I-Translate Developer Documentation

Amani Brik
T3SJ6V

Client Side Technologies
2022.

Introduction and Main Functionality

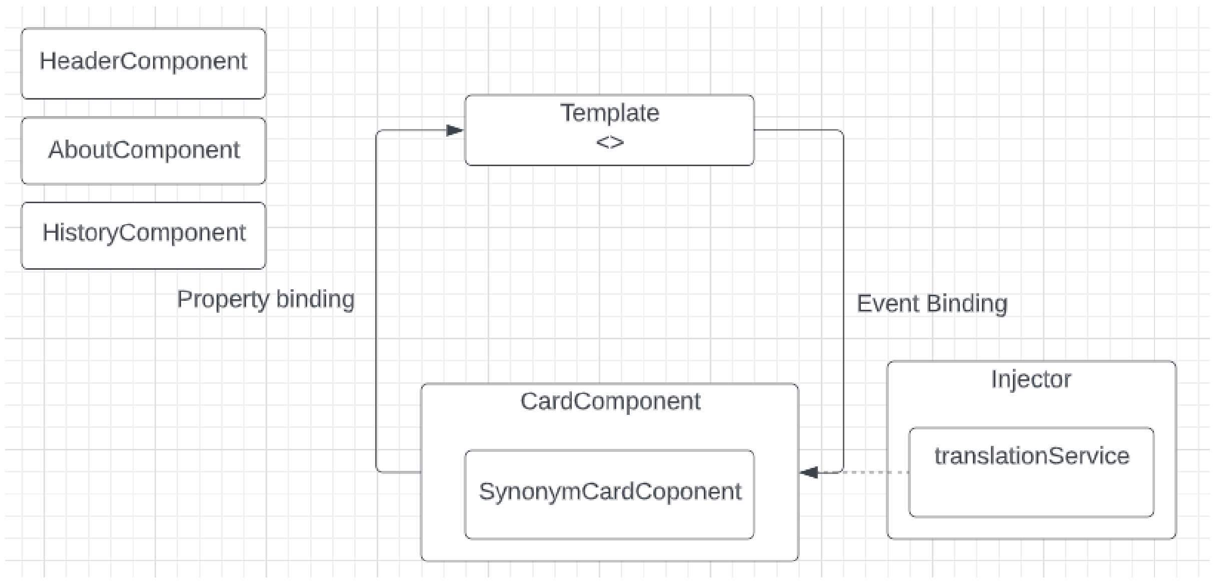
I-Translate is a Dictionary implemented using Angular. It uses the Yandex Dictionary API in order to get translations of words between 2 languages as well as synonyms and examples.

The app allows the user to choose the source language and target language through available lists, but not all translations are possible. It also allows the users to copy both their input and their result to the clipboard to make the usage of the dictionary more convenient in daily life.

The API accepts only one word in order to be functional, so this app works exactly like a normal dictionary would; it can only translate individual words. In fact, trying to translate a sentence using I-Translate would show you a message saying “Please enter only 1 word”.

The synonyms and the examples are also generated by the Yandex Dictionary API and they only appear when a word is being translated. The used API is not perfect, and thus, examples and synonyms are not always available. However, there are always some other similar words in the translation. These words are listed under “Other translations”.

Architecture



The Card component is the component that communicates with the translation service.

The Synonym Card component is a child of the Card component and is controlled by it. It receives input from the service indirectly through the card component.

The Header component is part of the AppComponent and is present throughout the whole app.

The About component appears only in certain occasions and is not one of the main components of the application.

The History Component is another page of the app that holds older translations and gets them from the local storage.

Components

Card:

This is the most important component in the entire app. The Card component contains 3 cards. Each card has its own importance and they all use the same information.

The Source card is the card that accepts the input from the user. It also has a dropdown list which lets the users choose their source language. That input is sent to the translation service once the user clicks on the “arrow” button.

Just like the Source card, the target card also has a dropdown list. This list lets the user choose their target language. The Target card does not accept any other user input, it just shows the first translation that the API gives.

The final card, which is called the Synonym card, is imported from another component that we will talk about later. This card accepts input from the Card component since it represents a child of the Card component. That input will then be used to show more information provided from the API.

Header

The Header component is always present in the application. It is used in the App Component and helps with the routing between the pages. The Header contains the Logo of the app, the name and a button that reroutes the user to the About page. When a user clicks on the logo, he/she will be rerouted to the main page.

SynonymCard

This class accepts input from its parent (the Card class) in order to prevent calling the API too often. The input structure is defined in a file under the services folder. The file is called `lookup.interface.ts` . Using that interface, it is easy to get the information we need. For example, in the implementation of this app, we could get some more possible translations, synonyms and example phrases with their own translations.

The synonym card component uses some angular material assets for the UI. It is more user friendly to have these long lists hidden under expansion panels than to let them be visible to the user. The comments in this class help understand the reasons behind some decisions better.

Pages/About

This page is simple. It contains an Angular material card that has some text and a button that lets us go back to the main page. The reason behind the creation of this page is to show how routing can be done.

Pages/History

The History page makes use of the local storage to display older translations. It also provides the possibility of deleting the content of the local storage.

Note: The word “page” might not be the most suitable for these classes since they still represent components. In fact, both About and History can be called from the App component through the `router-outlet`.

Services

Translation Service

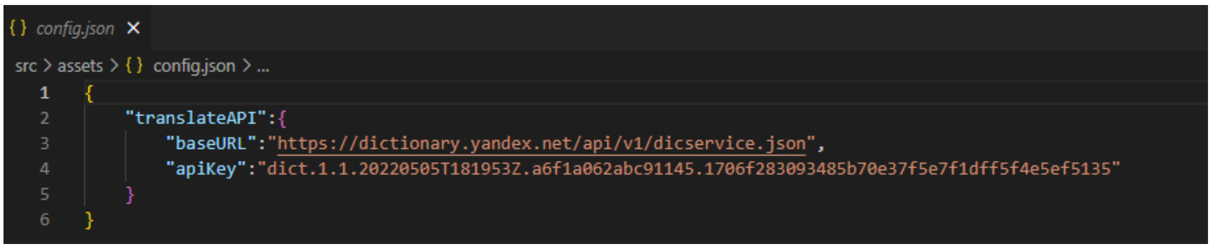
The translation service is the service that calls the Yandex Dictionary API. Its task is to implement the functions that get the configuration for the API (the base url and the api key), the available language translations and the translation results themselves. The configuration is not hard coded inside the service but is declared in a different file `config.json` . If the configuration changes, the file can be found under the assets folder.

A Config interface was created under the services folder to make it easier to deal with.

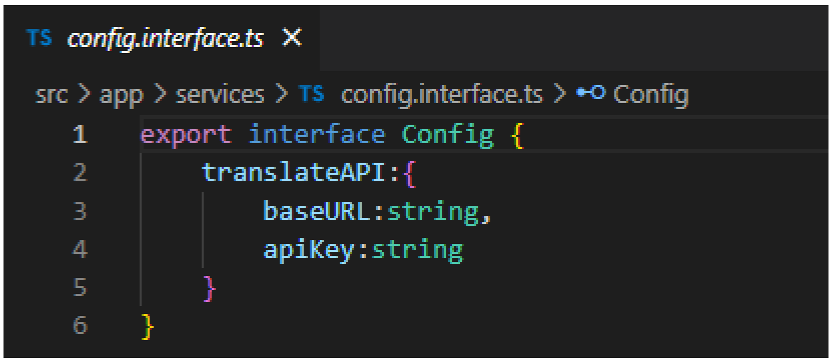
It is worth noting that the translation service makes use of asynchronous patterns and of different interfaces when calling the API. This will be discussed in detail in the next chapter.

Client-Server communication

The first thing that needs to be defined in a Client-Server Communication is the server configuration. For this reason, 2 files have been created in order to make accessing API configurations easier and more scalable. These files are: `config.json` and `config.interface.ts`. For this version of the app, we are currently using only 1 API (Yandex Dictionary API) that provides all the necessary functionalities. We called the API the translationAPI in the config files. As shown below



```json
{
    "translateAPI":{
        "baseURL":"https://dictionary.yandex.net/api/v1/dicservice.json",
        "apiKey":"dict.1.1.20220505T181953Z.a6f1a062abc91145.1706f283093485b70e37f5e7f1dff5f4e5ef5135"
    }
}
```



```typescript
export interface Config {
    translateAPI:{
        baseURL:string,
        apiKey:string
    }
}
```

The second step is to create the translation service through Angular. This service will make the API calls, so it has to use an HttpClient and, thus, we have to declare that in the constructor as following:

```typescript
constructor(private readonly http: HttpClient) {}
```

We can now implement functions that will do the API calls for us. The API that is used in this app has 2 main API methods documented: `getLangs` and `lookup`. The functions in the translationService have the same names as the API methods so that the developer knows what each of those methods does according to the documentation provided.

Documentation link if needed:
https://yandex.com/dev/dictionary/doc/dg/concepts/About.html

The implementation of the functions and the presentation of the results will be discussed in the next section but the rule of thumb for each of these functions is to get the configuration from the config files, add the endpoint that we need and use the resulting url to make an http get request in order to get the requested data.
Since we are using Typescript, the type of the result should be kept in mind as well. We could either use a string array type or create our own interface if the json results are too complex.

Translation Process

1. **API Call:**

For this section, the translation process is going to be discussed. From the API call until the integration of the data inside the HTML, this process is certainly the backbone of the whole application.

The first part of the process is the lookup method. This method is located in the translation service, as mentioned above. It makes an http get request to the API to get the translation using the user input. The return value is Promise<Lookup>.

Promise is used because the lookup method is asynchronous. This is useful so that the application does not stop working while waiting for the API response. The Lookup type is a type that has been created for this app to hold the expected results. It is defined under `lookup.interface.ts` .

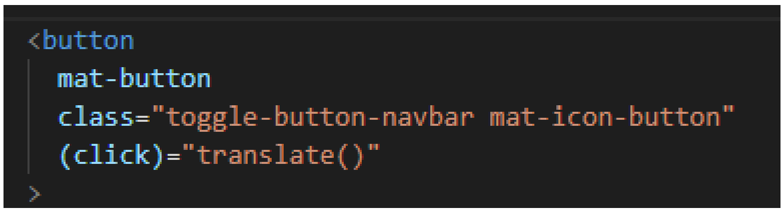
2. Service Call in the Card component

First, the Card component has to be able to use our created translation service. Therefore, we have to add our service to the constructor as following:

```
constructor(private readonly translationService: TranslationService) {}
```

Then, in the HTML file, we have to bind the input to an attribute that belongs to our class and implement a function that would trigger at the click event of our button.

The function that triggers at the click event is called translate().



```
<button
  mat-button
  class="toggle-button-navbar mat-icon-button"
  (click)="translate()"
>
```

This function will use the translation service and call the lookup function that we defined above. The parameters passed to the lookup method are the language pair as well as the user input that are defined as attributes in the Card component and are binded to the HTML.

The result is then added to the `translation` attribute that is binded to the target div in the html and is automatically shown on the screen. Naturally, we have to use our defined Lookup interface to be able to get the exact data that we need.

```
this.translation = this.lookup.def[0].tr[0].text;
```

The rest of the data (ex: synonyms and other translations) is passed to the synonym-card as input. Therefore, we don't have to make the API call twice; we can just share the data between our components.

Note: This function also registers the translation in the local storage. (If the translation is successful).