SharePoint list for CRUD (Create, Read, Update, and Delete) operations using the No Framework option.

The CRUD operations will be performed using REST APIs.

**Create SPFx Solution**

**Step 1**  
Open the command prompt. Create a directory for SPFx solution.

1. md spfx-crud-no-framework

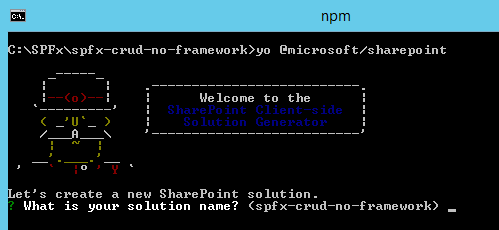
**Step 2**  
Navigate to the above-created directory.

1. cd spfx-crud-no-framework

**Step 3**  
Run Yeoman SharePoint Generator to create the solution.

1. yo @microsoft/sharepoint

**Step 4**  
Yeoman generator will present you with the wizard by asking questions about the solution to be created.



**Solution Name**Hit Enter to have the default name (spfx-crud-no-framework in this case) or type in any other name for your solution.  
  
Selected choice - Hit Enter.

**Target for component**Here, we can select the target environment where we are planning to deploy the client webpart; i.e., SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).  
  
Selected choice - SharePoint Online only (latest).

**Location of files**We may choose to use the same folder or create a subfolder for our solution.  
  
Selected choice - Same folder.

**Deployment option**Selecting Y will allow the app to be deployed instantly to all sites and will be accessible everywhere.  
  
Selected choice - N (install on each site explicitly).

**Type of client-side component to create**We can choose to create a client-side webpart or an extension. Choose the webpart option.  
  
Selected choice - WebPart.

**Web part name**Hit Enter to select the default name or type in any other name.  
  
Selected choice - NoFrameworkCRUD.

**Web part description**Hit Enter to select the default description or type in any other value.  
  
Selected choice - CRUD operations with no framework.

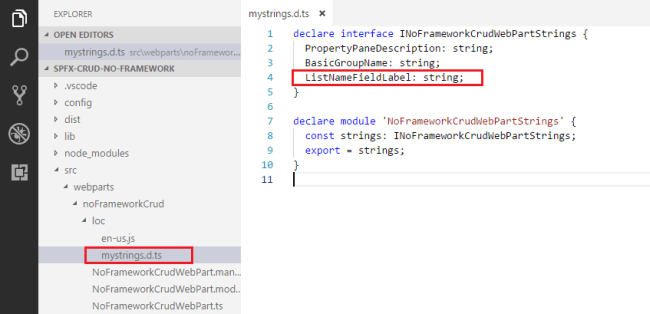
**Framework to use**Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)  
  
Selected choice - No JavaScript Framework.

**Step 5**  
Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take a significant amount of time.  
  
**Step 6**  
Once the scaffolding process is completed, in the command prompt type the below command to open the solution in the code editor of your choice.

1. code .

**Configure Property for List Name**  
SPFx solution by default has the description property created. Let us change the property to list name. We will use this property to configure the list name on which the CRUD operation is to be performed.

**Step 1**  
Open mystrings.d.ts under \src\webparts\noFrameworkCrud\loc\ folder.  
  
**Step 2**  
Rename DescriptionFieldLabel to ListNameFieldLabel.



declare interface INoFrameworkCrudWebPartStrings {

  PropertyPaneDescription: string;

  BasicGroupName: string;

  ListNameFieldLabel: string;

}

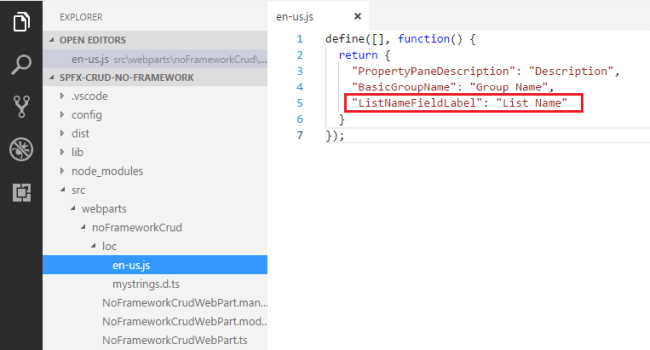
declare module 'NoFrameworkCrudWebPartStrings' {

  const strings: INoFrameworkCrudWebPartStrings;

  export = strings;

}

**Step 3**  
In en-us.js file under \src\webparts\noFrameworkCrud\loc\ folder set the display name for listName property.



define([], function() {

  return {

    "PropertyPaneDescription": "Description",

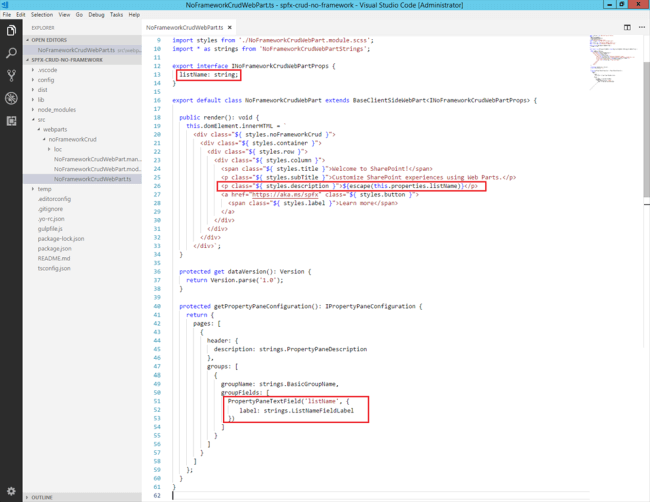
    "BasicGroupName": "Group Name",

    "ListNameFieldLabel": "List Name"

  }

});

**Step 4**  
Open main webpart file (NoFrameworkCrudWebPart.ts) under \src\webparts\noFrameworkCrud folder.  
  
**Step 5**  
Rename description property pane field to listName.



import { Version } from '@microsoft/sp-core-library';

import {

  BaseClientSideWebPart,

  IPropertyPaneConfiguration,

  PropertyPaneTextField

} from '@microsoft/sp-webpart-base';

import { escape } from '@microsoft/sp-lodash-subset';

import { SPHttpClient, SPHttpClientResponse } from '@microsoft/sp-http';

import { IListItem } from './IListItem';

import styles from './NoFrameworkCrudWebPart.module.scss';

import \* as strings from 'NoFrameworkCrudWebPartStrings';

export interface INoFrameworkCrudWebPartProps {

  listName: string;

}

export default class NoFrameworkCrudWebPart extends BaseClientSideWebPart<INoFrameworkCrudWebPartProps> {

  private listItemEntityTypeName: string = undefined;

  public render(): void {

    this.domElement.innerHTML = `

      <div class="${ styles.noFrameworkCrud }">

        <div class="${ styles.container }">

          <div class="${ styles.row }">

            <div class="${ styles.column }">

              <span class="${ styles.title }">Welcome to SharePoint!</span>

              <p class="${ styles.subTitle }">Customize SharePoint experiences using Web Parts.</p>

              <p class="${ styles.description }">${escape(this.properties.listName)}</p>

          <a href="https://aka.ms/spfx" class="${ styles.button }">

                <span class="${ styles.label }">Learn more</span>

              </a>

            </div>

          </div>

        </div>

      </div>`;

  }

  protected get dataVersion(): Version {

    return Version.parse('1.0');

  }

  protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {

    return {

      pages: [

        {

          header: {

            description: strings.PropertyPaneDescription

          },

          groups: [

            {

              groupName: strings.BasicGroupName,

              groupFields: [

                PropertyPaneTextField('listName', {

                  label: strings.ListNameFieldLabel

                })

              ]

            }

          ]

        }

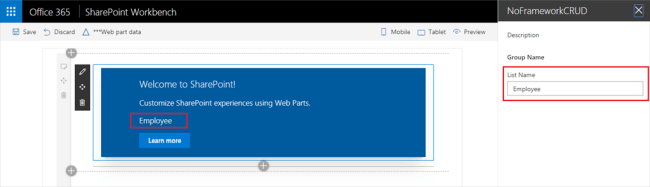
      ]

    };

  }

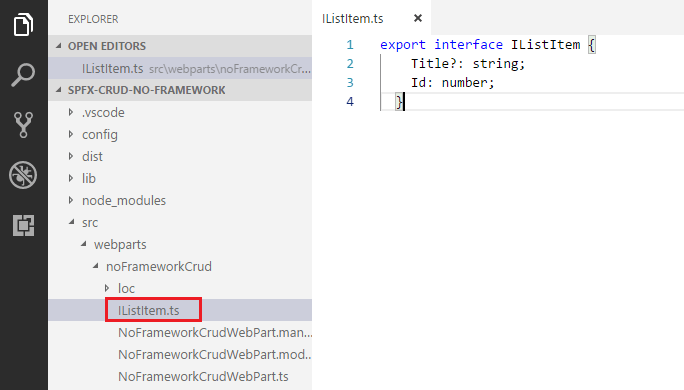
}

**Step 6**  
In the command prompt, type “gulp serve”.  
  
**Step 7**  
In the SharePoint local workbench page, add the web part.  
  
**Step 8**  
Edit the web part to ensure the listName property pane field is getting reflected.



**Model for List Item**

Let us add a class (IListItem.ts) representing the list item.



export interface IListItem {

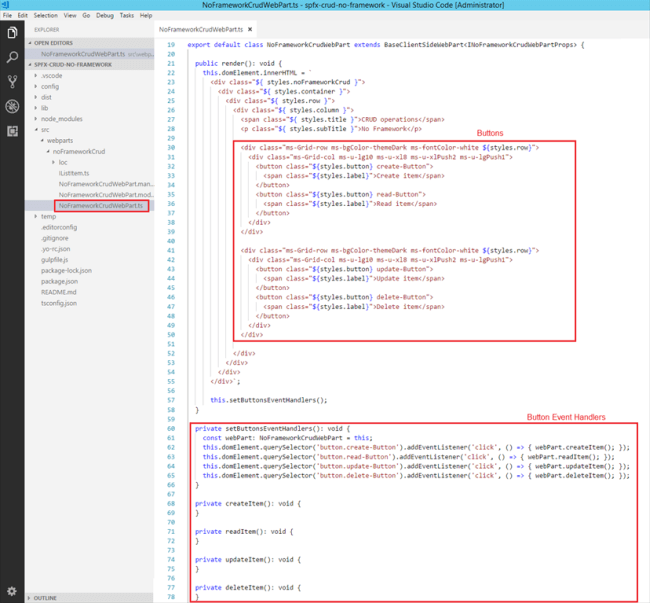
    Title?: string;

    Id: number;

}

**Add Controls to WebPart**

**Step 1**  
Open main webpart file (NoFrameworkCrudWebPart.ts) under \src\webparts\noFrameworkCrud folder.  
  
**Step 2**  
Modify Render method to include buttons for CRUD operations and add event handlers to each of the button.



**export** **default** **class** NoFrameworkCrudWebPart **extends** BaseClientSideWebPart<INoFrameworkCrudWebPartProps> {

**private** listItemEntityTypeName: string = undefined;

**public** render(): **void** {

**this**.domElement.innerHTML = `

      <div **class**="${ styles.noFrameworkCrud }">

        <div **class**="${ styles.container }">

          <div **class**="${ styles.row }">

            <div **class**="${ styles.column }">

              <span **class**="${ styles.title }">CRUD operations</span>

              <p **class**="${ styles.subTitle }">No Framework</p>

              <p **class**="${ styles.description }">Name: ${escape(**this**.properties.listName)}</p>

              <div **class**="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">

                <div **class**="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">

                  <button **class**="${styles.button} create-Button">

                    <span **class**="${styles.label}">Create item</span>

                  </button>

                  <button **class**="${styles.button} read-Button">

                    <span **class**="${styles.label}">Read item</span>

                  </button>

                </div>

              </div>

              <div **class**="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">

                <div **class**="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">

                  <button **class**="${styles.button} update-Button">

                    <span **class**="${styles.label}">Update item</span>

                  </button>

                  <button **class**="${styles.button} delete-Button">

                    <span **class**="${styles.label}">Delete item</span>

                  </button>

                </div>

              </div>

              <div **class**="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">

                <div **class**="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">

                  <div **class**="status"></div>

                  <ul **class**="items"><ul>

                </div>

              </div>

            </div>

          </div>

        </div>

      </div>`;

**this**.setButtonsEventHandlers();

  }

**private** setButtonsEventHandlers(): **void** {

**const** webPart: NoFrameworkCrudWebPart = **this**;

**this**.domElement.querySelector('button.create-Button').addEventListener('click', () => { webPart.createItem(); });

**this**.domElement.querySelector('button.read-Button').addEventListener('click', () => { webPart.readItem(); });

**this**.domElement.querySelector('button.update-Button').addEventListener('click', () => { webPart.updateItem(); });

**this**.domElement.querySelector('button.delete-Button').addEventListener('click', () => { webPart.deleteItem(); });

  }

**private** createItem(): **void** {

  }

**private** readItem(): **void** {

  }

**private** updateItem(): **void** {

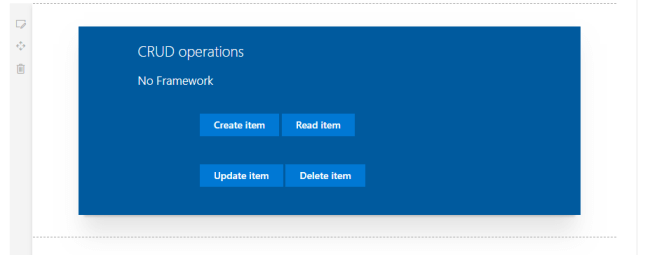
  }

**private** deleteItem(): **void** {

  }

}

**Step 3**  
In the command prompt type “gulp serve” to see the buttons on the webpart.



**Step 4**  
We will perform read, update, and delete operations on the latest item in the SharePoint list. Let us implement a generic method (getLatestItemId) which will return the id of latest item from a given list. We will use the REST API to query the list.

**private** getLatestItemId(): Promise<number> {

**return** **new** Promise<number>((resolve: (itemId: number) => **void**, reject: (error: any) => **void**): **void** => {

**this**.context.spHttpClient.get(`${**this**.context.pageContext.web.absoluteUrl}/\_api/web/lists/getbytitle('${this.properties.listName}')/items?$orderby=Id desc&$top=1&$select=id`,

      SPHttpClient.configurations.v1,

      {

        headers: {

          'Accept': 'application/json;odata=nometadata',

          'odata-version': ''

        }

      })

      .then((response: SPHttpClientResponse): Promise<{ value: { Id: number }[] }> => {

**return** response.json();

      }, (error: any): **void** => {

        reject(error);

      })

      .then((response: { value: { Id: number }[] }): **void** => {

**if** (response.value.length === 0) {

          resolve(-1);

        }

**else** {

          resolve(response.value[0].Id);

        }

      });

  });

}

**Implement Create Operation**

First, start by implementing the Create method, which will add an item to SharePoint list.

**private** createItem(): **void** {

**const** body: string = JSON.stringify({

    'Title': `Item ${**new** Date()}`

  });

**this**.context.spHttpClient.post(`${**this**.context.pageContext.web.absoluteUrl}/\_api/web/lists/getbytitle('${this.properties.listName}')/items`,

  SPHttpClient.configurations.v1,

  {

    headers: {

      'Accept': 'application/json;odata=nometadata',

      'Content-type': 'application/json;odata=nometadata',

      'odata-version': ''

    },

    body: body

  })

  .then((response: SPHttpClientResponse): Promise<IListItem> => {

**return** response.json();

  })

  .then((item: IListItem): **void** => {

**this**.updateStatus(`Item '${item.Title}' (ID: ${item.Id}) successfully created`);

  }, (error: any): **void** => {

**this**.updateStatus('Error while creating the item: ' + error);

  });

}

**private** updateStatus(status: string, items: IListItem[] = []): **void** {

**this**.domElement.querySelector('.status').innerHTML = status;

**this**.updateItemsHtml(items);

}

**private** updateItemsHtml(items: IListItem[]): **void** {

**this**.domElement.querySelector('.items').innerHTML = items.map(item => `<li>${item.Title} (${item.Id})</li>`).join("");

}

**Implement Read Operation**We will use the REST API to read the latest item.

**private** readItem(): **void** {

**this**.getLatestItemId()

    .then((itemId: number): Promise<SPHttpClientResponse> => {

**if** (itemId === -1) {

**throw** **new** Error('No items found in the list');

      }

**this**.updateStatus(`Loading information about item ID: ${itemId}...`);

**return** **this**.context.spHttpClient.get(`${**this**.context.pageContext.web.absoluteUrl}/\_api/web/lists/getbytitle('${this.properties.listName}')/items(${itemId})?$select=Title,Id`,

        SPHttpClient.configurations.v1,

        {

          headers: {

            'Accept': 'application/json;odata=nometadata',

            'odata-version': ''

          }

        });

    })

    .then((response: SPHttpClientResponse): Promise<IListItem> => {

**return** response.json();

    })

    .then((item: IListItem): **void** => {

**this**.updateStatus(`Item ID: ${item.Id}, Title: ${item.Title}`);

    }, (error: any): **void** => {

**this**.updateStatus('Loading latest item failed with error: ' + error);

    });

}

**Implement Update Operation**Firstly, we will get the latest item and update it.

**private** updateItem(): **void** {

  let latestItemId: number = undefined;

**this**.updateStatus('Loading latest item...');

**this**.getLatestItemId()

    .then((itemId: number): Promise<SPHttpClientResponse> => {

**if** (itemId === -1) {

**throw** **new** Error('No items found in the list');

      }

      latestItemId = itemId;

**this**.updateStatus(`Loading information about item ID: ${itemId}...`);

**return** **this**.context.spHttpClient.get(`${**this**.context.pageContext.web.absoluteUrl}/\_api/web/lists/getbytitle('${this.properties.listName}')/items(${latestItemId})?$select=Title,Id`,

        SPHttpClient.configurations.v1,

        {

          headers: {

            'Accept': 'application/json;odata=nometadata',

            'odata-version': ''

          }

        });

    })

    .then((response: SPHttpClientResponse): Promise<IListItem> => {

**return** response.json();

    })

    .then((item: IListItem): **void** => {

**this**.updateStatus(`Item ID1: ${item.Id}, Title: ${item.Title}`);

**const** body: string = JSON.stringify({

        'Title': `Updated Item ${**new** Date()}`

      });

**this**.context.spHttpClient.post(`${**this**.context.pageContext.web.absoluteUrl}/\_api/web/lists/getbytitle('${this.properties.listName}')/items(${item.Id})`,

        SPHttpClient.configurations.v1,

        {

          headers: {

            'Accept': 'application/json;odata=nometadata',

            'Content-type': 'application/json;odata=nometadata',

            'odata-version': '',

            'IF-MATCH': '\*',

            'X-HTTP-Method': 'MERGE'

          },

          body: body

        })

        .then((response: SPHttpClientResponse): **void** => {

**this**.updateStatus(`Item **with** ID: ${latestItemId} successfully updated`);

        }, (error: any): **void** => {

**this**.updateStatus(`Error updating item: ${error}`);

        });

    });

}

**Implement Delete Operation**REST APIs are used to find and delete the latest item.

**private** deleteItem(): **void** {

**if** (!window.confirm('Are you sure you want to delete the latest item?')) {

**return**;

  }

**this**.updateStatus('Loading latest items...');

  let latestItemId: number = undefined;

  let etag: string = undefined;

**this**.getLatestItemId()

    .then((itemId: number): Promise<SPHttpClientResponse> => {

**if** (itemId === -1) {

**throw** **new** Error('No items found in the list');

      }

      latestItemId = itemId;

**this**.updateStatus(`Loading information about item ID: ${latestItemId}...`);

**return** **this**.context.spHttpClient.get(`${**this**.context.pageContext.web.absoluteUrl}/\_api/web/lists/getbytitle('${this.properties.listName}')/items(${latestItemId})?$select=Id`,

        SPHttpClient.configurations.v1,

        {

          headers: {

            'Accept': 'application/json;odata=nometadata',

            'odata-version': ''

          }

        });

    })

    .then((response: SPHttpClientResponse): Promise<IListItem> => {

      etag = response.headers.get('ETag');

**return** response.json();

    })

    .then((item: IListItem): Promise<SPHttpClientResponse> => {

**this**.updateStatus(`Deleting item **with** ID: ${latestItemId}...`);

**return** **this**.context.spHttpClient.post(`${**this**.context.pageContext.web.absoluteUrl}/\_api/web/lists/getbytitle('${this.properties.listName}')/items(${item.Id})`,

        SPHttpClient.configurations.v1,

        {

          headers: {

            'Accept': 'application/json;odata=nometadata',

            'Content-type': 'application/json;odata=verbose',

            'odata-version': '',

            'IF-MATCH': etag,

            'X-HTTP-Method': 'DELETE'

          }

        });

    })

    .then((response: SPHttpClientResponse): **void** => {

**this**.updateStatus(`Item **with** ID: ${latestItemId} successfully deleted`);

    }, (error: any): **void** => {

**this**.updateStatus(`Error deleting item: ${error}`);

    });

}

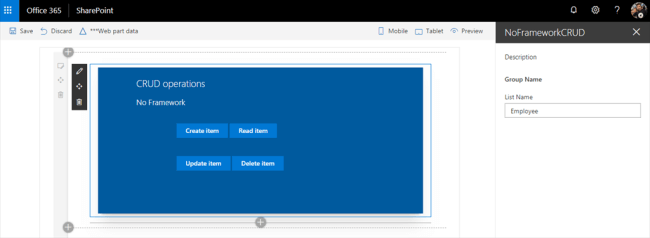
**Test the WebPart**

**Step 1**  
On the command prompt, type “gulp serve”.

**Step 2**  
Open SharePoint site.

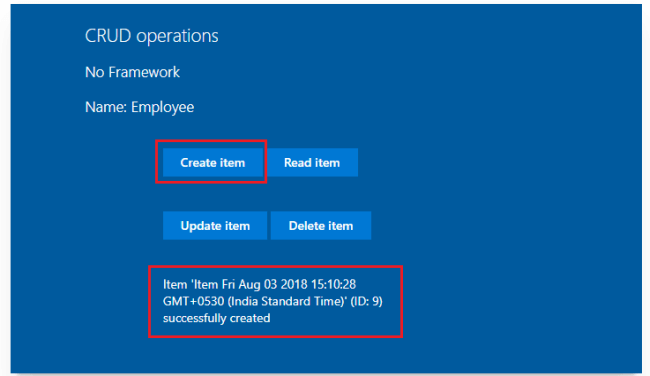
**Step 3**  
Navigate to /\_layouts/15/workbench.aspx.

**Step 4**  
Add the webpart to the page.  
  
**Step 5**  
Edit webpart; in the Properties pane, type the list name.

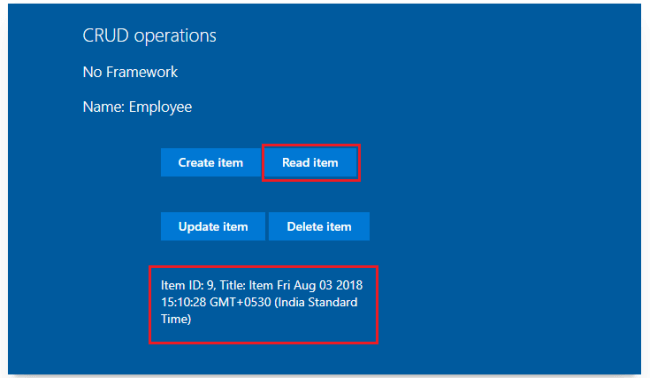


**Step 6**  
Click the buttons (Create Item, Read Item, Update Item, and Delete Item) one by one to test the webpart.  
  
**Step 7**  
Verify the operations are taking place in the SharePoint list.

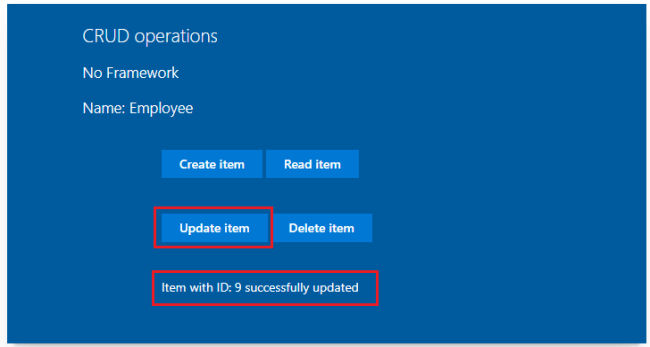
**Create Operation**



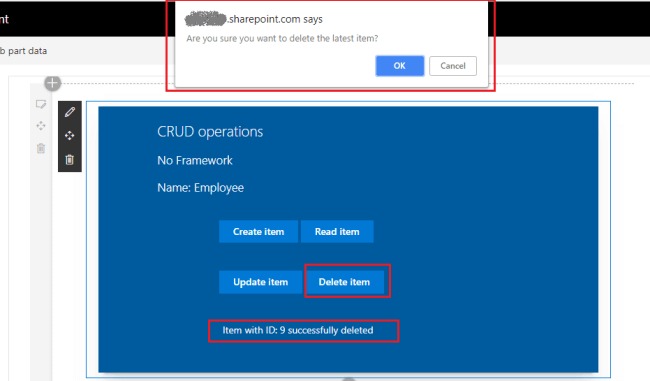
**Read Operation**



**Update Operation**

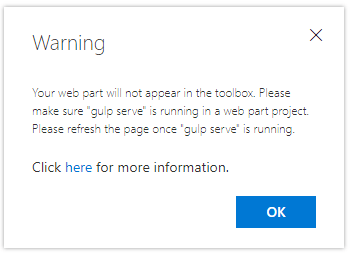


**Delete Operation**



**Troubleshooting**

In some cases, the SharePoint workbench (https://[tenant].sharepoint.com/\_layouts/15/workbench.aspx) shows this error even if the “gulp serve” is running.



Open the below URL in the next tab of the browser. Accept the warning message.

https://localhost:4321/temp/manifests.js

**Summary**

SharePoint framework client web parts can be developed with No JavaScript options. REST APIs can be used to perform the CRUD operations on SharePoint list.