Pixabay Finder with TypeScript, Material-UI and Thinking-in-React

A proof of concept experimental project to show how to apply the concepts of Thinking-In-React using TypeScript and Material UI.

The project was inspired by both Thinking in React https://reactjs.org/docs/thinking-in-react.html as well as a quite decent JavaScript training video by Traversy Media (React & Material UI Project Using The PixaBay API, www.youtube.com/watch?v=dzOrUmK4Qyw)

1 Project Setup

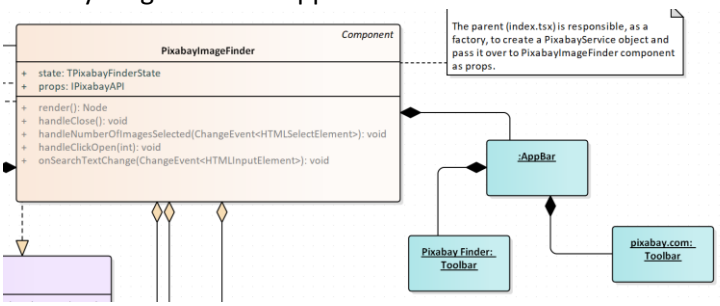
- Create project pixabayfinder
- Clone with VSC
- Open terminal: npx create-react-app pixabayfinder –typescript
- cd pixabayfinder
- npm start
- commit with VSC: **Project created and initialized with create-react-app typescript option**

2 Getting Started with Material UI

- npm install @material-ui/core @material-ui/icons
- Delete render content of App.tsx
- `import Button from "@material-ui/core/Button"`
- `<Button variant="contained" color="secondary">Pixabay Image Finder <img src={logo} className="App-logo" alt="logo" width="30px" /></Button>`
    - Just copy the image from the default App content.
    - Let's see how TypeScript prevent us from writing faulty code.
- Upon a compilation error, add "target": "es2015" to tsconfig, and **stop npm start**.
- Let's have a look at what we have built.
- Commit: Material UI installed.

3 Getting Started with Design 1st

Open EA and check out the application design: PixabayImageFinder in App.tsx



4 Implement AppBar with the Title and Image Toolbar

```
<AppBar position="sticky">
   <Toolbar>
     <img src={logo} className="App-logo" alt="logo" width="56px" />
     <h6>Pixabay Image Finder</h6>
```

- Let's add a number of lorem blocks of text to see if the app bar doesn't scroll off.

5 Replace H6 with Typography

```
Comment out the H6 section
 <Typography variant="h6" noWrap color="inherit">Pixabay Image Finder</Typography>
```

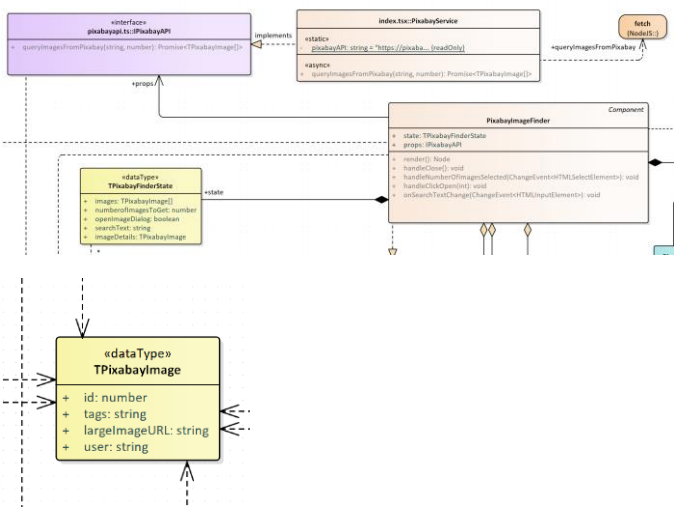
6 Add Pixabay Toolbar

"If you make use of the API, show your users where the images and videos are from, whenever search results are displayed. A link to Pixabay is required and you may use our logo for this purpose. That's the one thing we kindly request in return for free API usage."

The link is coming from https://pixabay.com/en/service/about/#goodies

```
<Toolbar>
  <a href="https://pixabay.com/">
    <img src="https://pixabay.com/static/img/public/leaderboard_b.png" width="100%" alt="Pixabay"/>
```

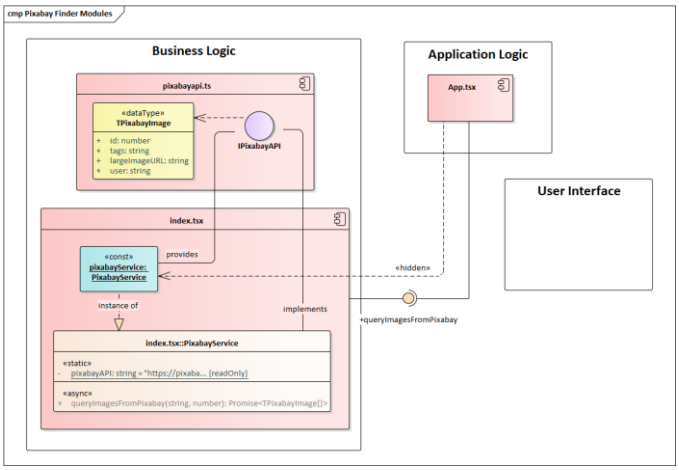
7 Show Business Logic Design



- TPixabayImage type definition (it comes from https://pixabay.com/api/docs/)
- IPixabayAPI interface (async), which is going to be the type of the props of the App component.
- PixabayService
- The props type of
- TPixabayFinderState with the images field.

8 Show Module Architecture

It is simplified for this tutorial.



- We're making a new module **pixabayapi.ts** for the image type definition and the interface function.
- Index.tsx is extended to be our business logic provider, a factory module for the PixabayService.
- App is the top level application logic component, receiving the interface from its parent, index.tsx
- User interface is coming later.

9 Write pixabayapi.tsx according to the specification

```
export type TPixabayImage = {
  id: number,
  tags: string,
  largeImageURL: string,
  user: string,
}
export interface IPixabayAPI {
  queryImagesFromPixabay(searchText:
string, numberofImagesToGet: number):
Promise<TPixabayImage[]>,
}
```

10 Implement PixabayService in index.jsx

10.1 Import Interface Definitions

```
import {TPixabayImage,IPixabayAPI} from
"./pixabayapi"
```

10.2 Add Pixabay AKI Key

From https://pixabay.com/api/docs/

```
class PixabayService implements
IPixabayAPI {
  private static readonly pixabayAPI =
https://pixabay.com/api/?key=126338-
8e2f836ed7b71bbd3fd183c37
```

10.3 Add Async Query Function Returning a Promise

```
public async
queryImagesFromPixabay(searchText: string,
numberofImagesToGet: number):
Promise<TPixabayImage[]> {
```

10.4 When Search Text is Defined, Compose the Query String According to the Pixabay Specification

```
if(searchText) {
      const q = PixabayService.pixabayAPI
+ "&image_type=photo&per_page=" +
numberofImagesToGet + "&q=" +
encodeURIComponent(searchText)
```

10.5 Use a Double Await Fetch to Get the Hits

```
try {
  const r = await fetch(q)
  const v = await r.json()
```

```
  return v.hits
```

10.6 Upon Errors Write Console Log and Rethrow the Error

```
} catch(reason) {
console.log("queryImagesFromPixabay:Query:
" + q, reason)
  throw new Error(reason)
```

10.7 Upon Empty Search Text, Return Empty Array

```
} else {return []}
```

10.8 Define a Const pixabayService Object

```
const pixabayService = new
PixabayService()
```

- Actually, this is the factory operation.

10.9 Pass the Interface from the API Service Object to the Application Logic Component as a Prop

```
ReactDOM.render(<App
queryImagesFromPixabay=
{pixabayService.queryImagesFromPixabay}/>,
```

11 Connect the App to the Service

11.1 Import API

```
import {TPixabayImage,IPixabayAPI} from
"./pixabayapi"
```

11.2 Define State Type for App

```
type TPixabayFinderState = {
  images:TPixabayImage[],
}
```

12 Add Type Parameters for Props and State

```
export default class App extends
React.Component<IPixabayAPI,TPixabayFinder
State> {
  public state:TPixabayFinderState = {
    images:[]
  }
```

12.1 Let's Call Our Service to See if It Works

```
async componentDidMount() {
const images = await
this.props.queryImagesFromPixabay("dogs",1
5)
this.setState({images})
```

12.2 What to Show While Waiting for the Query Results?

```
{!this.state.images.length && <div>Loading
Images ...</div>}
```

12.3 Quick Display of Query Result

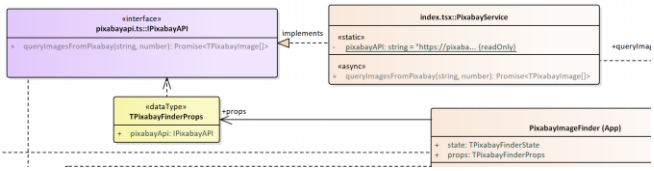
```
{this.state.images.map((i) => {
 return (<div key={i.id}>{i.tags} by
{i.user} <img src={i.largeImageURL}
width="100%"/></div>)
```

13 How TypeScript Helps Building a Maintainable and Scalable Business Application?

- Change the name of the user field in the TPixabayImage type, to see how VSC shows the errors.
- Compilation checks errors, too.
    - When you change only a file with only type definitions it doesn't enforce recompilation.

14 More Flexible API Props for App

14.1 Review the Improved Design



- New type definition for the App: TPixabayFinderProps with a field for the entire API.
- This is a nice demonstration how TypeScript (or Flow as a matter of fact) can help us a lot to perform a major refactoring in a safe way, which is nearly impossible with plain JavaScript. After we changed the primary location, VS Code together with TypeScript is automatically guiding us through all the required modifications.

14.2 In index.tsx Change Inferred Type of pixabayService to IPixabayAPI

- const pixabayService: IPixabayAPI = new PixabayService()

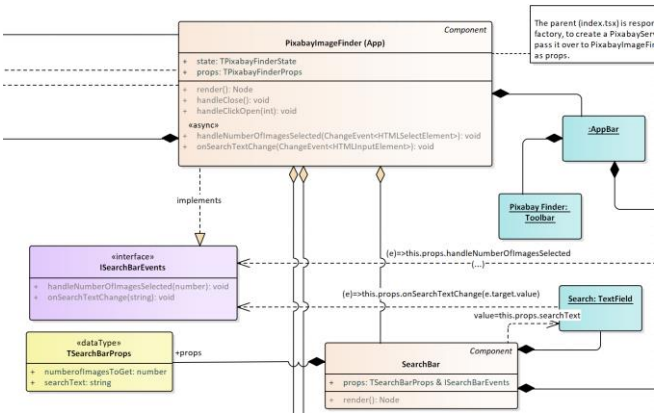
14.3 Add New Type to App.jsx and Change Props Type

- type TPixabayFinderProps = { pixabayApi: IPixabayAPI,}
  It doesn't have to be exported, since when the App is imported by index.tsx, the type definition is implicitly imported, too.
- React.Component<**TPixabayFinderProps**, …
- (in componentDidMount) this.props.**pixabayApi**. queryImagesFromPixabay

14.4 Review and Commit Changes

15 Adding Search Text

15.1 Review Design 1st



1. Interface ISearchBarEvents with onSearchTextChange. Note that the type of the parameter is string. Interface functions should be generic and serializable, as much as possible. Here, the App component doesn't care the details of what machinery the Search component uses to deliver search texts.
2. Props type TSearchBarProps with searchText: string
3. The App component implements async onSearchTextChange. It's async, since it calls the async queryImagesFromPixabay, just like in componentDidMount.
4. The SearchBar component creates a TextField and when being rendered, it simply, links the value property of the text field directly to the props.searchText received from the App component.
5. When the user enters text, for each change the onSearchTextChange is called.

15.2 Programming Search Bar with Material UI

1. Write ISearchBarEvents according to the design specification:
   interface ISearchBarEvents {
   onSearchTextChange(searchText:string):void
2. Extend TPixabayFinderState
   type TPixabayFinderState = {…,
   searchText: string,
3. Add ISearchBarEvents implementation to class **App** … **implements ISearchBarEvents**
4. Extend **App state** TPixabayFinderState = {…, **searchText: ""**
5. Add async interface implementation lambda function **public onSearchTextChange = async (searchText:string) => {**

this.setState({searchText: searchText})
const images = await this.props.pixabayApi.queryImagesFromPixabay(searchText,15)
this.setState({images})

It's almost the same as *componentDidMount*

6. Change componentDidMount to call: **await this.onSearchTextChange("dogs")**
7. Now, give it a try!
8. Write type **TSearchBarProps** = { searchText: string,
9. Write class **SearchBar** extends React.Component<**TSearchBarProps & ISearchBarEvents**> {
   Note the concatenation of the two types; we don't have to write an additional type definition. TypeScript and Flow are the only languages that support this kind of type concatenation.
10. Import Material UI components: import **TextField** from "@material-ui/core/TextField"
    import **InputAdornment** from "@material-ui/core/InputAdornment"
11. Add SearchBar render function:
     <**TextField** label="**Search**" value={**this.props.searchText**} style={{**marginTop:8**}} **InputProps**={{ endAdornment: <InputAdornment position="end"><**SearchIcon**/></InputAdornment>,}} helperText="Start typing search string" fullWidth={true} />
12. Then add **onChange**={(e) => this.props.**onSearchTextChange**(e.target.value)}

15.3 Compare with Thinking in React Sample

Section 5 has the code block for the Search Bar:

```
class SearchBar extends React.Component {
constructor(props) {super(props);
  this.handleFilterTextChange =
    this.handleFilterTextChange.bind(this);
  this.handleInStockChange =
    this.handleInStockChange.bind(this);  }
handleFilterTextChange(e) {
  this.props.onFilterTextChange(e.target.value); }
handleInStockChange(e) {
  this.props.onInStockChange(e.target.checked); }
render() { return ( <form>  <input  type="text"
placeholder="Search..."
value={this.props.filterText}
onChange={this.handleFilterTextChange} />
```

The only difference is that we use a lot simpler lambda syntax, which might have slight performance penalty compared to the cumbersome function binding solution.

15.4 Review the Changes on the Source Control Panel of VS Code

Then commit changes with message: **Search Bar with Search Text Field added**.