Preparing your PC

Create Your First Project

1. Install an Editor: Visual Studio Code

```
https://code.visualstudio.com/
```

You'll want a great editor for your script and code. Any good editor will do – even Notepad – but Visual Studio Code provides a lot of good editing features along with a vibrant add-ons capability.

2. Install NodeJS

```
https://nodejs.org/
```

NodeJS provides a lot of functionality, including a whole server-side development framework. In our case, though, we're really only using NodeJS for the Node Package Manager (NPM).

3. Production Windows Build Tools

```
npm install --global --production windows-build-tools
```

4. Install yeoman and gulp

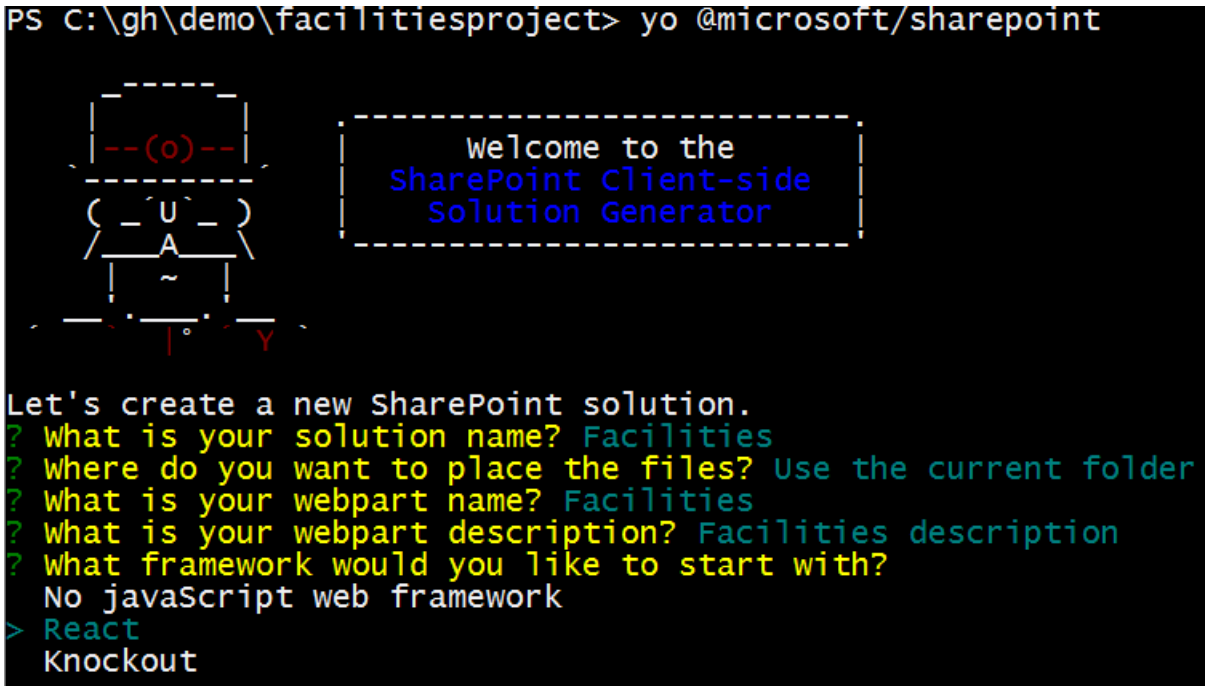
```
npm i -g yo gulp
```

5. Add the Yeoman SharePoint Generator

```
npm i -g @microsoft/generator-sharepoint
```

6. Create your first project

```
md facilitiesproject
cd facilitiesproject
yo @microsoft/sharepoint
```



7. Start Visual Studio Code

```
code .
```

Now we'll start Visual Studio Code, pointed at the root folder of your project.

8. Run your first project

```
gulp serve
```

This will run the Gulp in a continuous 'streaming' model

Start Retrieving Data

9. Add a State Object and Use it in Your Par

```
export interface IFacilitiesState
{
  items?: any[];
}
```

10 . Add a Data Retrieval Constructor

```
constructor(props: { description : string })
  {
    super(props);
    this.state = { items: new Array() };
    let self = this;
  fetch("https://spawesome.blob.core.windows.net/facilities/facilities.json",
           { "credentials": "omit" } )
      .then((response) => response.json())
      .then((responseData) => {
          self.setState( {
                    items: responseData,
                });
      });
  }
```

11. Add the following render markup

```
   return (
      <div className={styles.facilities}>
        <div>This is the <b>{this.props.description}</b> webpart.</div>
        <table>
          <tbody>
      {
```

```
          this.state.items.map(function(object, i)
          {
           return <tr><td><b>{object.name}</b></td><td>{object.status}</td></tr>;
          })
        }
            </tbody>
          </table>
        </div>
      );
```

Improve the Appearance

12. Run the following command from the prompt to add Office UI Fabric Libraries to your Project:

```
npm i office-ui-fabric-react --save
```

13. Add Imports for Office UI Fabric into Facilities.tsx, beneath the existing import statements (around line #7).

```
import
{
  DetailsList
} from 'office-ui-fabric-react';
```

14. Add the following render Function into Facilities.tsx:

```
public render(): JSX.Element {
    return (
      <div className={styles.facilities}>
        <div className="ms-font-su"> { this.props.description
}</div>

        <div className="ms-Grid">
          <div className="ms-Grid-row">
            <div className="ms-Grid-col ms-u-sm6 ms-u-md4 ms-u-lg6">
              <DetailsList    items={ this.state.items }
                              onItemInvoked={ (item, index) =>
this.setState( { selectedItem: item } ) }
                              onRenderItemColumn={ _renderItemColumn
}
                              columns={
                                [
                                  {
                                    key: "status",
                                    name: "Status",
                                    fieldName: "status",
                                    minWidth: 60
                                  },
                                  {
                                    key: "name",
                                    name: "Name",
                                    fieldName: "name",
                                    minWidth: 300
                                  }
                                ] } />
            </div>
          </div>
        </div>
      </div>
    );
  }

}
```

```
function _renderItemColumn(item, index, column)
{
  const fieldContent = item[column.fieldName];

  switch (column.key)
  {
    case 'status':
      return <div style={ { backgroundColor: fieldContent,
borderRadius: "16px", width: "16px", marginLeft: "6px" }
}> </div>;

    default:
      return <span>{ fieldContent }</span>;
  }
}
```

Add the following line to IFacilitiesState:

```
   selectedItem?: any;
```

Now, let’s add a new React Component

15. Add a new React component called “facility.tsx” to the “facilities” folder.

```
import * as React from 'react';
import {
  EnvironmentType
} from '@microsoft/sp-client-base';
import styles from '../Facilities.module.scss';
import {
  IWebPartContext
} from '@microsoft/sp-client-preview';
import
{
  DocumentCard,
  DocumentCardPreview,
  DocumentCardActivity,
  DocumentCardTitle
} from 'office-ui-fabric-react';

export interface IFacilityState {
}
export interface IFacilityProps {
  context?: IWebPartContext;
  item?: any;
}
export default class Facility extends React.Component<IFacilityProps,
IFacilityState> {
  constructor(props: { context : IWebPartContext })
  {
    super(props);
  }
  public render(): JSX.Element {
    return (
      <div>
        <DocumentCard>
          <DocumentCardTitle title={ this.props.item ? this.props.item.name : '' }
/>
          <DocumentCardPreview previewImages={ [
            this.props.item ?
            {
              previewImageSrc:
"https://spawesome.blob.core.windows.net/facilities/" +
this.props.item.name.toLowerCase() + ".jpg"
            } : ''
          ]}/>
          <DocumentCardActivity
            activity='Facility Manager'
            people={
                  this.props.item ?
                   [
                     {
```

```
                        name: this.props.item.facilitiesManagerName,
                        profileImageSrc:
'https://spawesome.blob.core.windows.net/resources/avatar-' +
this.props.item.facilitiesManagerAlias + '.png'
                      }
                    ] : null
                  }
                />
        </DocumentCard>
      </div>
    );
  }
}
```

16. Add the following to facilities.tsx, beneath the div that added the DetailsList:

```
            <div className="ms-Grid-col ms-u-sm6 ms-u-md8 ms-u-lg6">
              <Facility context={this.props.context}
item={this.state.selectedItem}  />
            </div>
```

17. Add a new data type definition, underneath <facilities>/ISPListList.ts.

```
// Define List Models
export interface ISPListList {
  value: ISPList[];
}
export interface ISPList  {
  Title: string;
  Description: string;
}
```

18. Add a Mock Data Http Client to tests/MockListHttpClient.ts:

```
// Setup mock Http client
import { ISPList } from '../ISPListList';
export default class MockListListHttpClient {
    private static _items: ISPList[] = [{ Title: 'Mock Issue List', Description:
'1' }];
    public static get(restUrl: string, options?: any): Promise<ISPList[]> {
      return new Promise<ISPList[]>((resolve) => {
```

```
            resolve(MockListListHttpClient._items);
        });
    }
}
```

19. Add some data retrieval code to FacilitiesWebPart.ts:

```
// Setup the Web Part Property Pane Dropdown options
  private _dropdownOptions: IPropertyPaneDropdownOption[] = [];
  public onInit<T>(): Promise<T> {
      this._getLists()
        .then((response) => {
          this._dropdownOptions = response.value.map((list: ISPList) => {
            return {
              key: list.Title,
              text: list.Title
          };
        });
      });
    return Promise.resolve();
  }
  // Retrieve Lists from SharePoint
  private _getLists(): Promise<ISPListList> {
    if (this.context.environment.type === EnvironmentType.Local) {
      return MockListListHttpClient.get(this.context.pageContext.web.absoluteUrl)
      .then((response) => {
         const listData: ISPListList = {
            value:
            [
                { Title: 'Mock List 1', Description: '1' },
                { Title: 'Mock List 2', Description: '2' },
                { Title: 'Mock List 3', Description: '3' },
                { Title: 'Mock List 4', Description: '4' }
            ]
            };
        return listData;
      });
    }
    else
    {
      return this.context.httpClient.get(this.context.pageContext.web.absoluteUrl
+ `/_api/web/lists`)
        .then((response: Response) => {
        return response.json();
      });
    }
```

```
  }
```

20. Add the following to FacilitiesWebPart.ts, down near the bottom in propertyPaneSettings()

```
    PropertyPaneDropdown('list', {
                  label: 'List',
                  options: this._dropdownOptions
                })
```

21. Add the following to FacilitiesWebPart.ts;

```
import { ISPListList, ISPList } from './ISPListList';

import {
  EnvironmentType
} from '@microsoft/sp-client-base';


import MockListListHttpClient from './tests/MockListListHttpClient';
```

22. Ensure that the rendering function in FacilitiesWebPart.ts looks like:

```
   const element: React.ReactElement<IFacilitiesProps> =
React.createElement(Facilities, {
      description: this.properties.description,
      list: this.properties.list,
      context: this.context
    });
```

23. Add the following to IFacilitiesWebPartProps:

```
export interface IFacilitiesWebPartProps {
  description: string;
  list?: string;
}
```

Add

```
IPropertyPaneDropdownOption,
PropertyPaneDropdown
```

to the list of imports in FacilitiesWebpart.ts

24. Update the following in facilities.tsx

Add this to the IFacilitiesProps interface definition:

```
context: IWebPartContext;
```

Add this at the top of facilities.tsx:

```
import {
  IWebPartContext
} from '@microsoft/sp-client-preview';

import Facility  from './Facility';
```

Update the Constructor to:

```
    constructor(props: { description : string, list : string, context :
IWebPartContext })
```

Retrieve and Display Lists

Add a list retrieval function to your Facility display, to show a list of issues.

25. Create MockIssueListHttpClient.ts underneath tests/ and add the following to it:

```
// Setup mock Http client
import { ISPIssue } from '../ISPIssueList';
export default class MockIssueListHttpClient {
    private static _items: ISPIssue[] = [{ Title: 'Mock Issue List', Description:
'1' }];
    public static get(restUrl: string, options?: any): Promise<ISPIssue[]> {
      return new Promise<ISPIssue[]>((resolve) => {
            resolve(MockIssueListHttpClient._items);
        });
    }
}
```

25. Add the following to ISPIssueList.ts, underneath Facilities:

```
// Define Issue Models
export interface ISPIssueList {
  value: ISPIssue[];
}
export interface ISPIssue {
  Title: string;
  Description: string;
}
```

26. Update facility.tsx:

```
import MockIssueListHttpClient from '../tests/MockIssueListHttpClient';
import { ISPIssueList } from '../ISPIssueList';
export interface IFacilityState {
  issues?: ISPIssueList;
}
export interface IFacilityProps {
  context?: IWebPartContext;
  item?: any;
```

```
  list?: string;
}
```

27. Replace the interior:

```
    this.state = { issues: null };
  }
  private lastList : string  = null;
  private lastItem : string = null;
  // Define and retrieve mock List data
  private _getMockListData(): Promise<ISPIssueList> {
    return
MockIssueListHttpClient.get(this.props.context.pageContext.web.absoluteUrl).then((
) => {
        const listData: ISPIssueList = {
            value:
            [
                { Title: 'Mock Issue 1', Description: '1' },
                { Title: 'Mock Issue 2', Description: '2' },
                { Title: 'Mock Issue 3', Description: '3' },
                { Title: 'Mock Issue 4', Description: '4' }
            ]
            };
        return listData;
    }) as Promise<ISPIssueList>;
  }
  // Retrieve List data from SharePoint
  private _getListData(): Promise<ISPIssueList> {
    return
this.props.context.httpClient.get(this.props.context.pageContext.web.absoluteUrl +
`/_api/web/lists/GetByTitle('` + this.props.list + `')/items?$filter=Facility eq
'` + this.props.item.Title + "'")
      .then((response: Response) => {
      return response.json();
      });
  }
  // Call methods for List data retrieval
  private _retrieveListAsync(): void
  {
    const self = this;
    this.lastItem = this.props.item;
    this.lastList = this.props.list;
    // Mock List data
    if (this.props.context.environment.type === EnvironmentType.Local) {
      this._getMockListData().then((response) => {
        self.setState( {
          issues: response,
        });
```

```
      });
    }
    // Get Full List data
    else {
      this._getListData()
        .then((response) => {
          self.setState( {
            issues: response,
          });
      });
    }
  }
  public render(): JSX.Element {

    if (this.props.item != null && this.props.list != null && this.props.context
!= null
        && (this.props.item != this.lastItem || this.props.list !=
this.lastList))
    {
      this._retrieveListAsync();
    }
```

28. Add a simple renderer:

```
    <div>{ this.props.list ? this.props.list : '(no list was selected.)' }</div>
        <table>
          <tbody>
            {
              this.state.issues ?
              this.state.issues.value.map(function(object, i) {
                return
<tr><td><b>{object.Title}</b></td><td>{object.Description}</td></tr>;
              }) : ''
            }
          </tbody>
        </table>
```

29. Add a

```
 list? : string
```

property to IFacilityProps, and IFacilitiesWebPartProps; add list={this.item.props} to Facilities.tsx and IFacilitiesWebPart.ts as appropriate.

You’re done!