Laboratorio SPFx Connecting to SharePoint & React Lifecycle

**Office Developer Bootcamp 2017**

Mario Cortés Flores

@mariocortesf

<https://geeks.ms/rockyouroffice365/>

Ejemplos:

<https://github.com/CompartiMOSS/Global-Office-365-Developer-Bootcamp/tree/master/SPFX%20Connecting%20to%20SP%20and%20React>

Contents

[1. Configuración de entorno de desarrollo 2](#_Toc497560073)

[2. Configuración de tenant 2](#_Toc497560074)

[3. Creación del Proyecto 2](#_Toc497560075)

[4. Añadir Office Fabric 4](#_Toc497560076)

[5. Obtener datos de SharePoint 9](#_Toc497560077)

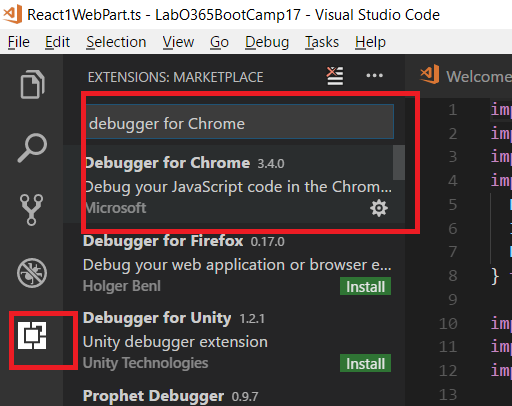
[6. Referencias 11](#_Toc497560078)

# Configuración de entorno de desarrollo

Antes de empezar el laboratorio debes asegurarte de disponer de las herramientas de desarrollo.

Consulta los pasos desde este enlace: <https://github.com/OfficeDev/TrainingContent/blob/master/SharePoint/SharePointFramework/01%20Getting%20Started%20with%20the%20SharePoint%20Framework/Lab.md>

Es recomendable que adicionalmente instales la extensión “Debugger for Chrome” para poder depurar el comportamiento de tus desarrollos desde Visual Studio Code.



# Configuración de tenant

Para poder desplegar tus ejemplos necesitarás disponer de una colección de sitios de SharePoint Online de tipo “Developer site”.

Encontrarás los pasos para configurar tu tenant desde el enlace:

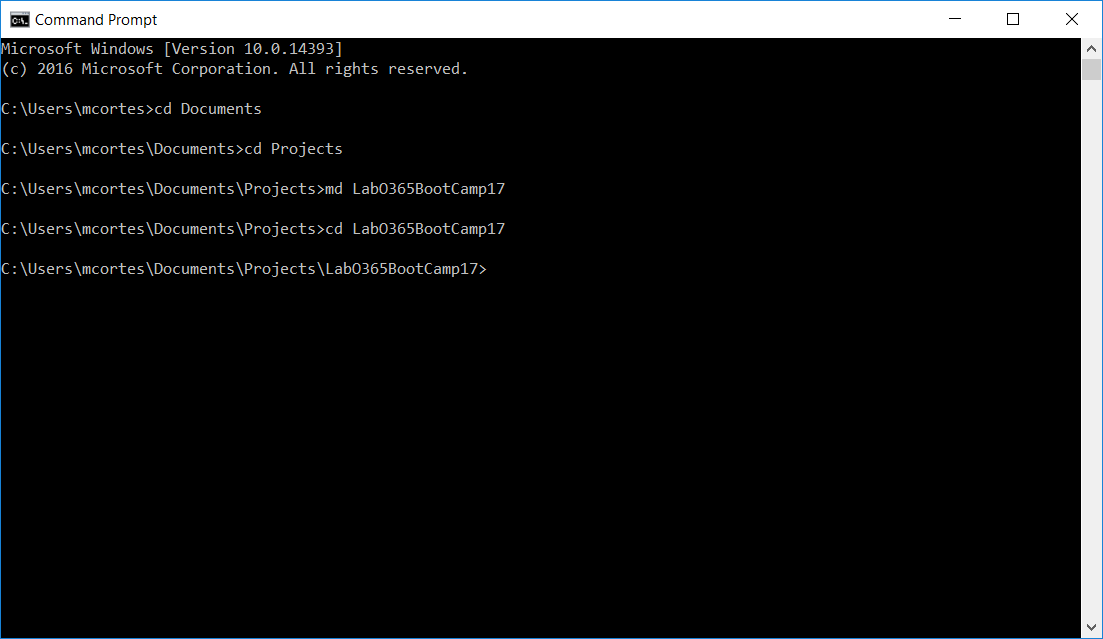
<https://github.com/OfficeDev/TrainingContent/blob/master/SharePoint/SharePointFramework/01%20Getting%20Started%20with%20the%20SharePoint%20Framework/Lab.md>

# Creación del Proyecto

Abre una consola de comandos de sistema CMD y ubícate en el directorio de trabajo donde deseas crear el Proyecto.

Una vez ubicado en tu carpeta de trabajo, crea una nueva carpeta para tu proyecto:

|  |
| --- |
| md LabO365BootCamp17  cd LabO365BootCamp17 |

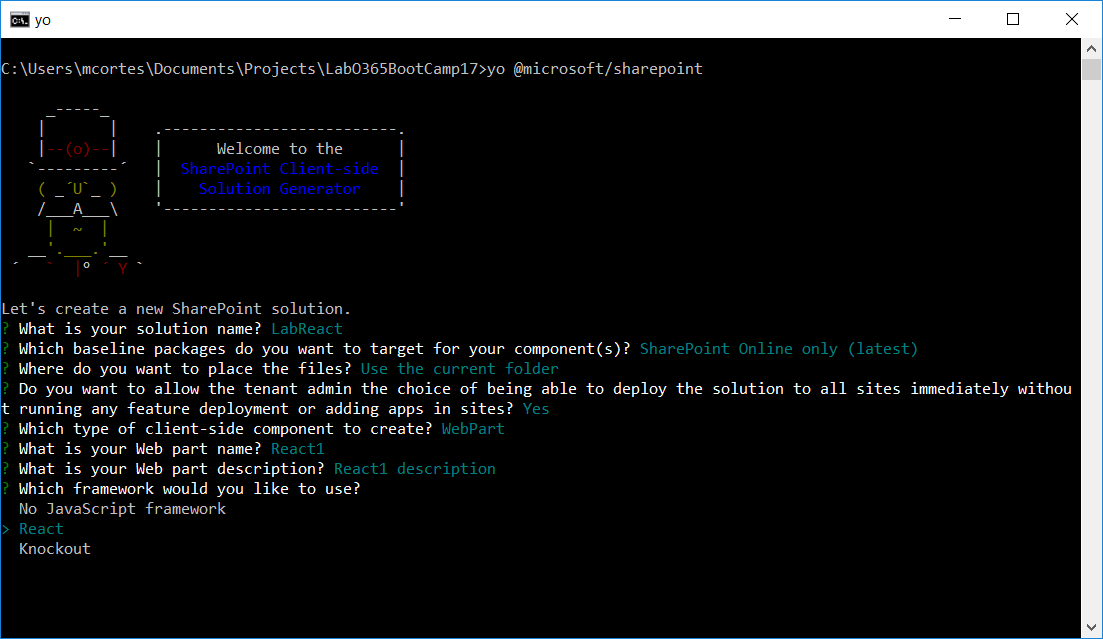


Crearemos con Yeoman el Proyecto del laboratorio basado en React:

|  |
| --- |
| yo @microsoft/sharepoint |

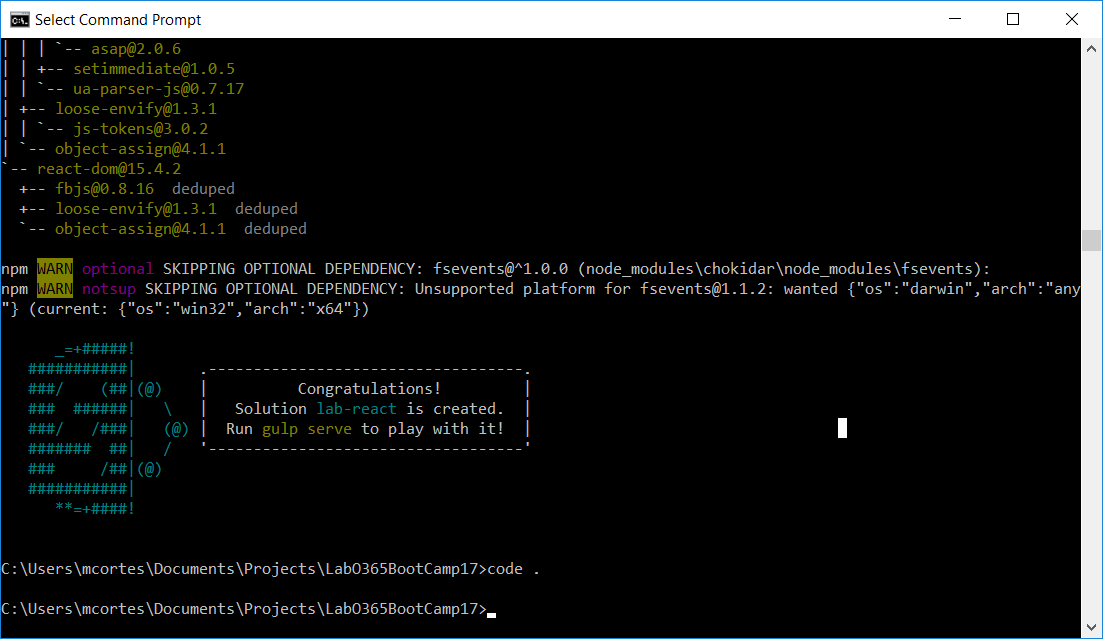
Indicaremos:

* LabReact
* SharePointOnline
* Use the current folder
* Yes
* WebPart
* React1
* Sample
* React

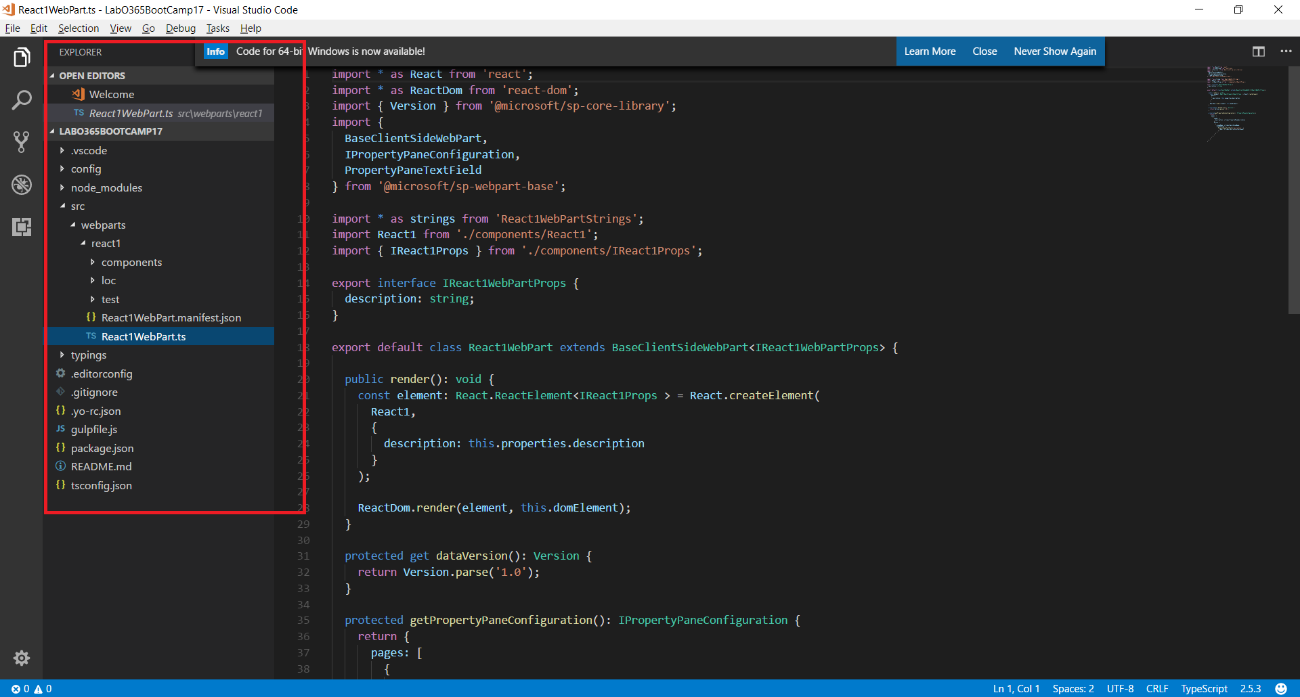


Una vez finalizado el proceso de creación, abriremos el proyecto con el editor, en este caso Visual Studio Code:

|  |
| --- |
| Code . |



Se abrirá entonces el editor de Visual Studio Code con el proyecto abierto. Revisa la estructura del proyecto desde el explorador de ficheros:

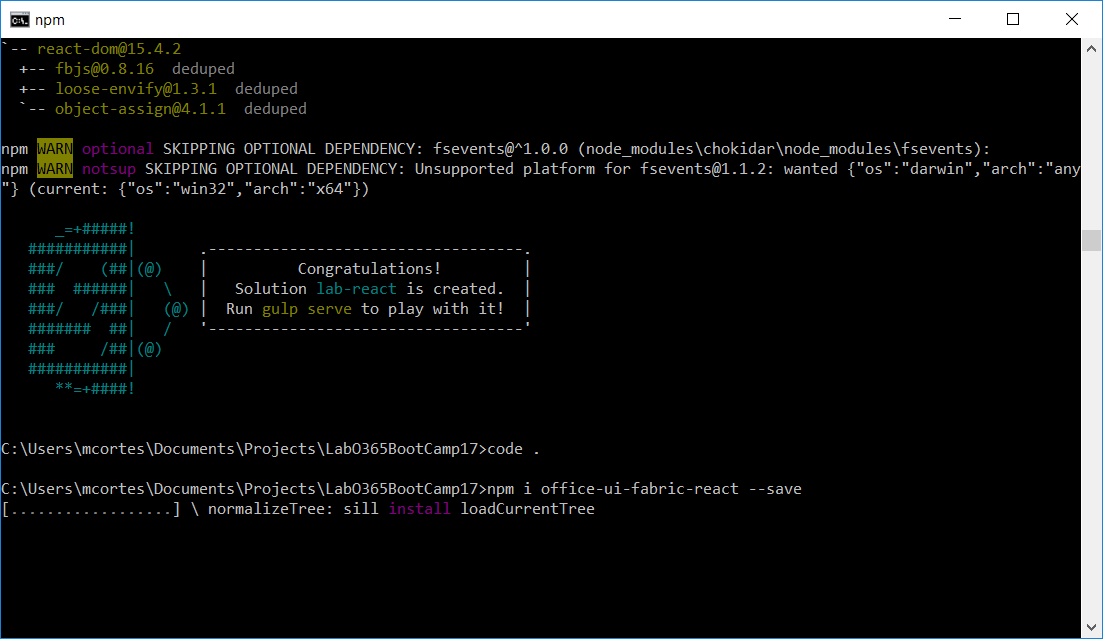


# Añadir Office Fabric

En este apartado utilizaremos los controles de Office Fabric UI desde nuestro WebPart.

Antes de comenzar, instalaremos el paquete de Office Fabric, desde la consola de comandos, ejecutaremos el siguiente comando:

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



Desde el explorador del Visual Studio Code navegaremos hasta la clase “React1.tsx” que encontrarás en la carpeta “src/webparts/react1/components”.

Añadiremos los siguientes import de Office Fabric para poder utilizar los controles de botones y listado de elementos:

|  |
| --- |
| import {  IButtonProps,  PrimaryButton  } from 'office-ui-fabric-react/lib/Button';  import {  DetailsList,  DetailsListLayoutMode,  Selection,  IColumn  } from 'office-ui-fabric-react/lib/DetailsList'; |

En el método Render dejaremos un código como este:

|  |
| --- |
| return (  <div className="ms-Grid">  <div className="ms-Grid-row">  <div className="ms-Grid-col ms-sm12">  <PrimaryButton  data-automation-id='cargar'  target='\_blank'  title='Cargar'  style={ { color: '#ffffff' } }  onClick={this.handleClick}  >  Cargar  </PrimaryButton>  </div>  </div>  <div className="ms-Grid-row">  <div className="ms-Grid-col ms-sm12">  <MarqueeSelection selection={null}>  <DetailsList  items={ this.state.items }  columns={ this.\_columns }  setKey='set'  layoutMode={ DetailsListLayoutMode.justified }  selectionPreservedOnEmptyClick={ true }  ariaLabelForSelectionColumn='Toggle selection'  ariaLabelForSelectAllCheckbox='Toggle selection for all items'  />  </MarqueeSelection>  </div>  </div>  </div> |

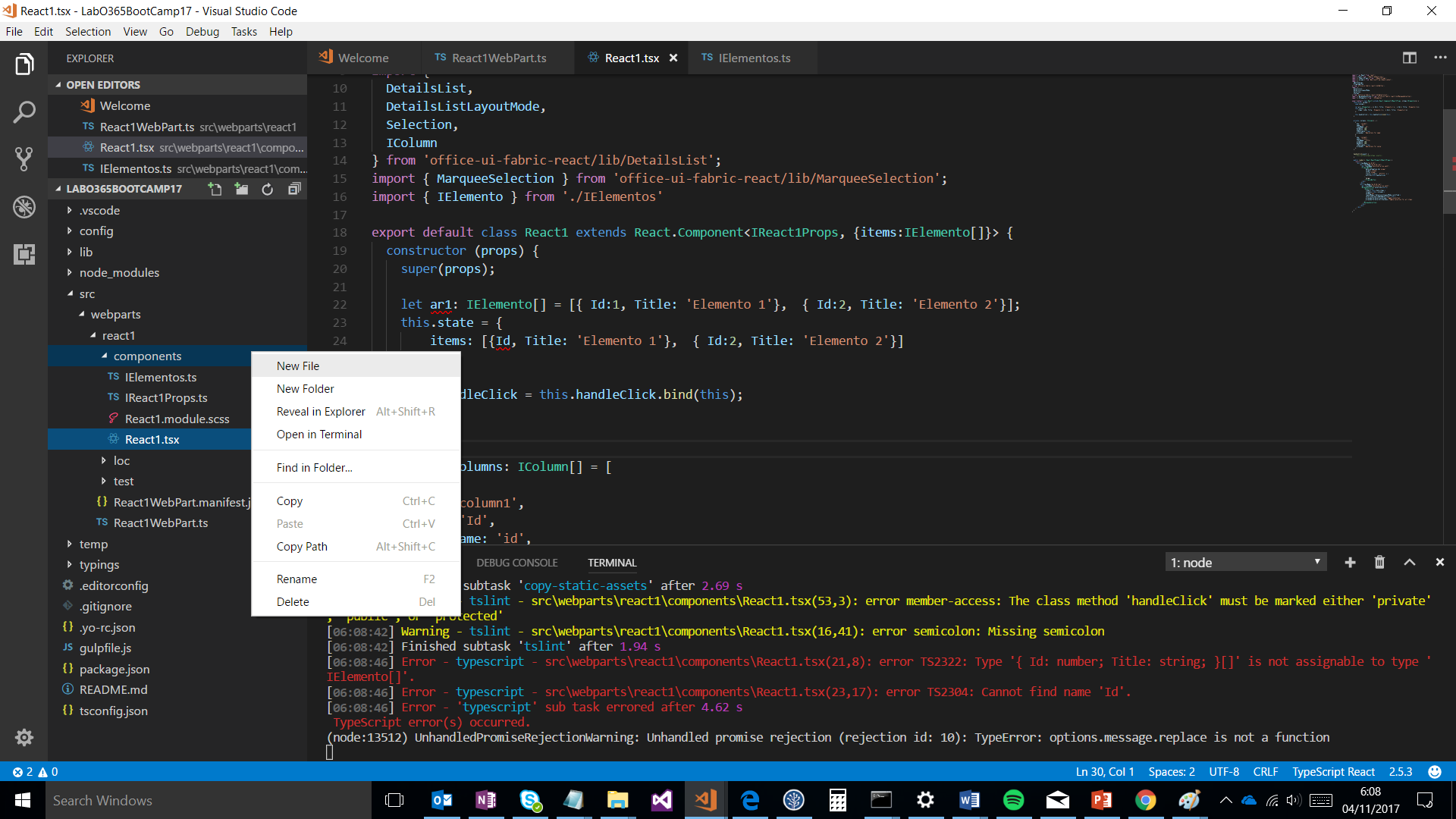
A continuación declararemos las columnas del listado declarando una variable en la clase:

|  |
| --- |
| private \_columns: IColumn[] = [  {  key: 'column1',  name: 'Id',  fieldName: 'Id',  minWidth: 100,  maxWidth: 200,  isResizable: true,  ariaLabel: 'Operations for name'  },  {  key: 'column2',  name: 'Título',  fieldName: 'Title',  minWidth: 100,  maxWidth: 200,  isResizable: true,  ariaLabel: 'Operations for value'  },  ]; |

Añadiremos una variable para almacenar el contexto de SharePoint proveniente del WebPart que inició la carga del componente:

|  |
| --- |
| private \_context: WebPartContext; |

Crearemos una interfaz IElementos para soportar la información de los elementos que se mostrarán en la lista. Desde el explorador, seleccionaremos la carpeta “components” con el botón derecho y pulsaremos en NewFile indicando el nombre “IElementos.ts”:



Definiremos la interfaz IElementos como:

|  |
| --- |
| export interface IElemento {  Id: string;  Title: string;  } |

Volveremos a la clase React1.tsx” y añadiremos un import a esta interfaz:

|  |
| --- |
| import { IElemento } from './IElementos' |

Modificaremos la definición de la clase “React1” para especificar el state:

|  |
| --- |
| export default class React1 extends React.Component<IReact1Props, {items:IElemento[]}> { |

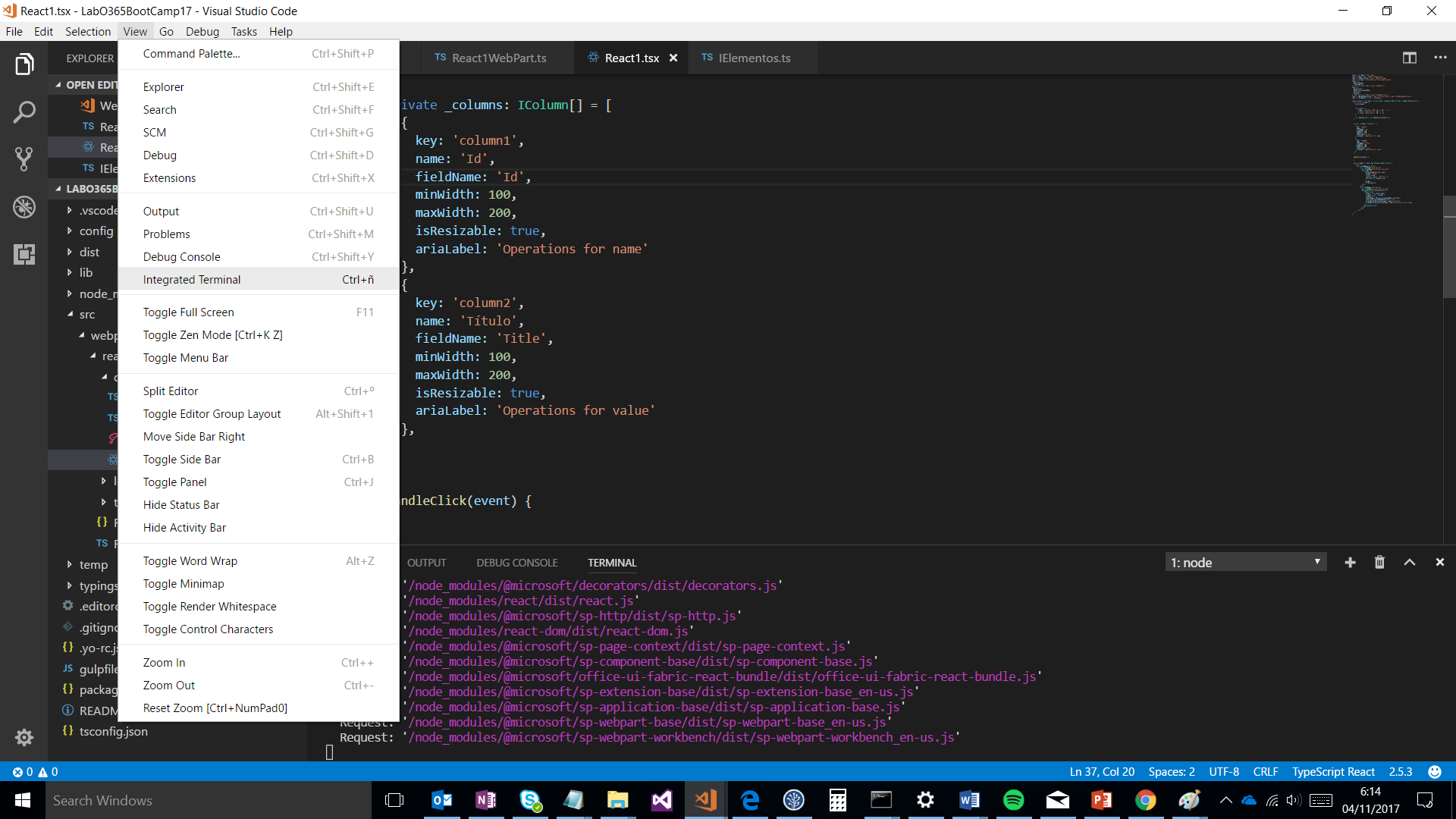
Añadiremos un constructor a la clase “React1” para definir el estado y el evento del botón:

|  |
| --- |
| constructor (props) {  super(props);    this.\_context = this.props.context;  this.state = {  items: [{ Title: 'Mock List', Id: '1' },  { Title: 'Mock List 2', Id: '2' },  { Title: 'Mock List 3', Id: '3' }]  };  this.handleClick = this.handleClick.bind(this);  } |

Crearemos el método “handleClick” para capturar el evento del botón y actualizar el estado con los resultados de la llamada a SharePoint, de momento estará vacío:

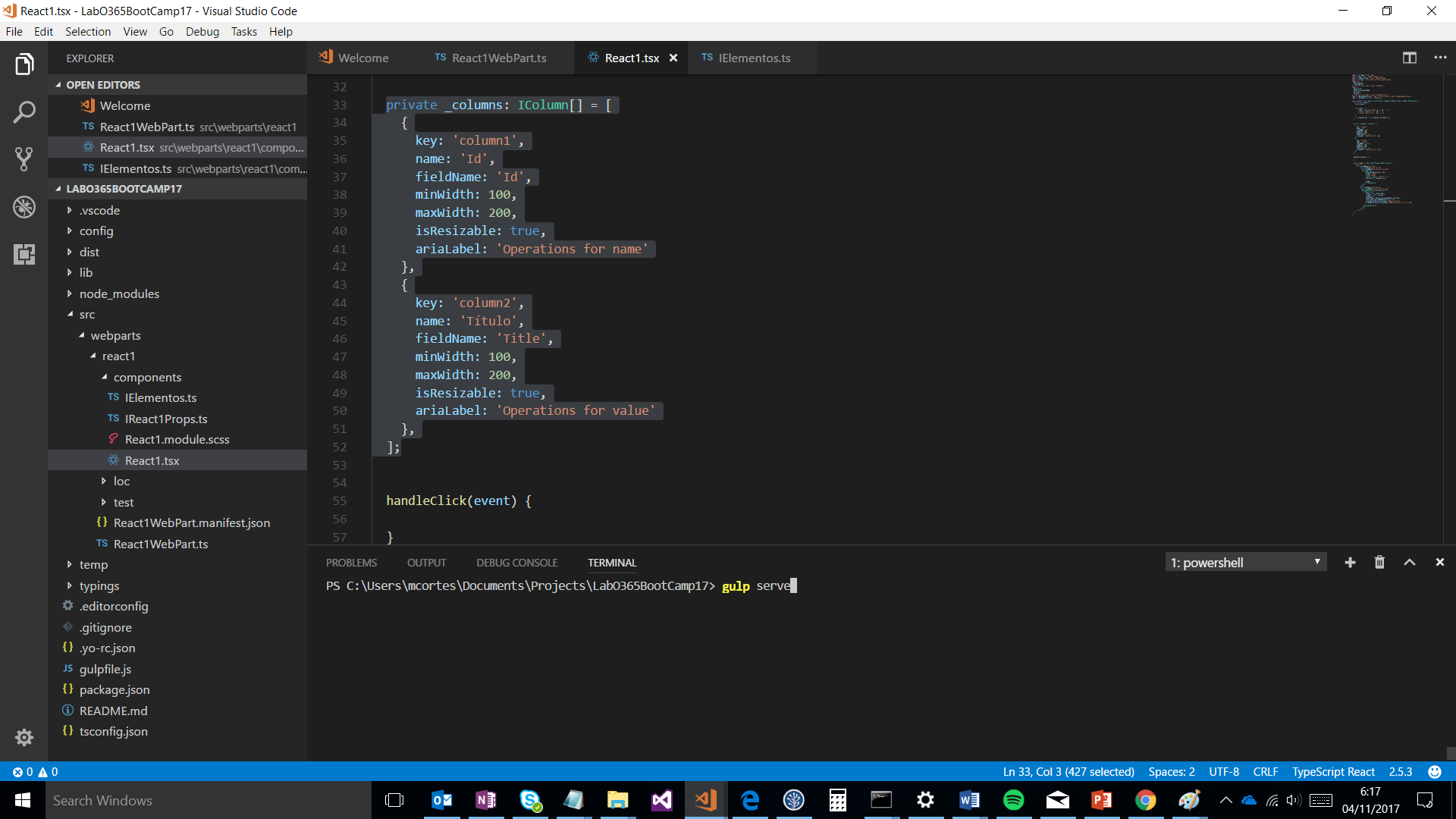
|  |
| --- |
| handleClick(event) {    } |

A continuación probaremos nuestro WebPart en local para lo que lanzaremos la compilación mediante el comando “gulp serve”. Podremos lanzar el comando o bien desde la consola de comandos anterior, o bien desde VSCode desde las opciones de menú “Integrated Terminal”:

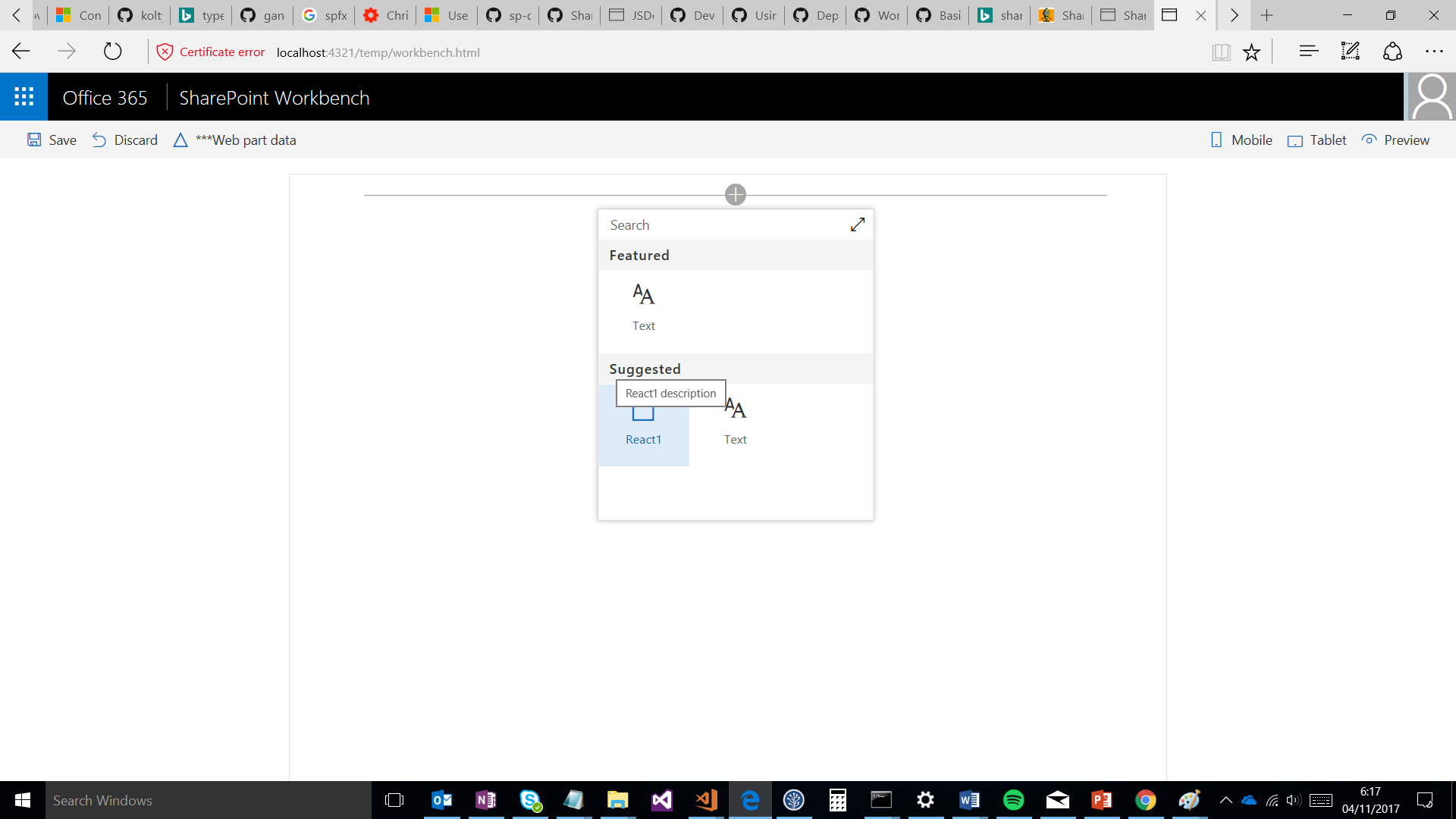


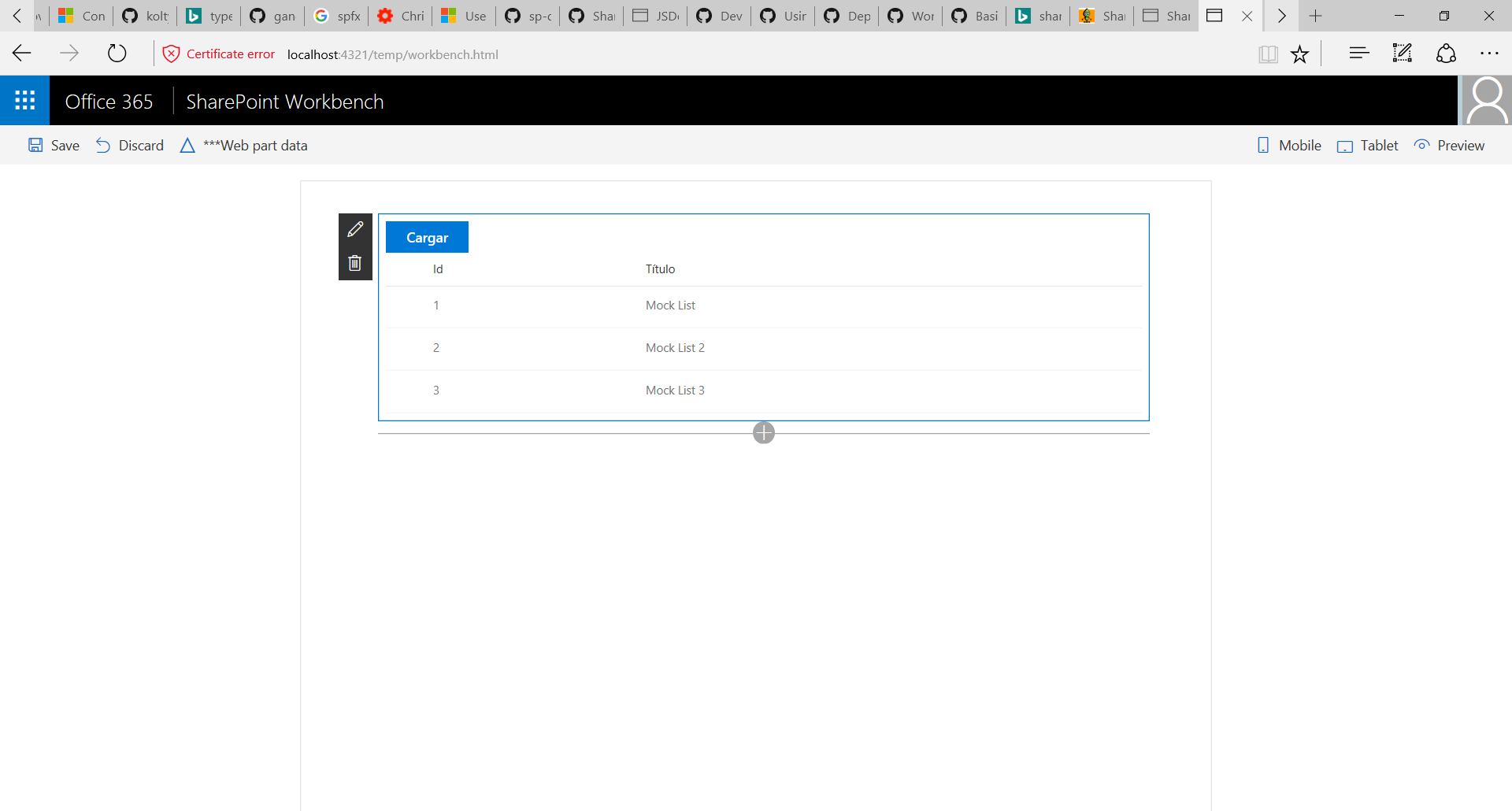
Introduciremos el comando:

|  |
| --- |
| gulp serve |



Al finalizar el proceso, se abrirá una ventana del navegador con la página Workbench desde la que podremos añadir nuestro webpart:





# Obtener datos de SharePoint

En este apartado aprenderemos a implementar un método que obtenga los elementos de una lista a partir de la clase SPHttpClient y se mostrarán en el control.

Modificaremos la interfaz “IReact1Props” añadiendo una nueva propiedad para guardar el nombre de la lista a consultar:

|  |
| --- |
| import { WebPartContext } from "@microsoft/sp-webpart-base";  export interface IReact1Props {  description: string;  listname:string;  context :WebPartContext;  } |

Desde Visual Studio Code crearemos un nuevo fichero “SPclient.ts” en la carpeta “components” para realizar las llamadas a SharePoint.

Añadiremos los import a la interfaz IELemento y el cliente SPHttp:

|  |
| --- |
| import { IElemento } from './IElementos';  import {  SPHttpClient,  SPHttpClientResponse  } from '@microsoft/sp-http'; |

Para poder instanciar un mock cuando estamos en local añadiremos el siguiente import:

|  |
| --- |
| import {  Environment,  EnvironmentType  } from '@microsoft/sp-core-library'; |

Definiremos la clase SPCliente con un método get que será el que utilicemos para devolver los elementos de la lista:

|  |
| --- |
| export default class SPClient {  public static get(context:any,listName:string): Promise<IElemento[]> {  if (Environment.type === EnvironmentType.Local) {  // Mock en modo Depuración local  return new Promise<IElemento[]>((resolve) => {  resolve(  [  { Title: 'Elemento 1', Id: '1' },  { Title: 'Elemento 2', Id: '2' }  ]);  });  }  return context.spHttpClient.get(context.pageContext.web.absoluteUrl + `/\_api/web/lists/getbytitle('${listName}')/items?$select=Title,Id`,  SPHttpClient.configurations.v1)  .then((response: SPHttpClientResponse) => {  return response.json().then((responseJSON: any) => {  return responseJSON.value;          });  });    }  } |

Volveremos a la clase “React1” y añadiremos un import a la clase SPClient:

|  |
| --- |
| import SPClient from './SPClient' |

A continuación, modificaremos el método “handleClick” de modo que modifique el estado con los resultados devueltos por la consulta a SharEPoint:

|  |
| --- |
| handleClick(event) {  SPClient.get(this.\_context, this.props.listname).then((res)=>{  this.setState({items:res});  });  } |

Instanciaremos nuestro componente desde el WebPart en el fichero React1WebPart.ts :

|  |
| --- |
| public render(): void {  const element: React.ReactElement<IReact1Props > = React.createElement(  React1,  {  description: this.properties.description,  listname: "contactos",  context: this.context  }  );  ReactDom.render(element, this.domElement);  } |

Volveremos a probar nuestro WebPart lanzando de nuevo el comando gulp serve, esta vez pulsaremos en el botón “cargar” para comprobar el funcionamiento del evento.

# Referencias

SPFx – SPFx - <https://docs.microsoft.com/en-us/sharepoint/dev/spfx/sharepoint-framework-overview>

Office Fabric UI - <https://developer.microsoft.com/en-us/fabric#/styles/animations>

Connect to SharePoint - <https://docs.microsoft.com/en-us/sharepoint/dev/spfx/web-parts/get-started/connect-to-sharepoint>

PnP JavaScript Core - <https://github.com/SharePoint/PnP-JS-Core>

Ejemplos SPFx - <https://github.com/SharePoint/sp-dev-fx-webparts/tree/master/samples>