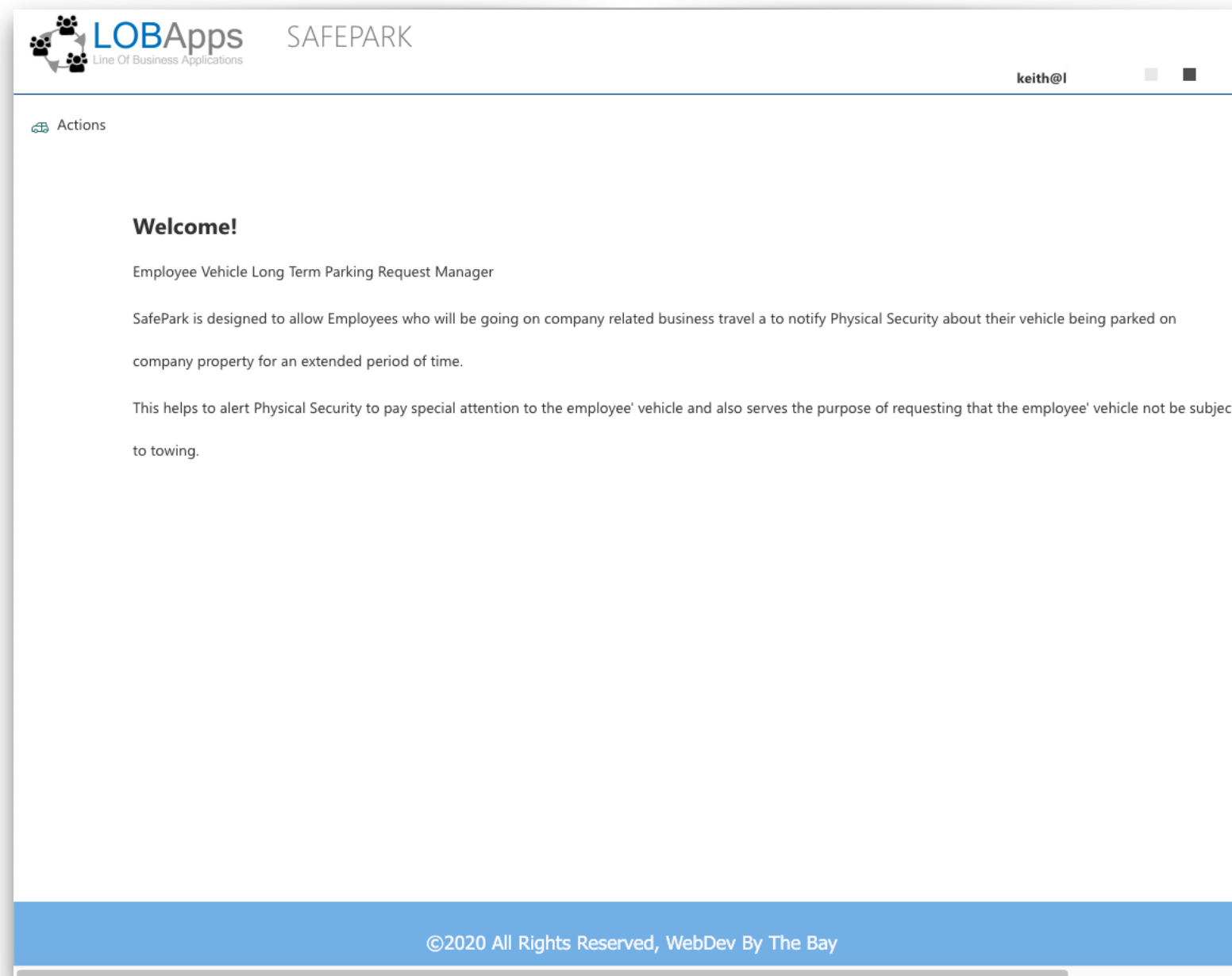


2020



Developing SharePoint Apps

Where To Start

Section 01

App Model Landscape

App Models - Provider Hosted

Provider Hosted

Microsoft supported!

No longer recommended!

Your code resides on your server and makes calls to SharePoint to surface SharePoint information.

Challenges:

1. Complexity - Your app must be registered with Active Directory to obtain what's called Access and Refresh Tokens.
2. Your app will need to manage the tokens and user credentials.
3. Your app must also be registered in the SharePoint App Catalog before you can interact with SharePoint resources.
4. If you make changes to your app, someone with SharePoint Administrator permissions will need to upload your changes to the SharePoint App Catalog.
5. No guarantee that your app will not break as new SharePoint versions are rolled out.

SharePoint Hosted

Microsoft supported!

No longer recommended!

Your code resides in SharePoint.

Challenges:

1. Your app must be registered in the SharePoint App Catalog before you can interact with SharePoint resources.
2. If you make changes to your app, someone with SharePoint Administrator permissions will need to upload your changes to the SharePoint App Catalog.
3. No guarantee that your app will not break as new SharePoint versions are rolled out.

SharePoint Framework

Microsoft supported!

Microsoft states that the SharePoint Framework or more commonly referred to as SPFx, is the only app model they guarantee not to break your app when new SharePoint versions are rolled out.

Your code is developed on your computer and the compiled code resides in SharePoint.

In my opinion this is a great step forward from the Provider and SharePoint Hosted models:

1. Your app must be registered in the SharePoint App Catalog before you can interact with SharePoint resources but it's not required to register your app with Azure active Directory.
2. If you make changes to your app, someone with SharePoint Administrator permissions will need to upload your changes to the SharePoint App Catalog.
3. You can test your app on your own computer within limits, you won't be able to surface information from SharePoint resources but you can setup mock data to simulate the experience until you either test your app in the SharePoint Workbench or publish to the SharePoint App Catalog.
4. You develop what is known as web parts, using modern web stack technologies, i.e. plain JavaScript, TypeScript or one of the many JavaScript Frameworks.
5. Microsoft guarantee's that your app will not break as new SharePoint versions are rolled out.

App Models - Microsoft Power Apps

Microsoft Power Apps

Microsoft supported!

Not really an app model but Power Apps is Microsoft's latest Low Code App Development Platform.

Power Apps is incredibly versatile, you can develop SharePoint Lists forms, standalone apps for Web and Mobile, Portals, AI Apps, Virtual Reality Apps, Embedded Apps, Single purpose apps to full blown Enterprise level Hub Apps.

Power Apps is part of what Microsoft calls their Power Platform.

Power Apps - Microsoft calls this Low Code but you develop with Excel like Formulas and drag and drop controls.

Power Automate - Workflows

Power Virtual Agents - Work with ChatBots and Virtual Assistants.

Power BI - Reports and Analytics

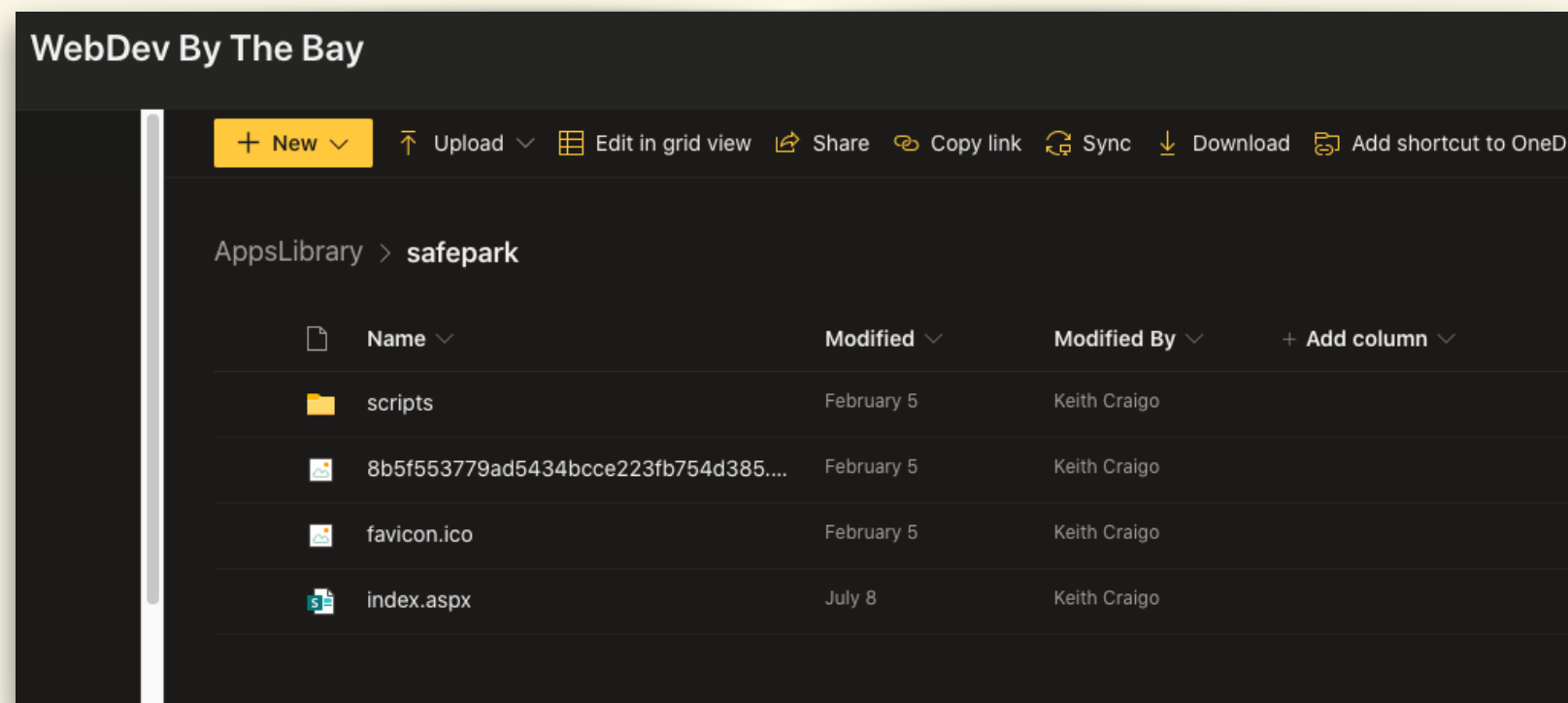
More info here: <https://powerusers.microsoft.com/>

App Models - SharePoint Folder Hosted

SharePoint Folder Hosted Apps

Not Microsoft supported!

May break as new versions of SharePoint are rolled out.



Section 02

SharePoint Folder Hosted Apps

SharePoint Folder Hosted Apps - Example

SafePark

SafePark is the name of the SharePoint Folder Hosted example app I will walk through in this book.

NOTE: You will need an **AppID** therefore you must register your version of this app with your Azure Active Directory App Registrations.

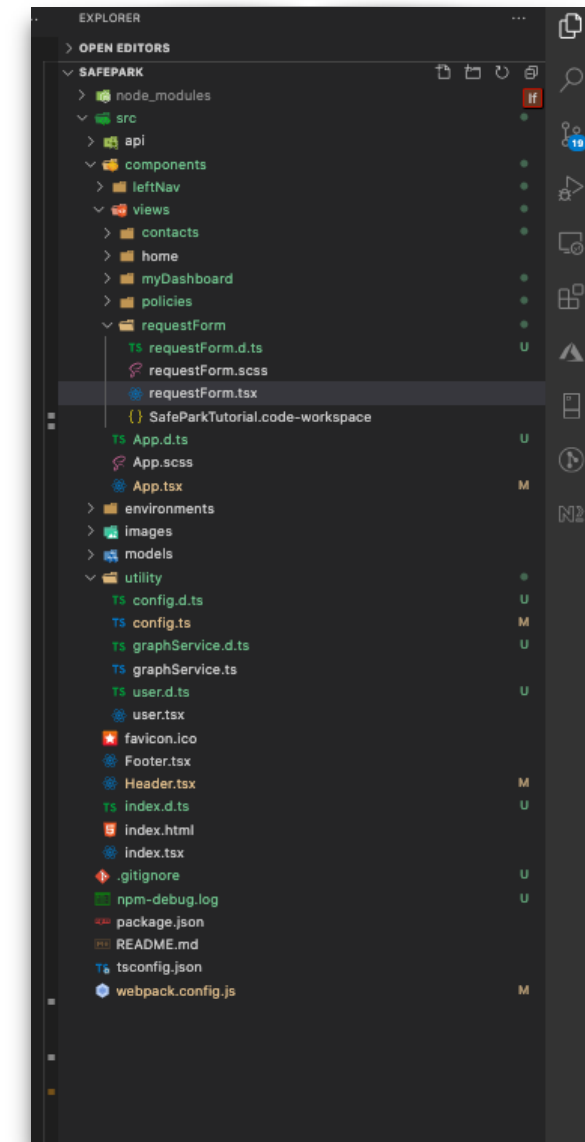
Why choose this model over one of the Microsoft supported models?

Permissions, see Provider and SharePoint Hosted Models above. Yes these permissions are necessary to prevent someone from either on purpose or accidentally compromising the company's Intellectual Property.

But requiring someone with elevated permissions to update the SharePoint App Catalog entry for your App just because you made a change to the color or weight of a font is overkill, non productive and a downright pain point.

Microsoft Power Apps is a fantastic approach as long as your data is being surfaced from SharePoint and or the O365OfficeUsers connector.

If you need to surface information from another data source then you or your organization will need to purchase additional licenses.



SafePark provides a way for your co-workers who travel on a business trip to notify Security that they need to park their vehicle on Company premises for a specified time range.

SharePoint Folder Hosted Apps - Example Form

SafePark


SafePark is developed with TypeScript, ReactJs, SharePoint, PnPJs, Fluent UI and the Microsoft Graph

Getting Started!


The React Hooks version for SafePark can be found on my [SafePark github repository](https://github.com/kcraigo/SafePark) or

<https://github.com/kcraigo/SafePark>

SafePark

 **Parking Policies** ☐ Please make sure you have read and understand the Parking Policy!

All Fields are Required!

 A Mobile App version of this form is available from the Power Apps Viewer. The viewer can be downloaded from the App Store or Google Play device.

Initiator Information

Employee:	Email:	Title:	Phone:
Keith Craigo	<input type="text"/>	CEO and Founder	<input type="text"/>

Employee Information

Employee First Name	Employee:	Title	Phone	EmployeeID
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

©2020 All Rights Reserved, WebDev By The Bay

SharePoint Folder Hosted Apps - Create a Project

SafePark

Pre Requisites:

1. Node - <https://nodejs.org/en/> - I recommend the LTS version

Create a new Project

1. You can either download the React Hooks version from <https://github.com/kcraigo/SafePark> or on the command line in a folder of your choice type **npx create-react-app safepark**

Shortcut TIP!

Windows - File Explorer url bar, of the folder, type cmd to open a terminal window.

MAC - Right click your folder and select New Terminal at Folder

2. Then open **package.json** and replace the contents with either the contents of the package.json file in the <https://github.com/kcraigo/SafePark> project or copy the code on the next page.

SharePoint Folder Hosted Apps - package.json

SafePark

```
{
  "name": "safepark",
  "version": "1.0.0",
  "description": "Parking Request Manager",
  "scripts": {
    "build": "webpack",
    "start": "webpack-dev-server --open --history-api-fallback"
  },
  "keywords": [],
  "author": "Keith Craig",
  "license": "ISC",
  "devDependencies": {
    "@babel/core": "^7.7.7",
    "@babel/preset-env": "^7.7.7",
    "@microsoft/microsoft-graph-types": "^1.12.0",
    "@types/es6-promise": "^3.3.0",
    "@types/node": "^10.17.11",
    "@types/react": "^16.9.17",
    "@types/react-dom": "^16.9.4",
    "@types/react-router": "^5.1.3",
    "@types/react-router-dom": "^5.1.3",
    "awesome-typescript-loader": "^5.2.0",
    "clean-webpack-plugin": "^0.1.19",
    "copy-webpack-plugin": "^4.6.0",
    "css-loader": "^0.28.11",
    "eslint": "^6.8.0",
    "eslint-plugin-react": "^7.17.0",
```

! Depending on when you are reading this, there may be newer versions of the node modules you see on this page. For help on updating, <https://nodejs.dev/learn/update-all-the-nodejs-dependencies-to-their-latest-version>

```
"expose-loader": "^0.7.5",
"file-loader": "^1.1.11",
"html-webpack-plugin": "^3.2.0",
"node-sass": "^4.9.3",
"office-ui-fabric-react": "^6.211.0",
"react": "^16.12.0",
"react-dom": "^16.12.0",
"react-hooks": "^1.0.1",
"react-router": "^5.0.1",
"react-router-dom": "^5.1.2",
"sass-loader": "^7.3.1",
"style-loader": "^0.21.0",
"typescript": "^3.7.4",
"url-loader": "^1.1.2",
"webpack": "^4.41.4",
"webpack-cli": "^3.3.10",
"webpack-dev-server": "^3.10.1",
"msal": "^1.2.1"
},
"dependencies": {
  "@babel/polyfill": "7.6.0",
  "@microsoft/sp-core-library": "1.9.1",
  "@microsoft/sp-lodash-subset": "^1.9.1",
  "@pnp/common": "^2.0.3",
  "@pnp/graph": "^2.0.3",
  "@pnp/logging": "^2.0.3",
  "@pnp/odata": "^2.0.3",
```

```
"@pnp/sp": "^2.0.0",
"@types/es6-promise": "0.0.33",
"@uifabric/example-data": "^7.0.2",
"@uifabric/react-hooks": "^7.0.1",
"babel-loader": "^8.0.6",
"es6-promise": "^4.2.8",
"eslint-plugin-react-hooks": "^2.3.0",
"isomorphic-fetch": "^2.2.1",
"node-sass": "4.13.0",
"popper.js": "1.16.0",
"prop-types": "15.7.2",
"query-string": "^6.8.3",
"querystringify": "^2.1.1",
"react-app-polyfill": "1.0.4"
}
}
```

On the command line in the same dir as package.json, execute either `npm install` or the shortcut, `npm i`

SharePoint Folder Hosted Apps - Webpack

SafePark

3. Next create the **webpack.config.js** file. Webpack is our packager, minifier and bundler. Copy the following code or use the webpack file in github.

```
const webpack = require("webpack");
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const CopyWebpackPlugin = require('copy-webpack-plugin');
const CleanWebpackPlugin = require('clean-webpack-plugin');
const rand = Math.floor(Math.random() * 999999);

module.exports = {
  entry: ['@babel/polyfill', './src/index.tsx'],
  output: {
    filename: 'scripts/bundle'+rand+'.js',
    path: path.resolve(__dirname, 'dist'),
    //,publicPath: '/' //CURRENTLY FAILING FOR LOCAL.
    //we need this for local testing, REMOVE Before Bundle
  },
  resolve: {
    extensions: ['.js', '.json', '.ts', '.tsx'],
  },
  devServer: {
    historyApiFallback: true,
  },
  plugins: [
    // UNCOMMENT FOR BUILD:
    new HtmlWebpackPlugin({ template:
path.join(__dirname, 'src', 'index.html'), filename: "./index.aspx" }),
```

```
// COMMENT OUT FOR BUILD - WE ONLY NEED THIS FOR LOCAL TESTING ON A MAC
    // new HtmlWebpackPlugin({ template: path.join(__dirname, 'src',
    'index.html') }),
    new webpack.DefinePlugin({
      "process.env.API_URL":JSON.stringify("https://lobapps.sharepoint.com/demo")
    }),
    new CleanWebpackPlugin(['dist']),
    new CopyWebpackPlugin([{ from: './src/favicon.ico', to: 'favicon.ico' }])
  ],
  module: {
    rules: [
      { test: /\.js$/, enforce: 'pre', exclude: /node_modules/,
        use: [{ loader: `babel-loader`, query: { presets:
['@babel/preset-env', '@babel/preset-react'] }}}],
      { test: /\.ts$/, loader: 'awesome-typescript-loader' },
      { test: /\.css$/, use: ['style-loader', 'css-loader' ]},
      { test: /\.scss$/, use: ["style-loader", "css-loader", "sass-loader" ]},
      { test: /\.(png|jpg|gif)$/, use: [{ loader: 'url-loader',
options: { limit: 8192 } }] }
    ],
  },
  mode: "development",
  devtool: 'source-map',
  devtool: 'cheap-eval-source-map'
};
```

SharePoint Folder Hosted Apps - Cache

SafePark

Side Note! In the **webpack.config.js**, the lines

```
const rand = Math.floor(Math.random() * 999999);
```

```
module.exports = {  
  entry: ['@babel/polyfill', './src/index.tsx'],  
  output: {  
    filename: 'scripts/bundle'+rand+'.js',  
    path: path.resolve(__dirname, 'dist')  
  }  
}
```

We create a const called **rand**, this is a random number that is guaranteed to never repeat. Then we append the rand const onto the generated filename. The generated filename is the name that webpack will save the bundled and minified script file for our project as.

Why do we need to use the const rand, why can't we just name the file with a static name?

When you save a file in SharePoint, SharePoint may serve the previously cached version of the file, that cached version could be in one of the trash bins, as long as it exists, SharePoint may serve it.

One way to prevent SharePoint from showing your users stale data is to delete the cached version from the trash bins, there are two trash bins in SharePoint.

The other way is to upload your file with a completely different name, the rand const ensures that our filename will be different on each build.

Another way to auto generate a new filename is to use the webpack feature called **Chunkhash**. Basically webpack saves your file with filename + a hash of the contents of the file, anytime the contents change the hash is updated on each build. I think this is a much more elegant approach. *Remove the const rand line then change*

filename: 'scripts/bundle'+rand+'.js' to **filename:** 'scripts/bundle.**[contenthash].js**', run a build, I use npm run build, your filename should now be something like **bundle.1476e77f5d11071fd606.js**.

Each time you make a change to one of your files, save and run build again, the filename hash will be different,
bundle.a9c57915307d4a917341.js

SharePoint Folder Hosted Apps - ChunkHashing

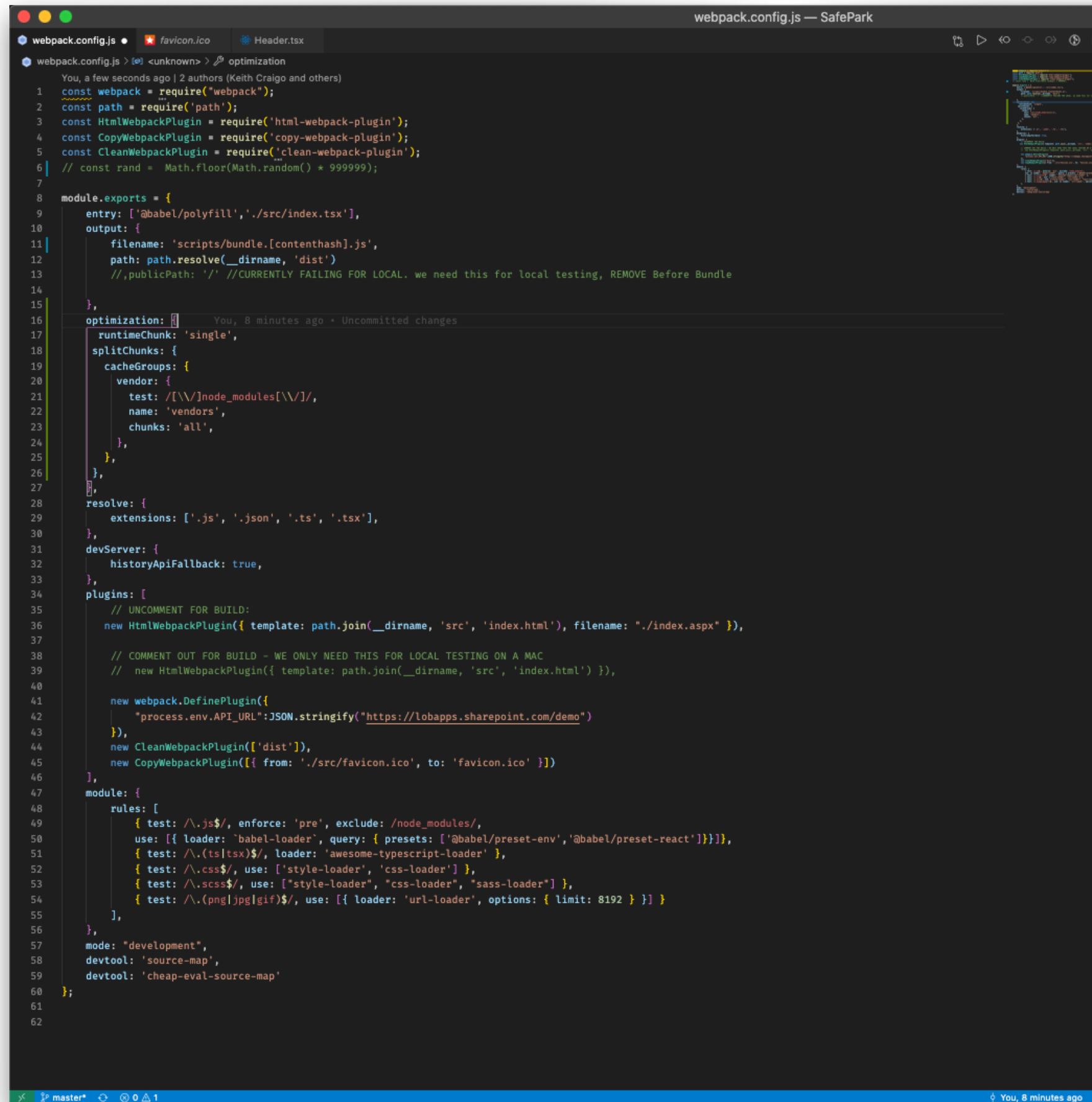
SafePark

Another great feature of Chunkhashing is that you can separate any third party vendor code from your code's hash.

This reduces the amount of code your viewers will need to download.

More information can be found in the Webpack documentation, <https://webpack.js.org/guides/caching/#output-filenames>

Now our **webpack.config.js** file should look like the following screenshot



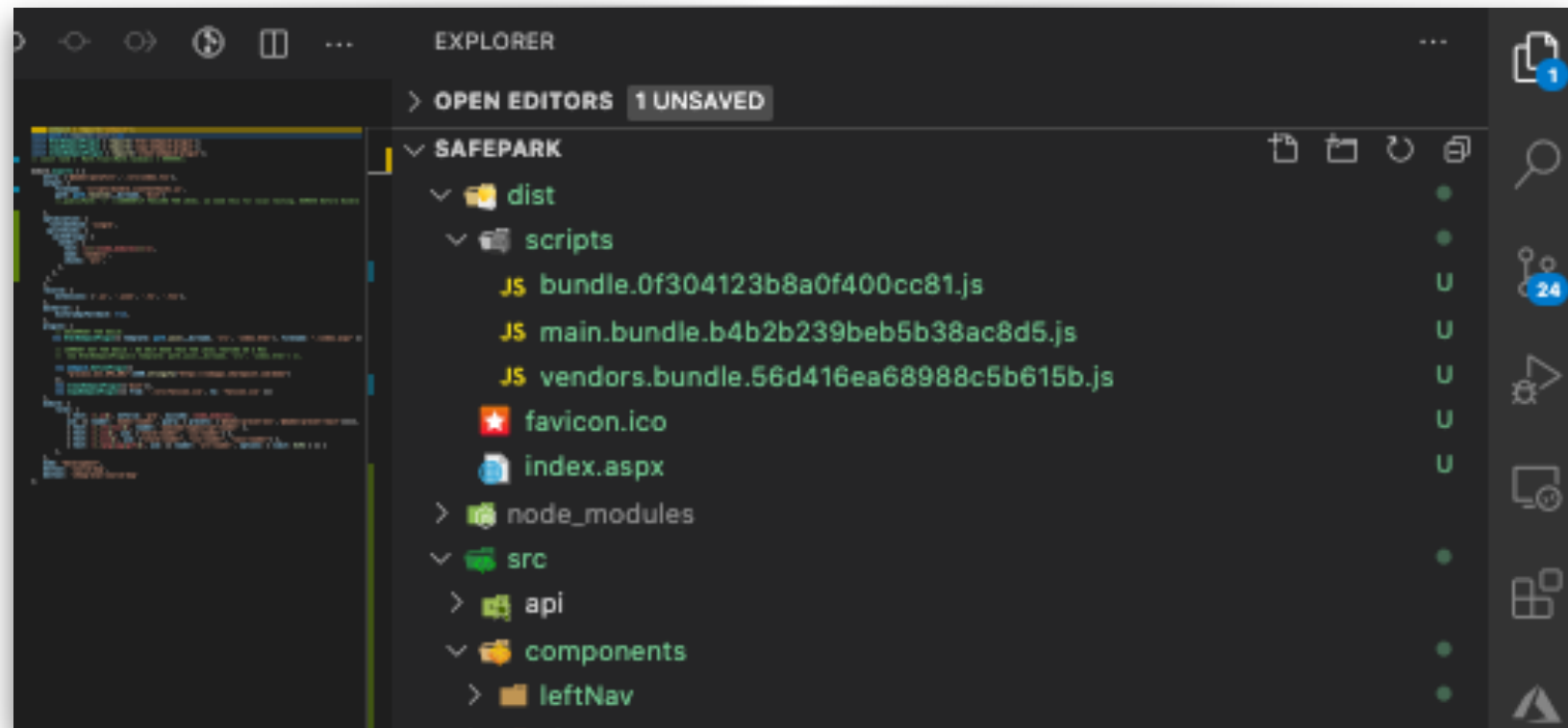
```
webpack.config.js — SafePark
webpack.config.js > <unknown> > optimization
You, a few seconds ago | 2 authors (Keith Craig and others)
1 const webpack = require('webpack');
2 const path = require('path');
3 const HtmlWebpackPlugin = require('html-webpack-plugin');
4 const CopyWebpackPlugin = require('copy-webpack-plugin');
5 const CleanWebpackPlugin = require('clean-webpack-plugin');
6 // const rand = Math.floor(Math.random() * 999999);
7
8 module.exports = {
9   entry: ['@babel/polyfill', './src/index.tsx'],
10  output: {
11    filename: 'scripts/bundle.[contenthash].js',
12    path: path.resolve(__dirname, 'dist')
13    //, publicPath: '/' // CURRENTLY FAILING FOR LOCAL. we need this for local testing, REMOVE Before Bundle
14  },
15  optimization: {
16    runtimeChunk: 'single',
17    splitChunks: {
18      cacheGroups: {
19        vendor: {
20          test: /[\\/]node_modules[\\/]/,
21          name: 'vendors',
22          chunks: 'all',
23        },
24      },
25    },
26  },
27  resolve: {
28    extensions: ['.js', '.json', '.ts', '.tsx'],
29  },
30  devServer: {
31    historyApiFallback: true,
32  },
33  plugins: [
34    // UNCOMMENT FOR BUILD:
35    new HtmlWebpackPlugin({ template: path.join(__dirname, 'src', 'index.html'), filename: './index.aspx' }),
36
37    // COMMENT OUT FOR BUILD - WE ONLY NEED THIS FOR LOCAL TESTING ON A MAC
38    // new HtmlWebpackPlugin({ template: path.join(__dirname, 'src', 'index.html') }),
39
40    new webpack.DefinePlugin({
41      "process.env.API_URL": JSON.stringify("https://lobapps.sharepoint.com/demo")
42    }),
43    new CleanWebpackPlugin(['dist']),
44    new CopyWebpackPlugin([{ from: './src/favicon.ico', to: 'favicon.ico' }])
45  ],
46  module: {
47    rules: [
48      { test: /\.js$/, enforce: 'pre', exclude: /node_modules/,
49        use: [{ loader: 'babel-loader', query: { presets: ['@babel/preset-env', '@babel/preset-react'] } } ]},
50      { test: /\.ts|tsx$/, loader: 'awesome-typescript-loader' },
51      { test: /\.css$/, use: ['style-loader', 'css-loader'] },
52      { test: /\.scss$/, use: ['style-loader', 'css-loader', 'sass-loader'] },
53      { test: /\.(png|jpg|gif)$/, use: [{ loader: 'url-loader', options: { limit: 8192 } } ] }
54    ],
55  },
56  mode: 'development',
57  devtool: 'source-map',
58  devtool: 'cheap-eval-source-map'
59 };
60
61
62
```


SharePoint Folder Hosted Apps - Bundling

SafePark

Notice in the screenshot on the previous page, I have included the webpack code to separate 3rd party vendor code.

```
,
  optimization: {
    runtimeChunk: 'single',
    splitChunks: {
      cacheGroups: {
        vendor: {
          test: /[\\/]node_modules[\\/]
        },
        name: 'vendors',
        chunks: 'all',
      },
    },
  },
},
```

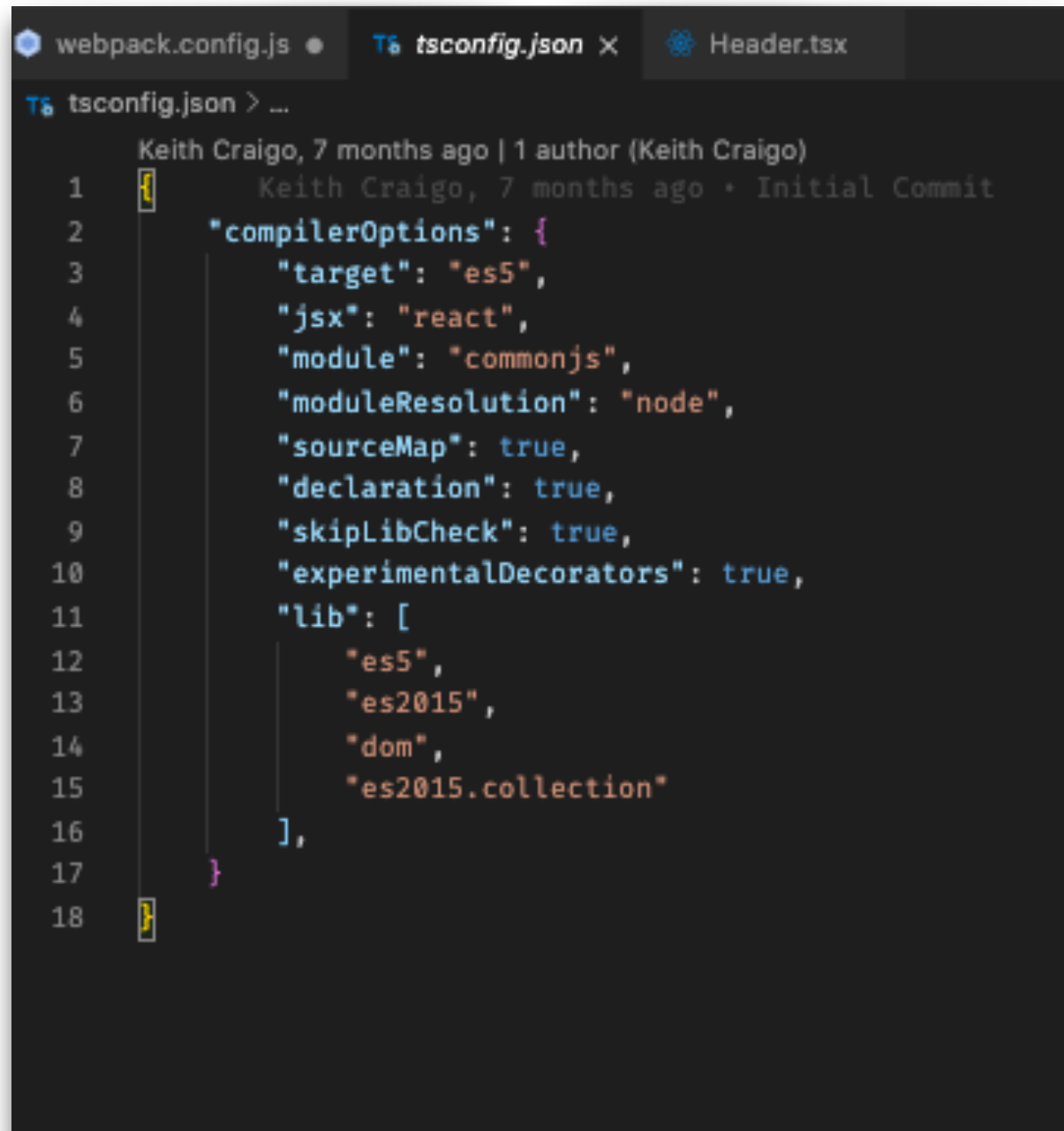


Notice the webpack build has put your code and the 3rd party vendor code into their own bundles. Now as long as you only make changes to your code, you will only need to upload the **main.bundle.[CHUNKHASH]** and **index.aspx** files to SharePoint after your build.

SharePoint Folder Hosted Apps - Config TypeScript

SafePark

4. In the root of the project let's create a **tsconfig.json** file to configure the TypeScript compiler.



The screenshot shows a code editor with three tabs: `webpack.config.js`, `tsconfig.json` (active), and `Header.tsx`. The `tsconfig.json` file is open, showing a JSON configuration for the TypeScript compiler. The configuration is as follows:

```
1 {
2   "compilerOptions": {
3     "target": "es5",
4     "jsx": "react",
5     "module": "commonjs",
6     "moduleResolution": "node",
7     "sourceMap": true,
8     "declaration": true,
9     "skipLibCheck": true,
10    "experimentalDecorators": true,
11    "lib": [
12      "es5",
13      "es2015",
14      "dom",
15      "es2015.collection"
16    ],
17  },
18 }
```

SharePoint Folder Hosted Apps - Project Files

SafePark

5. Next let's create a folder called **src** to be the home of all our source files.

Within src, let's create our containers.

We need the following files

- A. **index.txs**
- B. **index.html**
- C. **Header.tsx**
- D. **Footer.tsx**

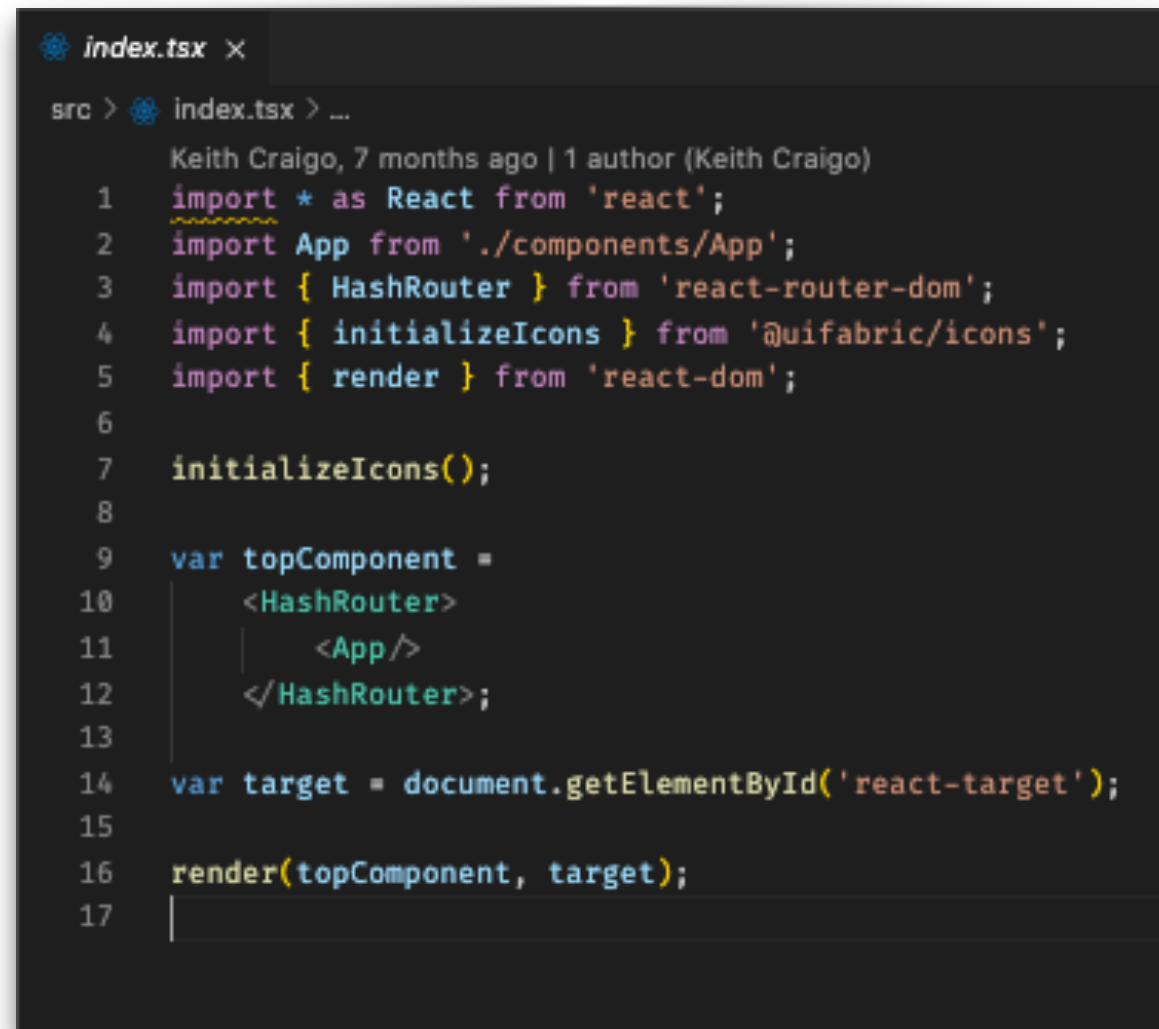
Let's start with **index.txs**

We use the **HashRouter** due to we won't be able to use the SharePoint Routing.

Even though this app lives in a SharePoint folder and surfaces information from SharePoint lists, it is not part of SharePoint, it lives on top of SharePoint.

We use the HashRouter due to SharePoint is not a web server so therefore we cannot use Server Side routing.

Please see a wonderful blog post titled Understanding The Fundamentals of Routing in React by Edmond Atto and John Kagga, <https://medium.com/the-andela-way/understanding-the-fundamentals-of-routing-in-react-b29f806b157e>



```
index.txs x
src > index.txs > ...
Keith CRAIGO, 7 months ago | 1 author (Keith CRAIGO)
1  import * as React from 'react';
2  import App from './components/App';
3  import { HashRouter } from 'react-router-dom';
4  import { initializeIcons } from '@uifabric/icons';
5  import { render } from 'react-dom';
6
7  initializeIcons();
8
9  var topComponent =
10    <HashRouter>
11      <App />
12    </HashRouter>;
13
14  var target = document.getElementById('react-target');
15
16  render(topComponent, target);
17
```

SharePoint Folder Hosted Apps - index.html

SafePark

6.index.html

Our main container for our React app

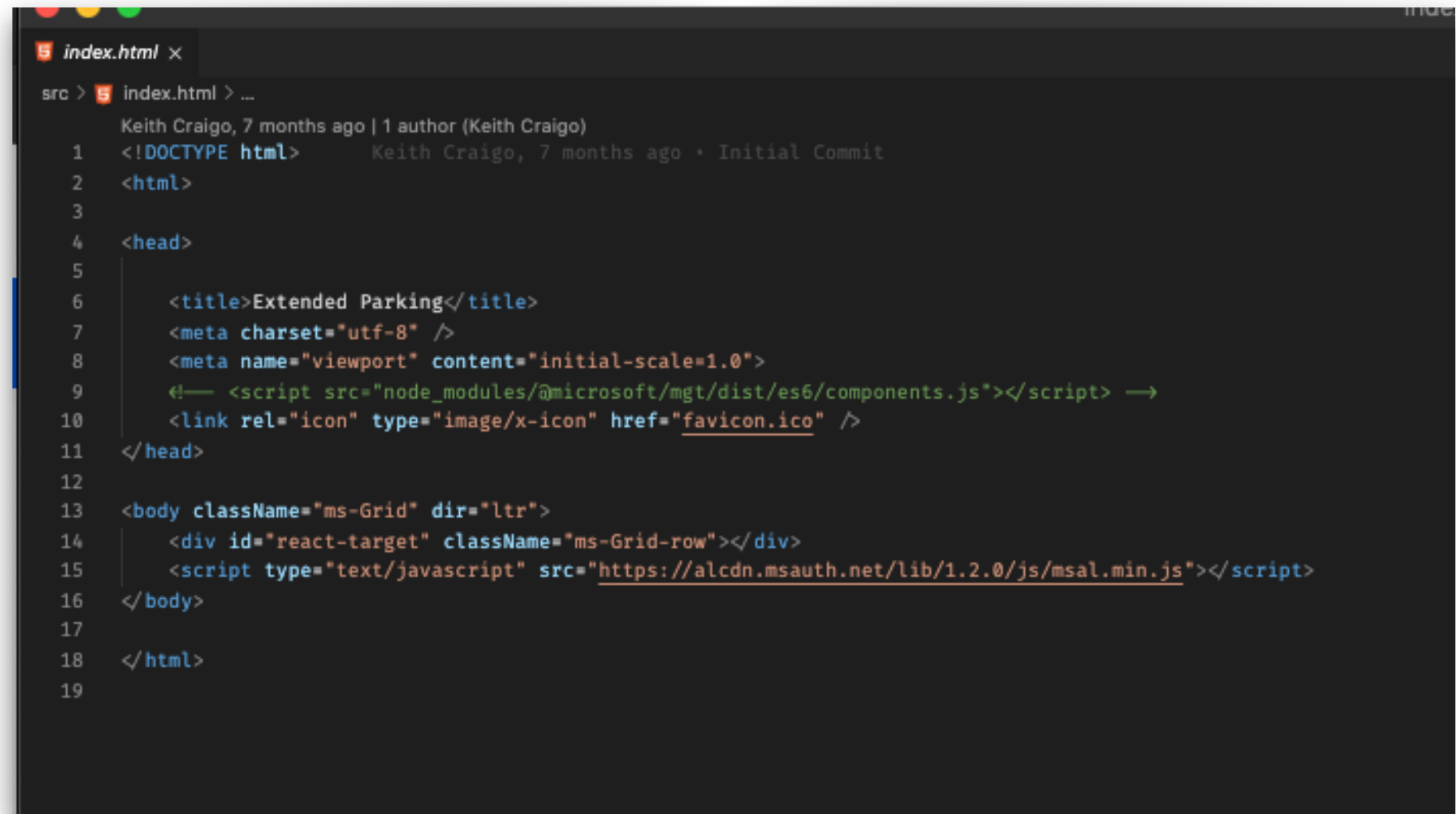
We specify where our React app should live in our index.html page

Our app will live inside the div with the id of **react-target**

Notice we also include a path to the

Microsoft Authentication Library (MSAL).

We need MSAL in order to authenticate users with Active Directory, if we are creating a SharePoint Hosted App or an app with the SharePoint Framework authentication would be taken care of by SharePoint.



```
index.html x
src > index.html > ...
Keith CRAIGO, 7 months ago | 1 author (Keith CRAIGO)
1 <!DOCTYPE html> Keith CRAIGO, 7 months ago • Initial Commit
2 <html>
3
4 <head>
5
6 <title>Extended Parking</title>
7 <meta charset="utf-8" />
8 <meta name="viewport" content="initial-scale=1.0">
9 <script src="node_modules/@microsoft/mgt/dist/es6/components.js"></script>
10 <link rel="icon" type="image/x-icon" href="favicon.ico" />
11 </head>
12
13 <body className="ms-Grid" dir="ltr">
14 <div id="react-target" className="ms-Grid-row"></div>
15 <script type="text/javascript" src="https://alcdn.msauth.net/lib/1.2.0/js/msal.min.js"></script>
16 </body>
17
18 </html>
19
```

SharePoint Folder Hosted Apps - Header.tsx

SafePark

7. Header.tsx

Notice the Hooks function **useEffect()** which combines the React class lifecycle methods. `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.

We are interested in the `componentDidMount` aspect. Once this component mounts we call a custom utility function I call **appUser()**.

But before we can call **appUser()** we need to create another config file for our connections. Then we need to create a **user** function that will use the settings in our **config.ts** to authenticate the user to SharePoint using the PNPJs library **currentUser** function. We authenticate the current user to get their email address **aka. userPrincipalName**, we will use `userPrincipalName` later on to call the Microsoft Graph to surface the current users Manager information in Azure Active Directory.

If not created already, create a folder named **src/utility**

In the utility folder create a new file named **config.ts**

Your code should look like the image on the next page.

```
Header.tsx x
src > Header.tsx > ...
You, a few seconds ago | 2 authors (Keith Craigo and others)
1 import * as React from "react";      Keith Craigo, 7 months ago · Initial Commit
2 import { LeftNavPanel } from "../components/leftNav/leftNav";
3 import { Switch, Route } from "react-router";
4
5 import Contacts from "../components/views/contacts/contacts";
6 import Home from "../components/views/home/home";
7 import MyDashboard from "../components/views/myDashboard/myDashboard";
8 import { Policies } from "../components/views/policies/policies";
9 import ParkingRequestForm from "../components/views/requestForm/requestForm";
10 import appUser from "../utility/user";
11 import { useEffect } from "react";
12
13 let currentUser = '';
14 export const Header = () => {
15
16     useEffect(() => {
17         appUser().then(res => {
18             currentUser = res;
19         });
20         return () => {
21             console.log("Header");
22         };
23     }, []);
24
25     return (
26         <div id="app-container" className="ms-Grid grid" dir="ltr">
27             <div className="Logo">
28                 <div id="banner-row">
29                     <div id="banner">
30                         <div id="appTitle">
31                             <a href="#">SAFE PARK!</a>
32                         </div>
33                         <div id="userInfo">
34                             <strong>{currentUser}</strong>
35                         </div>
36                     </div>
37                 </div>
38                 <div id="content-body-row">
39                     <div id="content-body" className="content-body ms-Grid">
40                         <LeftNavPanel />
41
42                         <Switch>
43                             <Route exact={true} path="/" component={Home} />
44                             <Route path="/home" component={Home} />
45                             <Route path="/policies" component={Policies} />
46                             <Route path="/contacts" component={Contacts} />
47                             <Route path="/myDashboard" component={MyDashboard} />
48                             <Route path="/requestForm" component={ParkingRequestForm} />
49                         </Switch>
50                     </div>
51                 </div>
52             </div>
53         </div>
54     );
55 };
56
57
58
59
```

SharePoint Folder Hosted Apps - config.ts

SafePark

8. config.ts

```
export default function config()
{
    const baseUrl = '[YOUR SHAREPOINT URL]';

    // Get AppID from your Azure Portal - Active Directory - App Registrations - App Overview
    // Page
    const appID = '[APPID FROM AZURE ACTIVE DIRECTORY REGISTRATION STEP]';

    // Find in many locations, i.e. Branding - Publisher Domain
    const tenant = '[YOUR TENANT URL]';

    // Get Tenant ID from your Azure Portal - Active Directory - Properties OR App
    // Registrations - App Overview Page
    const tenantID = '[TENANTID OR DIRECTORYID FROM AZURE ACTIVE DIRECTORY REGISTRATION STEP]';

    const authority = 'https://login.microsoftonline.com/' + tenantID + '/oauth2/v2.0/authorize?';
    const responseType = 'code';
    const responseMode = 'query';
    const state = '12345';

    // Where should Azure Active Directory return the Access and Refresh Tokens
    const redirectUri = 'https://[YOUR TENANT].sharepoint.com/[YOUR FOLDER URL]/index.aspx';

    // Requested Permissions
    const scopes = [
        'user.read.all',
        'calendars.read'
    ];

    return {appID: appID, baseUrl: baseUrl, redirectUri: redirectUri, scopes: scopes, tenant:
    tenant, tenantID: tenantID, authority: authority, responseMode: responseMode, responseType:
    responseType, state: state};
}
```

SharePoint Folder Hosted Apps - config.ts cont.

SafePark

8. config.ts - cont.

The following settings deserve Special Attention!

```
const baseUrl = '[YOUR SHAREPOINT URL]';
// Get AppID from your Azure Portal - Active Directory - App Registrations - App Overview Page
const appID = '[APPID FROM AZURE ACTIVE DIRECTORY REGISTRATION STEP]';

// Find in many locations, i.e. Branding - Publisher Domain
const tenant = '[YOUR TENANT URL]';

// Get Tenant ID from your Azure Portal - Active Directory - Properties OR App Registrations - App Overview
Page
const tenantID = '[TENANTID OR DIRECTORYID FROM AZURE ACTIVE DIRECTORY REGISTRATION STEP]';
const authority = 'https://login.microsoftonline.com/'+tenantID+'/oauth2/v2.0/authorize?';
```

And

```
/ Where should Azure Active Directory return the Access and Refresh Tokens
const redirectUri = 'https://[YOUR TENANT].sharepoint.com/[YOUR FOLDER URL]/index.aspx';

// Requested Permissions
const scopes = [
    'user.read.all',
    'calendars.read'
];
```

SharePoint Folder Hosted Apps - user.tsx

SafePark

9. user.tsx

In the utility folder create a new file called **user.tsx**

appUser() - This is an asynchronous function due to we do not know when the information will be returned.

Use the settings in our **config.ts** to authenticate the user using the **PNPJs** library **currentUser** function.

We return the users **userPrincipalName**, which is usually their email address, from their **SharePoint Profile** Account.

```
TS config.ts ● user.tsx ×
src > utility > user.tsx > ...

Keith Craig, 7 months ago | 1 author (Keith Craig)
1  import * as React from 'react';
2  import { useState } from 'react';
3
Keith Craig, 7 months ago · Initial Commit
4  import { Web } from "@pnp/sp/presets/all";
5  import config from "./config";
6
7  export default async function appUser()
8  {
9      const web = Web(config().baseUrl);
10     try {
11         const cUser = await web.currentUser();
12         console.log("UserPrincipalName: " + cUser.UserPrincipalName);
13         return cUser.UserPrincipalName;
14     } catch (error) {return error}
15
16 }
17
```


SharePoint Folder Hosted Apps - graphService.ts

SafePark

10. graphService.ts

In the utility folder create a new file called **graphService.ts**

graphService() - Use the settings in our **config.ts** to authenticate the user using the **PNPJs** library **currentUser** function.

We return the users details from their **Azure Active Directory Account**.

```
graphService.ts — SafePark
TS config.ts  TS user.tsx  TS graphService.ts x
src > utility > TS graphService.ts > ...
Keith Craig, 7 months ago | 1 author (Keith Craig)
1 // import "isomorphic-fetch"; Keith Craig, 7 months ago - Initial Commit
2 import { Client } from "@microsoft/microsoft-graph-client";
3 // import * as Msal from "msal";
4
5 import { UserAgentApplication } from "msal";
6
7 import { ImplicitMSALAuthenticationProvider } from "../../node_modules/@microsoft/microsoft-graph-client/lib/src/ImplicitMSALAuthenticationProvider";
8
9 import { MSALAuthenticationProviderOptions } from "../../node_modules/@microsoft/microsoft-graph-client/lib/src/MSALAuthenticationProviderOptions";
10
11 import config from "../config";
12
13 // An Optional options for initializing the MSAL @see https://github.com/AzureAD/microsoft-authentication-library-for-js/wiki/MSAL-basics#configuration-options
14 const msalConfig = {
15   auth: {
16     clientId: config().appID, // Client Id of the registered application
17     redirectUri: config().redirectUri
18   }
19 };
20 const graphScopes = config().scopes; // An array of graph scopes
21
22 // Important Note: This library implements loginPopup and acquireTokenPopup flow, remember this while initializing the msal
23 // Initialize the MSAL @see https://github.com/AzureAD/microsoft-authentication-library-for-js#1-instantiate-the-useragentapplication
24 const msalApplication = new UserAgentApplication(msalConfig);
25 const options = new MSALAuthenticationProviderOptions(graphScopes);
26 const authProvider = new ImplicitMSALAuthenticationProvider(
27   msalApplication,
28   options
29 );
30 const authOptions = {
31   authProvider // An instance created from previous step
32 };
33
34 const client = Client.initWithMiddleware(authOptions);
35
36 export async function graphAuthService(cEmp) {
37   try {
38     const userDetails = await client.api("/users/" + cEmp).get();
39
40     return userDetails;
41   } catch (error) {
42     throw error;
43   }
44 }
45
46 export async function getManager(cEmp) {
47   try {
48     const managerDetails = await client
49       .api("/users/" + cEmp + "/manager")
50       .get();
51
52     console.log("GRPHSVC-MANAGER DETAILS: " + managerDetails.displayName);
53
54     return managerDetails;
55   } catch (error) {
56     throw error;
57   }
58 }
59
```


SharePoint Folder Hosted Apps - Why user & graphService

SafePark

Why do we need both the **user.tsx** and **graphService.ts**, don't they return the same information?

Answer is Yes and No.

The **appUser** function in **user.tsx** returns a limited set of information from the workers SharePoint profile, their SharePoint profile is created from their Active Directory Account, but not all workers will have a SharePoint profile created for them. It depends on your organizations policies. Some workers job functions may not require they have network access so therefore it does not make sense to create resources for those workers who will never utilize the resource.

But you may still need to be able surface or submit the information for all workers, in the case where a SharePoint profile is not available, you still need to provide a way to search a workers information.

That's one of the reasons for **graphService.ts**.

We use the **Microsoft Graph** to surface the workers **Azure Active Directory** information.

SharePoint Folder Hosted Apps - Models

SafePark

11. Models

Let's build our interfaces now!

Create a new folder named **models** in our **src** folder.

In the models folder we need to create 3 interfaces.

1. **IBuilding.ts**
2. **ICampus.ts**
3. **ILocation.ts**

```
TS IBuilding.ts x
src > models > TS IBuilding.ts > ...
Keith Craig, 7 months ago | 1 author (Keith Craig)
1 export default interface IBuilding{
2     Id: number;
3     Title: string;
4 }
5
```

```
TS ICampus.ts x
src > models > TS ICampus.ts > ICampus
Keith Craig, 7 months ago | 1 author (Keith Craig)
1 export default interface ICampus{
2     Id: number;
3     Title: string;
4 }
5
```

```
$ ILocation.ts x
src > models > TS ILocation.ts > ILocation
Keith Craig, 7 months ago | 1 author (Keith Craig)
1 export default interface ILocation {
2     Id: number;
3     Building: string;
4     DirectionFromBuilding: string;
5     ParkingStructure: boolean;
6     PoleNumber: string;
7     NearestLandmark: string;
8     DirectionFromLandmark: string;
9 }
```

SharePoint Folder Hosted Apps - Components

SafePark

12. Components

Let's build some components!

Create a new folder named **components** in your **src** folder.

In the components folder we need to create **App.tsx**

App.tsx is where we set the Theme of the app, setup any variables or components of app to be made Globally available with **export const ConfigContext = React.createContext(null);**

<ConfigContext.Provider value={ ... }>

More information about createContext here:

<https://reactjs.org/docs/context.html>

Next create an **App.scss** and style to your liking or use the styles in <https://github.com/kcraigo/SafePark/blob/master/src/components/App.scss>

```
App.tsx x
src > components > App.tsx > ...
You, a few seconds ago | 2 authors (Keith Craigo and others)
1
2 import * as React from 'react';
3 import { loadTheme } from 'office-ui-fabric-react'; Keith Craigo, 7 months
4 import { withRouter } from 'react-router';
5 import '../node_modules/office-ui-fabric-react/dist/css/fabric.min.css';
6 import './App.scss';
7 import { Header } from '../header';
8 import { Footer } from '../footer';
9
10
11 const configValue = {
12   showWelcome: true
13 }
14
15
16 loadTheme({
17   palette: {
18     themePrimary: '#4eadd9',
19     themeLighterAlt: '#f4fafa',
20     themeLighter: '#d4eae8',
21     themeLight: '#b2d8d9',
22     themeTertiary: '#72b1b3',
23     themeSecondary: '#438e8f',
24     themeDarkAlt: '#2e7273',
25     themeDark: '#276061',
26     themeDarker: '#1d4747',
27     neutralLighterAlt: '#f8f8f8',
28     neutralLighter: '#f4f4f4',
29     neutralLight: '#eaeaea',
30     neutralQuaternaryAlt: '#dadada',
31     neutralQuaternary: '#d0d0d0',
32     neutralTertiaryAlt: '#c8c8c8',
33     neutralTertiary: '#bab8b7',
34     neutralSecondary: '#a3a2a0',
35     neutralPrimaryAlt: '#8d8b8a',
36     neutralPrimary: '#323130',
37     neutralDark: '#605e5d',
38     black: '#494847',
39     white: '#ffffff',
40   }
41 });
42
43
44 export const ConfigContext = React.createContext(null);
45
46 Keith Craigo, 7 months ago | 1 author (Keith Craigo)
47 class App extends React.Component<any, any> {
48
49   render(): JSX.Element {
50     return (
51       <ConfigContext.Provider value={configValue}>
52         <Header />
53         <Footer />
54       </ConfigContext.Provider>
55     );
56   }
57
58 }
59
60
61 export default withRouter(App);
62
```

SharePoint Folder Hosted Apps - Components cont.

SafePark

12. Components - cont

Let's build some components!

Create an new folder in src/components called **requestForm**

In requestForm create a new file named **requestForm.tsx**

requestForm.tsx code is too large to copy and paste to this book, instead I will refer you to the code in my GitHub. I'll focus on the main points in this book.

Starting with setting up the SharePoint context, **sp.setup({**

Not only do we need to specify what type of data we will accept from SharePoint we need to setup the baseUrl for our apps routing. **What?**

If we don't setup the baseUrl here then the browser will default to your apps baseUrl, the folder location in SharePoint where the app lives and you will always get a 404 error message that a list cannot be found.

SharePoint resources must be referenced from the baseUrl of your SharePoint tenant, not the baseUrl of your app.

// other imports omitted for brevity

```
import { sp } from "@pnp/sp/presets/all";
```

// other code omitted for brevity

```
export const ParkingRequestForm = () => {  
  sp.setup({  
    sp: {  
      headers: {  
        Accept: "application/json;odata=verbose"  
      },  
      baseUrl: config().baseUrl  
    }  
  });  
};
```

// other code omitted for brevity

SharePoint Folder Hosted Apps - requestForm.tsx

SafePark

12. Components - cont

Let's build some components!

If you have not already, create a new folder in src/components called **requestForm**

In requestForm create a new file named **requestForm.tsx**

requestForm.tsx code is too large to copy and paste to this book, instead I will refer you to the code in my GitHub. I'll focus on the main points in this book.

Returning SharePoint list data to Fluent UI controls!

i.e the Fluent UI DropDown control

This control expects the data to be in the form of [Key, Text] pairs but in the SafePark example the data from SharePoint is returned as [Id, Title] we need to use a Reducer to change the shape of the data to the [Key,Text]. Once the data from SharePoint is returned we send or dispatch the type and data as a message.

The Reducer is setup as a listener to dispatch

```
const [state, dispatch] = useReducer(reducer, []);
```

```
// CAMPUSES
async function _getCampuses(): Promise<void> {
  await sp.web.lists
    .getByTitle("Campus")
    .items.select("Id", "Title")
    .getAll()
    .then(response => {
      const cmps = response.map(desc => {
        return {
          ...desc
        };
      });
    });
}
```

```
dispatch({
  type: "setCampuses",
  data: cmps
});
});
}
```

// Other code omitted for brevity ...

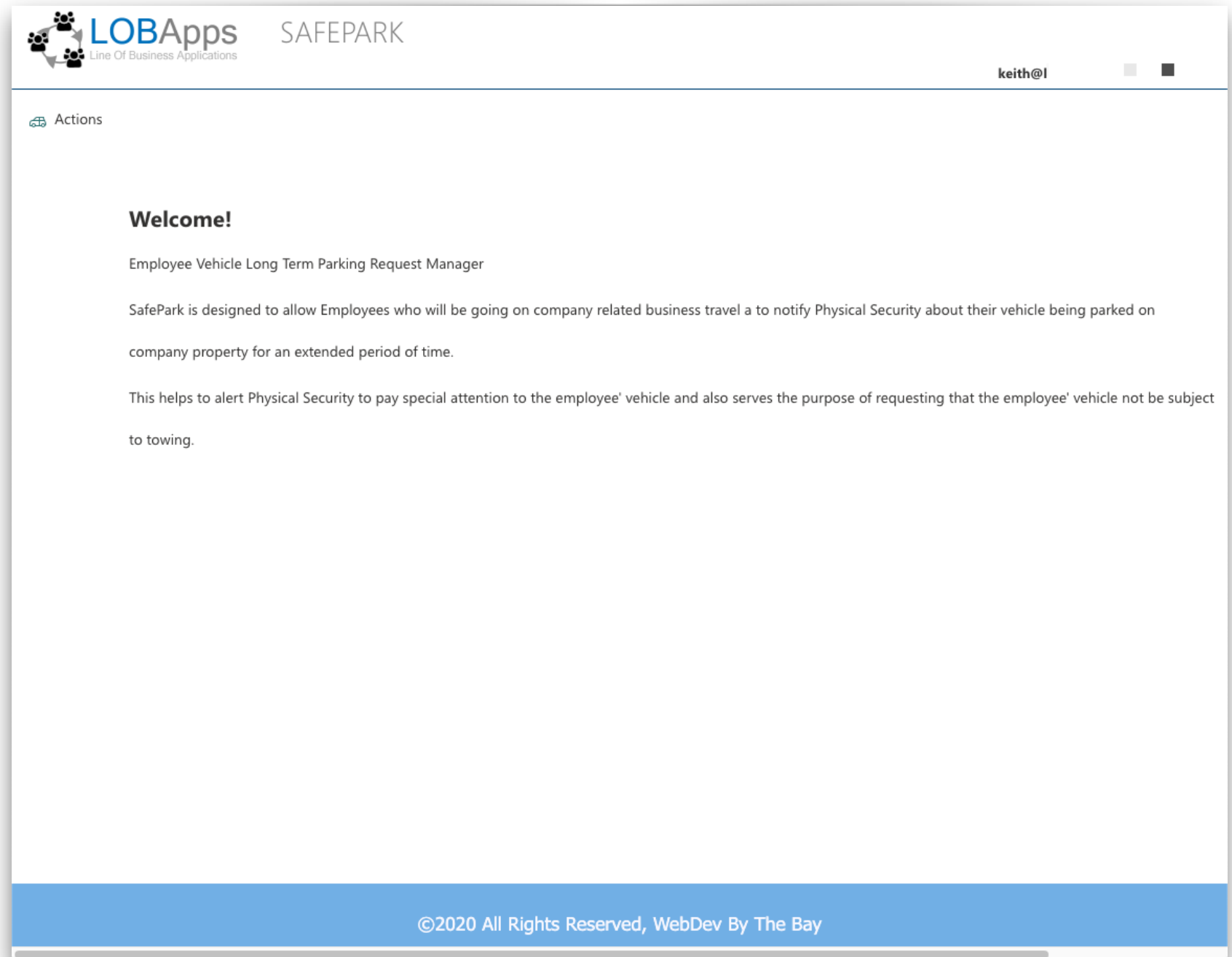
```
function reducer(state: any, action: { type: any; data: any[] }) {
  switch (action.type) {
    case "setCampuses": {
      // We need to rename the keys: Id to key and Title to text
      return {
        campuses: action.data.map(function(cm: { [x: string]: any }) {
          cm["key"] = cm["Id"];
          delete cm["Id"];
          cm["text"] = cm["Title"];
          delete cm["Title"];
          return cm;
        });
      };
    }
  }
}
```

SharePoint Folder Hosted Apps - UI

SafePark

For the rest of this tutorial I will refer you to take a look at the code in my github repository for this project:
<https://github.com/kcraigo/SafePark>

Let's now focus on the User Experience!



SharePoint Folder Hosted Apps - Navigation

SafePark

User Experience!

leftNav.tsx

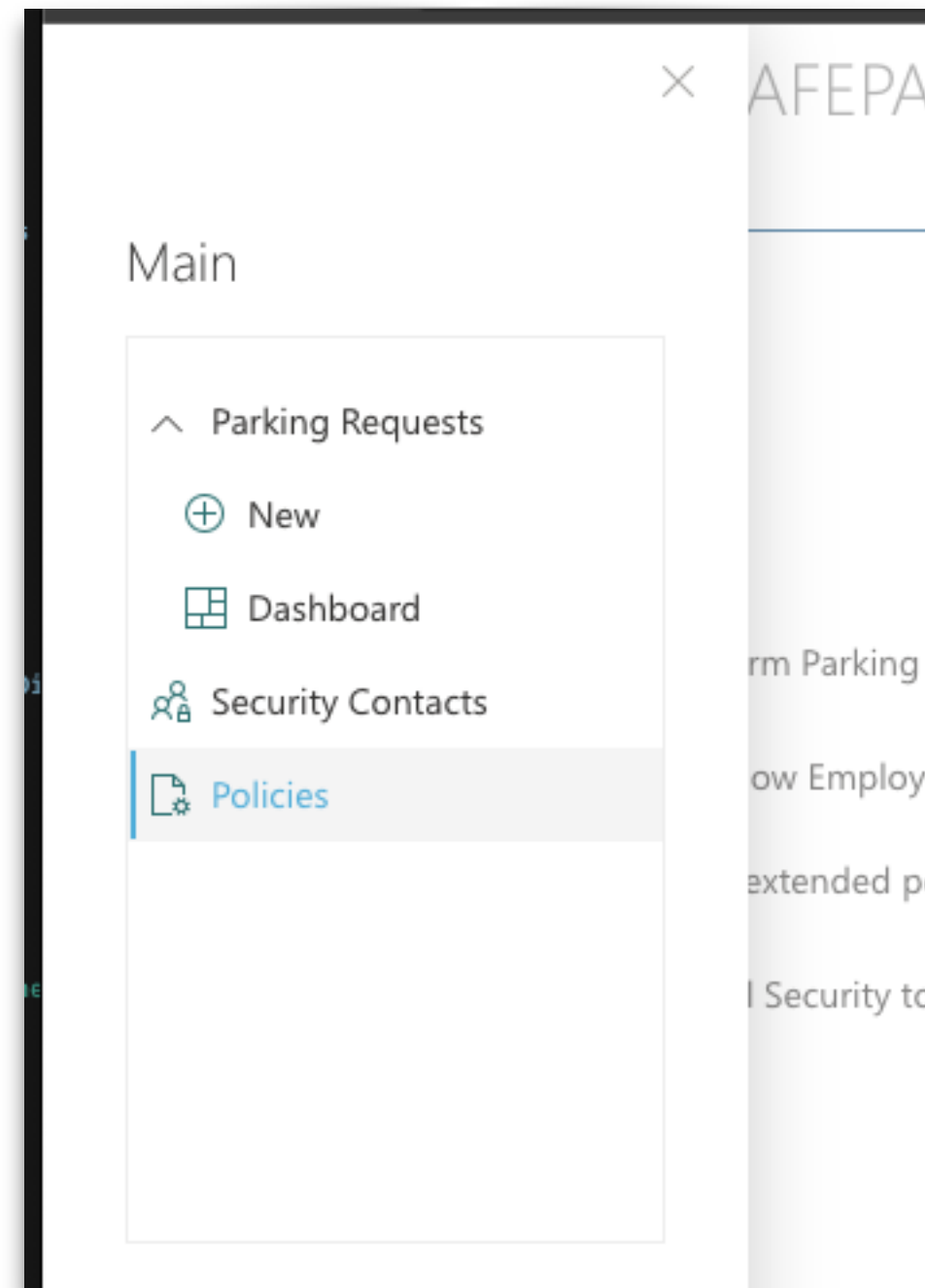
Side Panel Menu - Fluent UI customNear PanelType

If this is the first time the user clicks the side panel **Parking Requests - New** button, they will be asked to login with their organizational credentials.

Their credentials will then be checked against Azure Active Directory using the **graphService.ts** and **config.ts** settings we created earlier. If authenticated then they will be authorized to proceed.

If the user has already been authenticated or they have been authenticated with the organizations O365 tenant, then they will be authorized to proceed.

Once authenticated and authorized the users information will then be retrieved from their Active Directory account and surfaced in the forms Initiator Information fields.



SharePoint Folder Hosted Apps - Search

SafePark

User Experience!

When the Search button is clicked we pass the Employee's first name to graphService.ts - **_searchUser** which appends the organizational email domain onto the users first name.

I kept this example simple but this can be a more complicated schema, you may need to account for a last name, if the employee is a contractor or an RFT(Regular Full Time) or however your organization has email addresses configured.

Employee Information				
Employee First Name	Employee:	Title	Phone	EmployeeID
<input type="text" value="barney"/>	Barney Rubble	Officer		<input type="text"/>
<input type="button" value="SEARCH"/>				

Manager Information			
Name:	Email:	Title:	Phone:
Keith Craigo	keith@[REDACTED]	CEO and Founder	[REDACTED]

Emergency Contact Information	
-------------------------------	--

SharePoint Folder Hosted Apps - Surface SharePoint data

SafePark

User Experience!

Campuses and Buildings are surfaced from their respective SharePoint lists using the PnPJs library SharePoint functions.

The screenshot shows a web application interface for submitting a parking request. The form is organized into several sections with labels above the input fields:

- Contact Information:** Includes fields for Contact Name, Phone (with a format guide () -), and Email.
- Dates:** Includes Start Date and Return Date, each with a calendar icon.
- Vehicle Information:** A section header followed by fields for Make, Model, Year, Color, and License Plate.
- Description:** A large text area for "Distinguishing features and or marks to assist Security personnel in identifying your vehicle."
- Parking Information:** A section header followed by dropdown menus for Campus (labeled "Select a Campus"), Building (labeled "Select a Building"), and Level (labeled "Level").
- Justification:** A large text area for providing a justification for the request.

At the bottom right of the form is a blue button labeled "Submit Parking Request". The footer of the application contains the text "©2020 All Rights Reserved, WebDev By The Bay".

SharePoint Folder Hosted Apps - Stay Tuned!

SafePark

There is so much more!

There are many ways to accomplish the same thing.

I personally will use one of the other Approaches before developing a SharePoint Folder Hosted app mostly due to compliance with organization policies.

This approach could be considered a side loaded application and could if not careful introduce potential Security concerns.

If you have stuck with me this far, I thank you for your time.

Please take a look at my blog at <https://webdevbythebay.com> for more tutorials like this one in the near future.

That you,

Keith Craigo



WEBDEV BAY
BY THE