Research Document

Table of contents

Contents

Introduction

About the research

The goal of the research is to investigate how contents of the website can be translated on different languages with a simple click. The user should be able to read on their own language for a more comfortable experience which will in turn increase the probability that they will return in the future. The translation of the text should happen as easily as possible.

Approach

The question to be answered is: “How do we create a feature which allows the user to change the language of the website? ”

First, a list of subquestions will be created which will help in finding an answer for the dilemma. Then, we will look at each subquestion and try to answer it by using different techniques such as searching sources on the Internet and trying to

create application which can translate its content. And finally, we will write a conclusion to our findings and recommendations.

Research

Library research

First, a number of subquestions should be created:

1. What packages should be installed for the application and how should they be installed.
2. How do we define the translations on the different languages?
3. How do we extract the values that need to be displayed?
4. How do we change the language?
5. How do we make sure that the components are loaded correctly every time?

This website (https://blog.shahednasser.com/how-to-internationalize-a-react-app/) explains everything about how a React application can be internationalized and it gives answers to the subquestions. There is a video lesson on YouTube which also provides information on the subject (https://youtu.be/w04LXKlusCQ).

Answers

1) The packages that are needed for translations should be installed through the Terminal. The library, i18next, should be installed with the following command:

   npm install react-i18next 118next –save

   A language detector also needs to be installed to detect the language of the user.

   npm I i18next-http-backend i18next-browser-languagedetector

2) Create a new folder called “locales” in the public folder. Then create several different folders with different names for the languages (for example, “en” for English, “de” for German, etc). Then create a translation.json file for each folder and write the contents that should be included in the website and their translation:

For German, the file would look something like that:

```
{
  "greeting": "Hallo",
  "text": "Vielen Dank für Ihren Besuch auf unserer
Website.",
  "language": "Sprache"
}
```

And for English, it would look something like that:

```
{
  "greeting": "Hello",
  "text": "Thank you for visiting our website.",
  "language": "Language"
}
```

3) First, a class called i18nshould be created which looks like that:

```
import i18n from "i18next";
import { initReactI18next } from "react-i18next";
import Backend from 'i18next-http-backend';
import I18nextBrowserLanguageDetector from "i18next-browser-
languagedetector";

i18n
    .use(Backend)
    .use(I18nextBrowserLanguageDetector)
    .use(initReactI18next) // passes i18n down to react-
i18next
    .init({
        fallbackLng: 'en',
        debug: true,

        interpolation: {
            escapeValue: false // react already saves from
xss
        }
    });
```

```
export default i18n;
```
Then, call the class to the main page and create locale and assign a value to it like that:
```
 const [locale, setLocale] =
useState(i18n.language);
```
And create a class called LocaleContext with the following code:
```
import React from "react";

const defaultValue = {
   locale: 'en',
   setLocale: () => {}
}

export default React.createContext(defaultValue);
```
Then, surround the code in the LocaleContex.Provider tag like that:

```
<LocaleContext.Provider value={{locale, setLocale}}>
          <React.Suspense fallback={<Loading />}>
          some random code...
          </React.Suspense>
</LocaleContext.Provider>
```
In the different pages call the useTranslation hook like that:
```
import { useTranslation } from "react-i18next";
```
Assign its value to t:
```
const {t} = useTranslation();
```
And then use one of the values of t in the code itself (for example, the value ‘greeting’):
```
<h1>{t('greeting')}</h1>
```
And the code now displays the value that you have written in the translation files.

4) Put a function which would change the language:
```
function changeLocale (l) {
    if (locale !== l) {
        i18n.changeLanguage(l);
    }
}
```

We need to put a dropdown menu and the code for it would look something like that:

```
<Navbar.Toggle aria-controls="basic-navbar-nav" />

<Navbar.Collapse id="basic-navbar-nav">

    <Nav className="me-auto">

        <NavDropdown title={t('language')} id="basic-nav-
dropdown">

            <NavDropdown.Item href="#" onClick={() =>
changeLocale('en')}>English</NavDropdown.Item>

            <NavDropdown.Item href="#" onClick={() =>
changeLocale('ar')}>العربية</NavDropdown.Item>

            <NavDropdown.Item href="#" onClick={() =>
changeLocale('de')}>Deutsch</NavDropdown.Item
>
        </NavDropdown>

    </Nav>
</Navbar.Collapse>
```

However, it's not always necessary to make this functionality with a dropdown menu. It can be done with simple buttons or with other elements as well.

In the changeLocale function in the NavDropdown tag write the name of the folder of the language that should be chosen when the user clicks a specific item. The changeLocale function then takes the value and selects the language in question.

5) Surround the entire code in a React.Suspense tag to make sure it loads properly. In the fallback function write the name of the page which should load in case of any trouble like that `<React.Suspense fallback={<Loading />}>.`

Workshop

Step 1

Workshop research was carried out to confirm the answers that were found for the subquestions and to test if everything works properly. The first part of the research was done by creating a new React application from scratch by following the instructions on the website. After the application was finished, it was confirmed that the translation function works properly.

Step 2

After the results of the research were confirmed, the translation features were integrated in the group project as well. There were a lot of problems which occurred during this step but after a long time they were resolved very easily. The problem was that the class which was used wasn't importing from the correct place which prevents the language of the application from being changed. Sometimes, after the user types its name in the code, it imports from a library called "118n" by assumption. Therefore, it's always important to check if i18n always imports from the correct place.

Conclusions

The translations should be declared in separate folders in the public folder. Two classes by the names of LocaleContext and i18n should be created which allows the user to use the translations and to change the language. The useState hook should be used to take the language property from i18n and assign it to locale and also an event listener “languageChanged” should be used to change the value when the user selects a different language. The entire code in the main page should be surrounded by a LocaleContext.Provider tag and a React.Suspense tag. A dropdown menu should be created and when an item is selected, a function called “changeLocale” is also initiated. In this function, the language which is used by the application is changed with the language chosen by the user. Then we should use the useTranslation hook to take the language that is currently chosen and display it in the code itself.

Recommendations

When importing i18n the from the i18n class, it should always be checked if the application is importing from the correct place because sometimes it imports from a library called “118n” by assumption and in that case, the application wouldn’t work.

The folders should have understandable names that indicate what languages they contain. The text files should also contain understandable names for the values they contain. The different translation files should contain the exact same values because if there are differences, it might cause issues for the app.

The contents of the page should always be covered in a React.Suspense tag because otherwise the application won’t always load properly and most of the time, there would be just a white background with no content on it.

Bibliography

How to Internationalize a React App from Shahed Nasser. Retrieved from: https://blog.shahednasser.com/how-to-internationalize-a-react-app/

React Multi Language App - i18next Tutorial. Retrieved from: https://youtu.be/w04LXKlusCQ