

Arlington County Code Challenge

By Teju Peri

E-mail: teju.fullstackdeveloper@gmail.com

Contents

Key Aspects: SharePoint Framework, JavaScript/TypeScript Integration using React. Code clarity, Modularity, Limiting any Technical Debt	1
What is Office UI Fabric React components in your SharePoint client-side webpart	1
React in SPFx (SharePoint Framework) projects	2
Project Structure	3
My Code solution (CodeTestPart.tsx)	3
Code Explanation	5
My Code solution (SharePointListReactCRUD.tsx)	5
Code Explanation	9
Conclusion, further analysis, and alternative approaches.....	10

Key Aspects: SharePoint Framework, JavaScript/TypeScript Integration using React. Code clarity, Modularity, Limiting any Technical Debt

I have used GitHub, VSCode, SharePoint Framework (SPFx) using react js, PnP js and PnP SPFx react controls, Node.js, npm and tried to use several components interactively.

What is Office UI Fabric React components in your SharePoint client-side webpart

Office UI Fabric React is the front-end framework for building experiences for Office and Office 365. It includes a robust collection of responsive, mobile-first components that make it easy for us to create web experiences by using the Office Design Language.

Web parts can be built like in the image below that uses the







DocumentCard component of Office UI Fabric React.

Image Credit: JustinMind

React in SPFx (SharePoint Framework) projects

1. Project setup and structure
2. Referencing a React component in our SPFx solution
3. Creating a React class component
4. Taking advantage of the already available controls
5. React fundamentals like Props & state
6. Lifecycle methods
7. Event handlers
8. Splitting components
9. Passing state between components
10. The render prop technique
11. Redux
12. React hooks
13. Copying props to state
14. Using unsafe lifecycle methods

Project Structure

 CodeTestPart.module.scss	10/5/2021 3:29 PM	SCSS Style She...	2 KB
 CodeTestPart.tsx	10/6/2021 7:17 PM	TypeScript JSX...	5 KB
 ICodeTestPartProps.ts	10/5/2021 3:29 PM	TS File	1 KB
 SharePointListReactCRUD.tsx	10/6/2021 7:17 PM	TypeScript JSX...	6 KB

My Code solution (CodeTestPart.tsx)

```
import * as React from 'react';
import styles from './CodeTestPart.module.scss';
import { ICodeTestPartProps } from './ICodeTestPartProps';
import { escape } from '@microsoft/sp-lodash-subset';
import { DetailsList, DetailsListLayoutMode, Selection, IColumn } from 'office-ui-fabric-react/lib/DetailsList';
```

// Modified by Teju Peri - *** 10-6-21 *** Arlington County - MS Solution Architect Role

/*For simplicity sake I am just changing per the requirement.
Please see my solution/readme for a more extensive solution/extension for this*/

```
export interface IDetailsListBasicExampleItem {
  key: number;
  name: string;
  status: string;
}

export interface IDetailsListBasicExampleState {
  items: IDetailsListBasicExampleItem[];
  selectionDetails: string; /* Added selection Details. We can press enter/focus on
item row etc.and show alerts
of which Item form the ToDo list has been selected*/
}
```

```
export default class CodeTestPart extends React.Component<ICodeTestPartProps,
IDetailsListBasicExampleState> {
  private _selection: Selection;
  private _allItems: IDetailsListBasicExampleItem[];
  private _columns: IColumn[];

  /* I have alos tried to show how to use the Property '_selection',
  gave it an initializer and assigned it in the constructor.*/

  constructor(props: ICodeTestPartProps) {
    super(props);
    //Here I am filling with 3 dummy values in the Task List
    this._allItems = [];
```

```

    for (let i = 0; i < 3; i++) {
        this._allItems.push({
            key: i,
            name: 'ItemNumber ' + i,
            status: 'Pending',
        });
    }

    this._selection = new Selection({
        onSelectionChanged: () => this.setState({ selectionDetails:
this.getSelectionDetails() }),
    });

    this._columns = [
        { key: 'column1', name: 'Task Name', fieldName: 'name', minWidth: 100, maxWidth:
200, isResizable: true },
        { key: 'column2', name: 'Status', fieldName: 'status', minWidth: 100, maxWidth:
200, isResizable: true },
    ];

    this.state = {
        items: this._allItems,
        selectionDetails: this.getSelectionDetails(),
    };
}
public render(): React.ReactElement<ICodeTestPartProps> {
    const { items, selectionDetails } = this.state;
    /* In the code below we can use focus on a item row, click etc. and excute a
function like
    "onItemInvoked" to show an alert etc. We can use an Announced/Text to show the
alert
    */
    return (
        <div className={styles.codeTestPart}>
            <div className={styles.container}>
                <div className={styles.row}>
                    <div className={styles.column}>
                        <span className={styles.title}>Welcome to SharePoint!</span>
                        <p className={styles.subTitle}>Customize SharePoint experiences using Web
Parts.</p>
                        <p className={styles.description}>{escape(this.props.description)}</p>
                        <DetailsList
                            items={items}
                            columns={this._columns}
                            setKey="set"
                            layoutMode={DetailsListLayoutMode.justified}
                            selection={this._selection}
                        />
                    </div>
                </div>
            </div>
        </div>
    );
}

//Some kind of functions here to add the selection Details Logic to our app.

```

```

private getSelectionDetails(): string {
    const selectionCount = this._selection.getSelectedCount();

    switch (selectionCount) {
        case 0:
            return 'No task items have been selected';
        case 1:
            return 'One task item selected: ' + (this._selection.getSelection()[0] as
IDetailsListBasicExampleItem).name;
        default:
            return `${selectionCount} items selected`;
    }
}

private onItemInvoked = (item: IDetailsListBasicExampleItem): void => {
    alert(`An Task Item has ben selected: ${item.name}`);
};
}

/*
Adding an item to a hypothetical SharePoint Online/0365 list "To Do" in the SharePoint
framework using reactjs .
*/
private async SaveDataSPList() {
    let web = Web(this.props.webURL);
    await web.lists.getByTitle("ToDoList").items.add({

        name: this.state.name,
        status: this.state.status,

    }).then(i => {
        console.log(i);
    });
    alert("Created a new item in the To DO List Successfully");
    this.setState({ name: "", status: "" });
    this.fetchData();
}

//Please Check SharePointListReactCRUD.tsx for the complete code

```

Code Explanation

I have simply looped over the `_allItems` array and inserted 3 dummy items into the “To Do” Task List setting the name, status values etc.

My Code solution (SharePointListReactCRUD.tsx)

```

// Created by Teju Peri - *** 10-6-21 ***  Arlington County - MS Solution Architect Role

/* Not the complete example or the app/codebase for lack of time
But to give an idea of CRUD using SharePoint List/React
Dummy Code- Full Syntax check not done*/

```

```

import * as React from 'react';
import styles from './TejuSPReactCRUD.module.scss';
import { ITejuSPReactCRUDProps } from './ITejuSPReactCRUDProps';
import { escape } from '@microsoft/sp-lodash-subset';

import { TextField } from 'office-ui-fabric-react/lib/TextField';
import { Label } from 'office-ui-fabric-react/lib/Label';
import { sp, Web, IWeb } from "@pnp/sp/presets/all";
//If we want to use some Date fields or say leverage SharePoint PeoplePicker
import { DatePicker, IDatePickerStrings } from 'office-ui-fabric-react/lib/DatePicker';
import { PeoplePicker, PrincipalType } from "@pnp/spfx-controls-react/lib/PeoplePicker";
import "@pnp/sp/lists";
import "@pnp/sp/items";
import { PrimaryButton } from 'office-ui-fabric-react/lib/Button';

import { WebPartContext } from "@microsoft/sp-webpart-base";

```

```

// This will be in the File ITejuSPReactCRUDProps.ts

```

```

export interface ITejuSPReactCRUD {
  description: string;
  context: WebPartContext;
  webURL: string;
}

```

```

/* This will be in the File TejuSPReactCRUDWebPart.ts file and import the component
SharePointListReactCRUD.tsx in this file */

```

```

public render(): void {
  const element: React.ReactElement < ITejuSPReactCRUDProps > = React.createElement(
    CRUDReact,
    {
      description: this.properties.description,
      webURL: this.context.pageContext.web.absoluteUrl,
      context: this.context
    }
  );
  ReactDOM.render(element, this.domElement);
}

```

```

//Interface Declaration
export interface IStates {
  Items: any;
  ID: any;
  name: any;
  status: any;
  HTML: any;
}

```

```

export default class SharePointListReactCRUD extends
React.Component<ITejuSPReactCRUDProps, IStates> {
  //Everything empty set in the constructor
  constructor(props) {
    super(props);
    this.state = {
      Items: [],
      name: "",
      ID: 0,
      status: "",
      HTML: []
    };
  }

  //Get Items from SP List using the componentDidMount() method
  public async componentDidMount() {
    await this.fetchData();
  }

  public async fetchData() {
    let web = Web(this.props.webURL);
    const items: any[] = await web.lists.getByTitle("ToDoList").items.select("*,
"name").get();
    console.log(items);
    this.setState({ Items: items });
    let html = await this.getHTML(items);
    this.setState({ HTML: html });
  }

  public async getHTML(items) {
    var tabledata = <table className={styles.table}>
      <thead>
        <tr>
          <th>Task name</th>
          <th>Task Status</th>
        </tr>
      </thead>
      <tbody>
        {items && items.map((item, i) => {
          return [
            <tr key={i} onClick={() => this.findData(item.ID)}>
              <td>{item.name}</td>
              <td>{item.status}</td>
            </tr>
          ];
        })}
      </tbody>
    </table>;
    return await tabledata;
  }
}

```

```

    }
    /*

```

Adding an item to a hypothetical SharePoint Online/0365 list "To Do" in the SharePoint framework using reactjs .

using PnP once the item gets created, we can call a setState() to make the form fields empty.

fetchData() method is then called, so that it will reload the data and we can see the updated items.*/*

```

    private async SaveDataSPList() {
        let web = Web(this.props.webURL);
        await web.lists.getByTitle("ToDoList").items.add({

            name: this.state.name,
            status: this.state.status,

        }).then(i => {
            console.log(i);
        });
        alert("Created a new item in the To DO List Successfully");
        this.setState({ name: "", status: "" });
        this.fetchData();
    }
    //setState()
    //fetchData()
    private async DeleteDataSPList() {

    }
    private async UpdateDataSPList() {

    }

    public render(): React.ReactElement<ITejuSPReactCRUDProps> {
        return (
            <div >
                <h1>ReactJs - CRUD Operations </h1>
                {this.state.HTML}
                <div className={styles.btngroup}>
                    <div><PrimaryButton text="Create" onClick={() =>
this.SaveDataSPList()} /></div>
                    <div><PrimaryButton text="Update" onClick={() =>
this.UpdateDataSPList()} /></div>
                    <div><PrimaryButton text="Delete" onClick={() =>
this.DeleteDataSPList()} /></div>
                </div>
                <div>
                    <form>
                        <div>
                            // We can use SharePoint PeoplePicker as well to get user
name properties etc.
                            <Label>SharePoint User Name</Label>
                            <PeoplePicker
                                />
                        </div>
                        <div>
                            <Label> To Do List Task Name</Label>

```



```

        <TextField value={this.state.name} multiline
onChange={value => this.onChange(value, "name")} />
      </div>
    <div>
      <Label>Task Status</Label>
      <TextField value={this.state.status} multiline
onChange={value => this.onChange(value, "status")} />
    </div>
  </form>
</div>
</div >
);
}
}

```

Code Explanation

SharePoint framework CRUD operations using react

spfx crud operations using react – We are displaying items from SharePoint Online list “To Do”

sharepoint framework crud operations – We are inserting items from SharePoint Online list “To Do”

sharepoint framework crud operations – We are updating items from SharePoint Online list “To Do”

spfx crud operations using react pnp – We are deleting items from SharePoint Online list “To Do”

We are creating a SPFx client side web part using the React framework and will do the CRUD operations.

We can use Node.js, npm to install **PnP SPFx react controls**

Not the complete example or the app/codebase for lack of time

But to give an idea of CRUD using SharePoint List/React

It will be Dummy Code- Full Syntax check not done

We create/include `ITejuSPReactCRUDProps.ts` with interface code which we include in the same file for lack of time.

We have `TejuSPReactCRUDWebPart.ts` file and in this we import the component `SharePointListReactCRUD.tsx`

We use all required `Import` statements

Then we have the code to render the form using react controls. We can use Office ui fabric react controls and also PnP react controls for People picker etc., if needed.

Then we declare the interface

In constructor we set everything (empty)

In the react componentDidMount() method, there will be code to get the items from the SharePoint Online list.

We then add item to the SharePoint Online “To Do” list in the SharePoint framework using reactjs

We add the item using PnP and once the item gets created, we call the setState (), to make the form fields empty. The fetchData() method is called then, so that it will reload the data and we can see the updated items.

We then use “gulp serve” to use SharePoint workbench and test the SharePoint web part.

Conclusion, further analysis, and alternative approaches

It is clear that any strategy or solution should concentrate on efficient code, modularity

Our approach is simple for the lack of time. But I will discuss certain alternative strategies and further work we can do to dig deeper for generating actionable insights.

1. We can use Office ui fabric react controls and also PnP react controls for People picker etc.
2. We can leverage the following from Office UI React Fabric and extend our SharePoint webpart further
 - Use Galleries & Pickers
 - Work with SharePoint Items/Lists
 - Use Notifications & Engagement
 - Leverage commands, Menus & Navs

React Hooks

Use React Hooks with SharePoint Framework

3. Use various SPFx Extensions
4. Invoke Anonymous REST API's from SPFx Web Part
5. Leverage External Libraries from SPFx Web Part

6. Use MSGraph API using React Framework
7. Use other Office UI Fabric React Controls in SPFx Web Part
8. Develop Provider and Consumer Web Parts
9. Develop Single Page Applications (SPA)
10. Implement Localization Support for SPFx Web Part
11. Create Custom Tabs for Microsoft Teams
12. Deploy SPFx solutions to Office 365 CDN and Azure CDN
13. Develop Custom Gulp Tasks