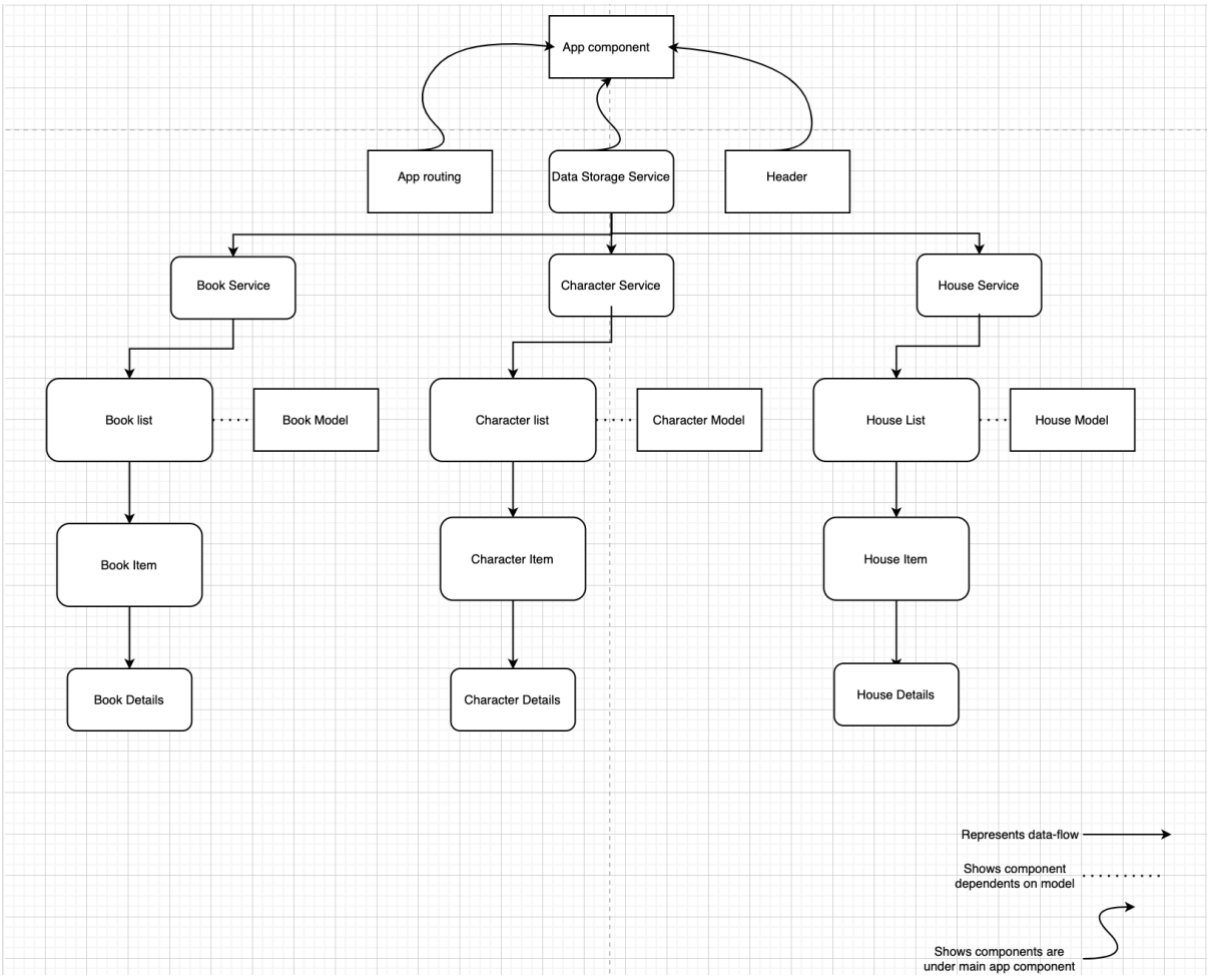
# GoT Angular Documentation

## Introduction

This app is written in Angular with TypeScript and its main purpose is to display information about the Game of Thrones book series in a pleasant and interactive way. All the data is taken from the free API of Ice and Fire which contains 'All the data from the universe of Ice and Fire you've ever wanted!'. Link: [https://anapioficeandfire.com](https://anapioficeandfire.com)
When starting the app the user can see a header with 3 choices: Books, Characters and Houses. Clicking any of them will display a list of the kind they chose. Each element displayed can be further clicked to see more details about it. The user can also perform a search among the elements displayed, the results will be updated with each key press.

## Architecture




App component – starting point of the application. Every component in the app is a child of the main app component.
App routing – module that makes routing possible inside the app. The various possible paths are defined here.

Header – header component that makes navigation between the main components possible.
Book, Character, House services – injectable services that hold the logic for managing their respective items.
Book, Character, House lists – list components that hold their respective items.
Book, Character, House items – item components that are displayed inside the lists. They are children of their respective list components.
Book, Character, House details – child of items components. They are displayed when item is clicked.
Book, Character, House models – define the respective objects displayed in the lists.

The dataflow in the app is from top – down in respect to the above diagram. Child components get the data from their parent components, those get the data from their respective service, which in turn all get it from the shared Data Storage Service.

## Classes

- Header component – The header component consists of an html and a ts file. The html file contains the design of the application header and the clickable elements to navigate between the different views. The ts file is responsible initiating the fetch data calls in the data storage service.
- App-routing – Module that holds the navigation paths and routing logic.
- Data storage – Shared, injectable service responsible for making API calls and sending the data to the respective services. Its fetch functions are called from the header's init method when the app starts. Each fetch function makes an asynchronous API call.
- Book service – injectable service that holds the logic for getting, setting and managing books. It contains a Subject to which the book-list component is subscribed. When the data fetching is completed successfully in the data storage component it is sent to this service. As soon as the data arrives here the subject notifies all its subscribers about the arrival of the data and they can then get it as well. This service also implement PipeTransform to enable dynamic searching among objects.
- Book list – Component consisting of html and ts files. The html file is responsible for displaying a list of books received in a ngFor loop. The .html file is also responsible for sending the book and its index to the book-item child component since it displays a list of book-items. The .ts file contans the logic for getting the data from the book service. Here in the ngOnInit we subscribe to the Subject of the book Service and when that is fired we fetch the list of books.
- Book item – Child component of Book list. Consists of .html and .ts file. The .ts file gets the book and index as @Input from the book list parent component. The .html file defines how a book item looks and provides clicking functionality and routing to the book details component when clicked.
- Book details – Child component of Book item. Consists of .html and .ts file. In the onInit of the .ts file we subscribe to the route.params chang when an item is clicked in the parent book item component. Here we will get the id of the item clicked and we can then get it from the book service.

- The service – parent – child structure and functionaility is similar for the Characters and Houses part.

## Client server communication

The client – server communication in my app is simple and straightforward. I have a shared, dedicated data-storage class to serve this purpose. The app makes only get requests to the API, which in turn sends structured responses in JSON format.

The data is fetched in an asynchronous way, using the HttpClient of Angular:

```
fetchBooks() {
  console.log('Fetching books!');
  // Fetch the books asynchronously and send them to the bookService afterwards.
  this.http.get<Book[]>( url: this.baseUrl + 'books?' + 'page=1&pageSize=50').subscribe( next: books => {
   console.log('Fetched books: ', books);
   this.bookService.setBooks(books);
 });
}
```

We are subscribed to the http call and as soon as we get the response we send it to the bookService where we further process and use it.