

Understand widgets



This help page is for users in Creator 6. If you are in the older version (Creator 5), [click here](#). Know your [Creator version](#).

Widgets are used to extend the capabilities of your Zoho Creator application so that you could perform tasks that could not be accomplished using the in-built features. It equips you with additional features to enhance the front-end capabilities of your app.

Widget capabilities

Widgets enable you to:

- Create and customize features that are specific to your app
- Redefine the way your application interacts with your customers
- Take complete control of the front-end design
- Incorporate third party apps that cater to your Creator app

Create and customize features

There are many features in Zoho Creator that helps you bring your application together. However, there can arise instances where you are unable to find the exact solution to a requirement. Widgets can help you resolve such minor setbacks. You can create a feature, configure it as a widget, and add it to your page to attain the kind of usability you are looking for.

You can create and add your widget in Creator in two ways.

- [Uploading your widget file](#)
- [Importing from Figma](#)

Uploading your widget file

You can customize your widget's appearance and feel and then upload the widget file. To do this, you can inject **JavaScript** code directly into your widget file and incorporate the required interactive and customized functionalities. Creator enables you to upload the widget file from your local device and also supports **drag and drop functionality** in the [page builder](#).



Any direct changes to the widget script and its elements (for example, CSS and JS files) will require re-uploading the modified file for the widget to work as required.

A few points that can be considered before you create your widget file.

- Define how you want your widget to behave in the uploaded file. For example, you might want to trigger specific actions or display certain data.
- Apply CSS styles to make your widget visually appealing and fit the overall design of your application.
- Test your widget and check for errors by uploading various files and ensuring it behaves as expected.



Click [here](#) to know how to create a widget by uploading your file.

Importing from Figma

You can also create widgets in Creator by importing your existing Figma design from Figma UI Kit. This kit comes with comprehensive usage instructions and supported components for Creator widgets. The designed screen can be accessed as a widget by copying the design's [frame URL](#) and [access token](#) and pasting them into Creator's widget builder. This ability to create widgets from Figma files directly helps bridge the gap between design and development and quickly create efficient widgets with minimal effort.



Click [here](#) to know how to create a widget by importing your Figma design file.

Use Cases

Case 1: Uploading your widget file

Let's assume you are an automobile dealer and have an application dashboard for your monthly sales related data. This dashboard has details of the number of vehicles sold, a comparison of the previous month's sales, a ranking of the models sold, the sales teams' individual performance scores, and a customer feedback form. You would need to display a form that is of feedback matrix type. You can configure the form as a widget in your page. Using Java Script, CSS, and HTML, you can create this form, upload it to

Creator, and introduce it into the required page as a widget. You can use Creator API to add, update, edit, and delete records from this report.

Case 2: Importing your Figma design file into Creator

Let's assume that you've created a Contacts Management application in Creator. This application has a Contact Details form, in which your consumers must enter their required data. You may not be able to convey the precise depth and context of your requirements in the conventional form style. However, you can add tabs, modify the dropdown menu, and construct your form in a wizard-style manner using our UI kit in Figma and import the customized design into Creator as a widget. This way you can quickly and effectively establish the form's purpose and context with an enhanced user experience.

Redefine app interface

Widgets boost your control over the front-end design of your application. You will be able to completely redesign your app to suit your requirements and to mirror the uniqueness of your brand. This gives you a mechanism that changes the way your app interacts with your customers.

Let's assume you have your application's landing page designed using pages. But if you want to explore more options to design your interface, widgets can assist you with it. If you have a long form in your application, presenting it in a single page might be too tedious for the user. You can create a progressive form that groups form fields into sections and displays them one after the other in a linear manner using Java Script, CSS, and HTML. Then, upload it as a widget in Creator, and drag and drop it into your page. Your custom page is made available in your application.

Configuring widgets parameters

You can define parameters within a widget to accommodate dynamic values and map them to your application's fields in the [Widget Configuration](#) section. These widget parameters can be specified in the script present in the **manifest.json** file (this file specifies how to launch your application) during the widget creation. This feature enables you to exert greater control over the customization of widget parameters and ensures the seamless integration of the widget with your application interface, thereby enhancing personalization.

You can easily edit the **manifest.json** file in which you can specify the title (name) and app component types.

Possible Key Values	Data type of the value	Character limits	Edit Widget params	Delete Widget params
name	string	50	not possible	not possible
type	as mentioned below	-	not possible	not possible
defaultValue	string	40	possible	possible
placeholder	string	-	yes	possible
help	string	140	possible	possible
mandatory	boolean	-	possible	possible

The components and data values which can be used for configuring the key "type" are [application](#), [form](#), [report](#), [field](#), *string*, *integer*, *float*, and *boolean*.



Note:

- All the 'name' and 'type' keys mentioned above are case sensitive.
- The 'name' & 'type' keys are mandatory for every configuration. These should not be edited or deleted during widget update
- The values for 'name' key should be unique
- Application, Form, Report type should exist only once. Field type should not exist without a Form/Report type
- The total number of configurations should not exceed 20

Upon [adding the widget to the page](#) in the edit mode of the application, the **Widget Configuration** menu will slide in from the right, presenting mappable fields. This enables you to tailor the widget to suit your specific needs.

Let's assume you have created two applications in Creator, an *Employee Management* application and an *HR Management* application. It's essential to maintain and update records in both application dashboards whenever an employee checks in or out of your organization. To streamline this process, a unified widget can be created, and its fields can be configured dynamically using the manifest.json file. This one widget can then be used across the two applications, enabling users to customize field values based on the specific requirements of each application. The **Widget Configuration** pane, which

appears upon adding the widget to a page, provides you with the flexibility to assign values relevant to each application, thereby enhancing the functionality and usability of the applications created within Creator. The following is an example **manifest.json** script which can be used in the above mentioned applications,

```
1. {
2.   "service": "CREATOR",
3.   "cspDomains": {
4.     "connect-src": []
5.   }
6.   "config":
7.   [
8.     {"name": "welcomeText", "type": "string", "defaultValue": "Welcome"},
9.     {"name": "appName", "type": "Application"},
10.    {"name": "formName", "type": "Form", "help": "Provide Employee form details"},
11.    {"name": "reportName", "type": "Report"},
12.    {"name": "fieldName1", "type": "Field"},
13.    {"name": "fieldName2", "type": "Field"},
14.    {"name": "intVal", "type": "integer", "mandatory": true},
15.    {"name": "boolval", "type": "boolean"},
16.    {"name": "floatval", "type": "float"}
17.  ]
18. }
```

JS API to fetch Config in Widget parameters

```
ZOHO.CREATOR.init().then(function(){
  const widgetParams = ZOHO.CREATOR.UTIL.getWidgetParams();
});
```

Incorporate third party apps

Third party apps that cater to your Creator app's requirements can be incorporated into your pages as widgets. Let's assume you have an application for the Online Courses that you offer. The app has registration details, contact details, course details, and webinar schedules. Your page contains course details, registration form, and upcoming webinars. If you have a registered user logged in, you need to display a timer to count down to the start of the relevant webinar. You can incorporate this timer from a third party service or third party javascript plugins into your Creator application as a widget.

Points to remember

- The widgets with JS APIs will not work on the published pages.
- The widget ZIP folder contains the **widget.html** file inside the App folder by default. This is the **index file** i.e., this file will be incorporated as the widget in your page.
- If you've created another folder inside the App folder and moved the widget.html file inside the new folder, then you need to specify the index file name in the following format:

```
/<folder-name>/<filename>.html
```

Limitations

- You can create up to **50 widgets** in your Zoho account.
- For internal hosting,
 - the maximum size of the widget ZIP file must not exceed **10 MB**.
 - the number of files inside the widget ZIP file must not exceed **250**.
 - the maximum size of a file inside the widget ZIP file must not exceed **5 MB**.
- The **widget ZIP file name** must not exceed **100 characters** and can contain **alphanumerics (A-Z, a-z, 0-9), _, \, \$, ., -**.
- The **file names** inside the widget ZIP file must not exceed **50 characters** and can contain **alphanumerics (A-Z, a-z, 0-9), _, ., \$, -**.
- The **folder name** inside the widget ZIP file can contain **alphanumerics (A-Z, a-z, 0-9), _, \$, -**.
- **File types** supported include **.txt, .md, .XML, .dre, .jpg, .jpeg, .png, .gif, .css, .js, .HTML, .json, .mp3, .svg, .woff, .ttf, .eot, .otf, .woff2, .webm, .mp4**.

Create widget by uploading file



This help page is for users in Creator 6. If you are in the older version (Creator 5), [click here](#). Know your [Creator version](#).

Before creating a widget, you need to have the Command Line Interface installed. [Learn more](#)



Note: For internal hosting, the maximum size of the widget ZIP file is 10 MB and the number of files inside the widget Zip file must not exceed 255.

In Creator, you can create a widget in two ways.

- [Uploading your widget file](#)
- [Importing a Figma file](#)

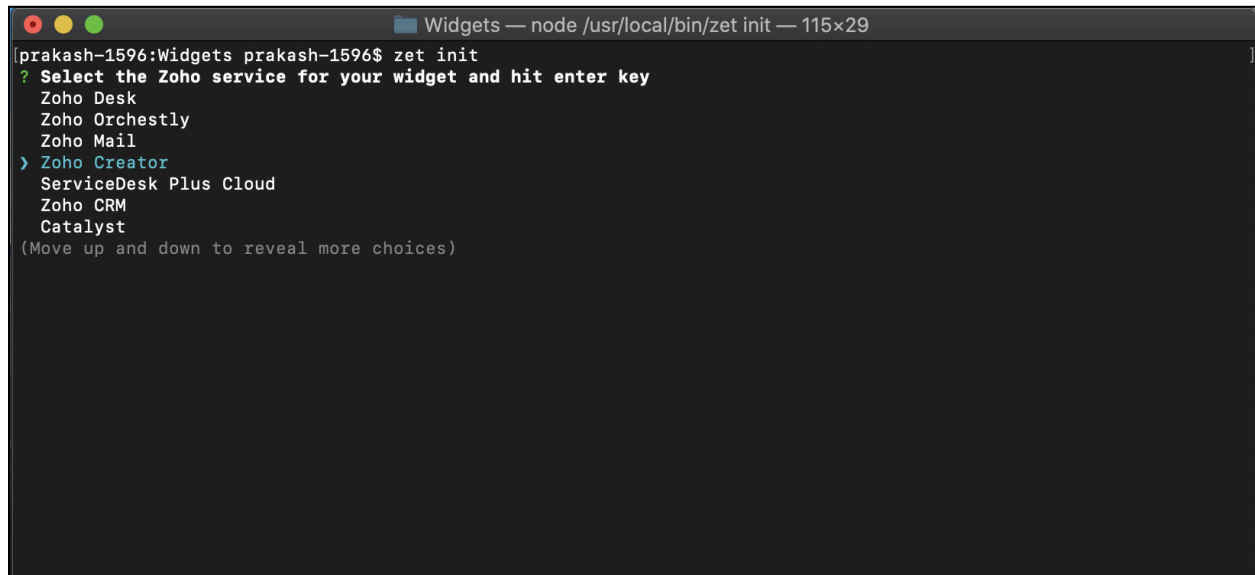
Upload your widget file

1. Open the terminal.
2. Run the following command to create a new project:

```
$ zet init
```

3. A list of *Zoho Services* will be displayed.

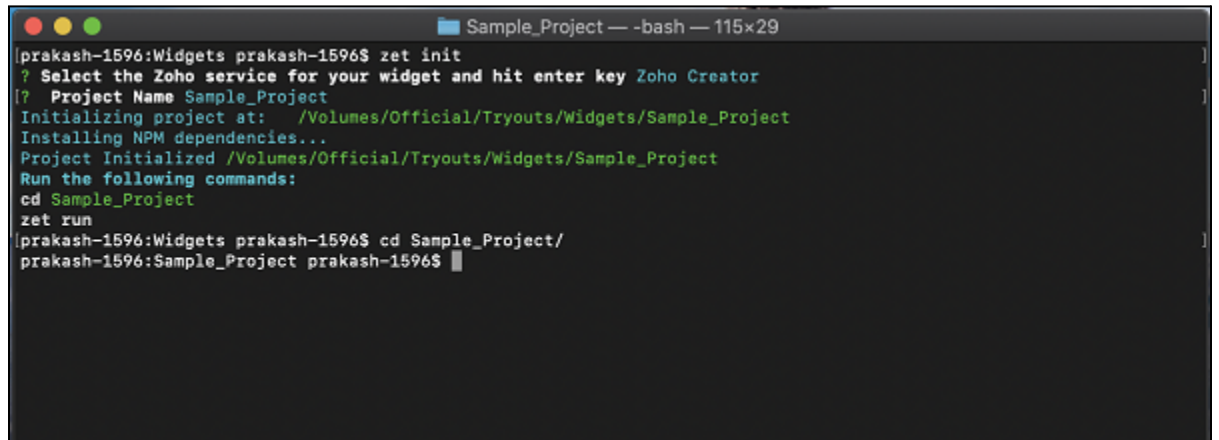
4. Select **Zoho Creator** as the required service.

A terminal window titled 'Widgets — node /usr/local/bin/zet init — 115x29'. The prompt is 'prakash-1596:Widgets prakash-1596\$'. The command 'zet init' has been executed, resulting in the following output:

```
? Select the Zoho service for your widget and hit enter key
Zoho Desk
Zoho Orchestly
Zoho Mail
> Zoho Creator
ServiceDesk Plus Cloud
Zoho CRM
Catalyst
(Move up and down to reveal more choices)
```

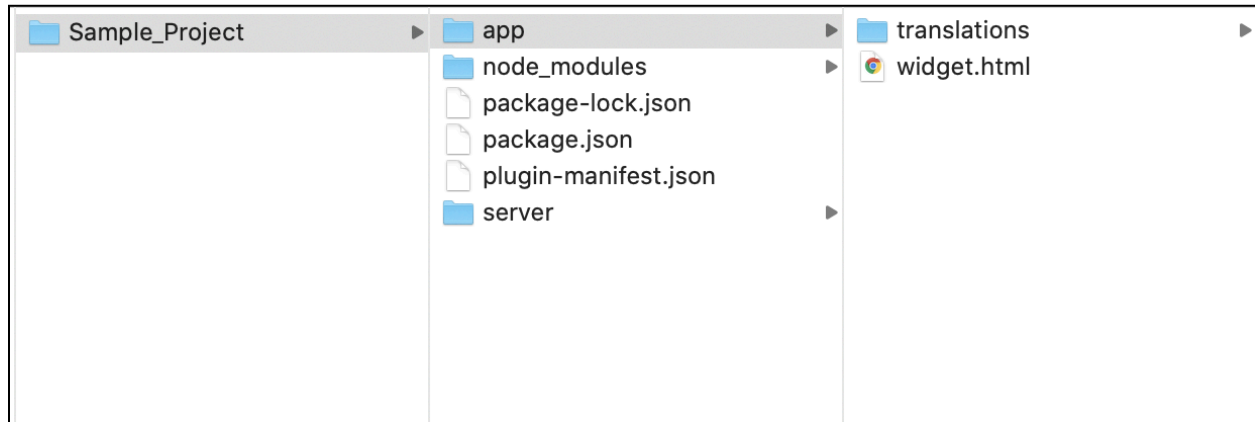
5. Enter the *Project Name*. The sample project will be created in the respective folder directory.
6. Run the following command to open the project folder.

\$ cd {Project_name}

A terminal window titled 'Sample_Project — -bash — 115x29'. The prompt is 'prakash-1596:Widgets prakash-1596\$'. The command 'zet init' has been executed, resulting in the following output:

```
? Select the Zoho service for your widget and hit enter key Zoho Creator
[?] Project Name Sample_Project
Initializing project at: /Volumes/Official/Tryouts/Widgets/Sample_Project
Installing NPM dependencies...
Project Initialized /Volumes/Official/Tryouts/Widgets/Sample_Project
Run the following commands:
cd Sample_Project
zet run
prakash-1596:Widgets prakash-1596$ cd Sample_Project/
prakash-1596:Sample_Project prakash-1596$
```


7. Open the app folder in your project folder.



8. You can configure the **manifest.json** file for your widget by following the below steps:
 - After creating a folder for the project, run the following command to open the project folder.
`$ cd {Project_name}`Open the App folder in your project folder.
 - Find the plugin-manifest.json file in this folder.
 - Configure the file with a name field and type field for application components(application, forms, and reports) to be used in the widget.
9. Find the Widget.html file in the folder. Enter your code in the Widget.html file.
The widget.html file contains the structure, design and components of the Widget. To use the Zoho Creator APIs in the widget, you can use the API helpers provided in the [JS SDK Library](#).

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <script src="js/widgetsdk.js"></script>
    <script type="text/javascript">
      ZOH0.CREATOR.init()
      .then(function(data) {
        var config = {
          viewName : "All_Registrations",
          id : "33461800000711"
        }
        ZOH0.CREATOR.API.getRecordByID(config)
          .then(function(reponse) {
            document.getElementById('title').innerHTML = "Welcome "+reponse.data.Name;
          });
      });
    </script>
  </head>
  <body>
    <h2 id="title">This is a sample Widget built using Zoho Extension toolkit.</h2>
  </body>
</html>

```

10. Run the following command to validate your application:

```
$ zet validate
```

11. This will validate your app package and identify any violations. They should be corrected before updating the zip in the developer console.

12. Run the following command to pack the project.

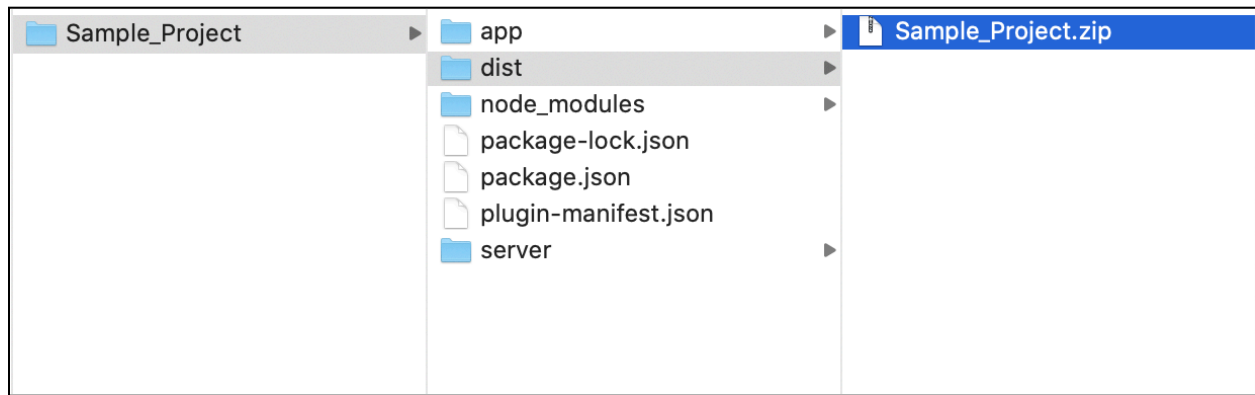
```
$ zet pack
```

```

prakash-1596:Widgets prakash-1596$ zet init
? Select the Zoho service for your widget and hit enter key Zoho Creator
[?] Project Name Sample_Project
Initializing project at: /Volumes/Official/Tryouts/Widgets/Sample_Project
Installing NPM dependencies...
Project Initialized /Volumes/Official/Tryouts/Widgets/Sample_Project
Run the following commands:
cd Sample_Project
zet run
prakash-1596:Widgets prakash-1596$ cd Sample_Project/
prakash-1596:Sample_Project prakash-1596$ zet validate
Validation Rules passed successfully.
prakash-1596:Sample_Project prakash-1596$ zet pack
Packing project assets...
Extension packed successfully as Sample_Project.zip in dist folder.
prakash-1596:Sample_Project prakash-1596$

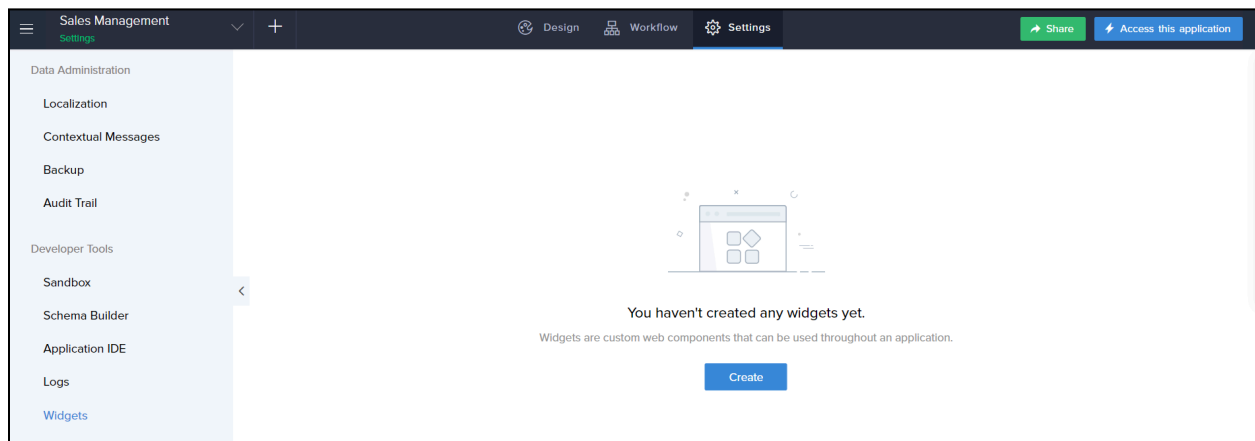
```

13. A zip file of the application will be created in the dist folder of your project directory.

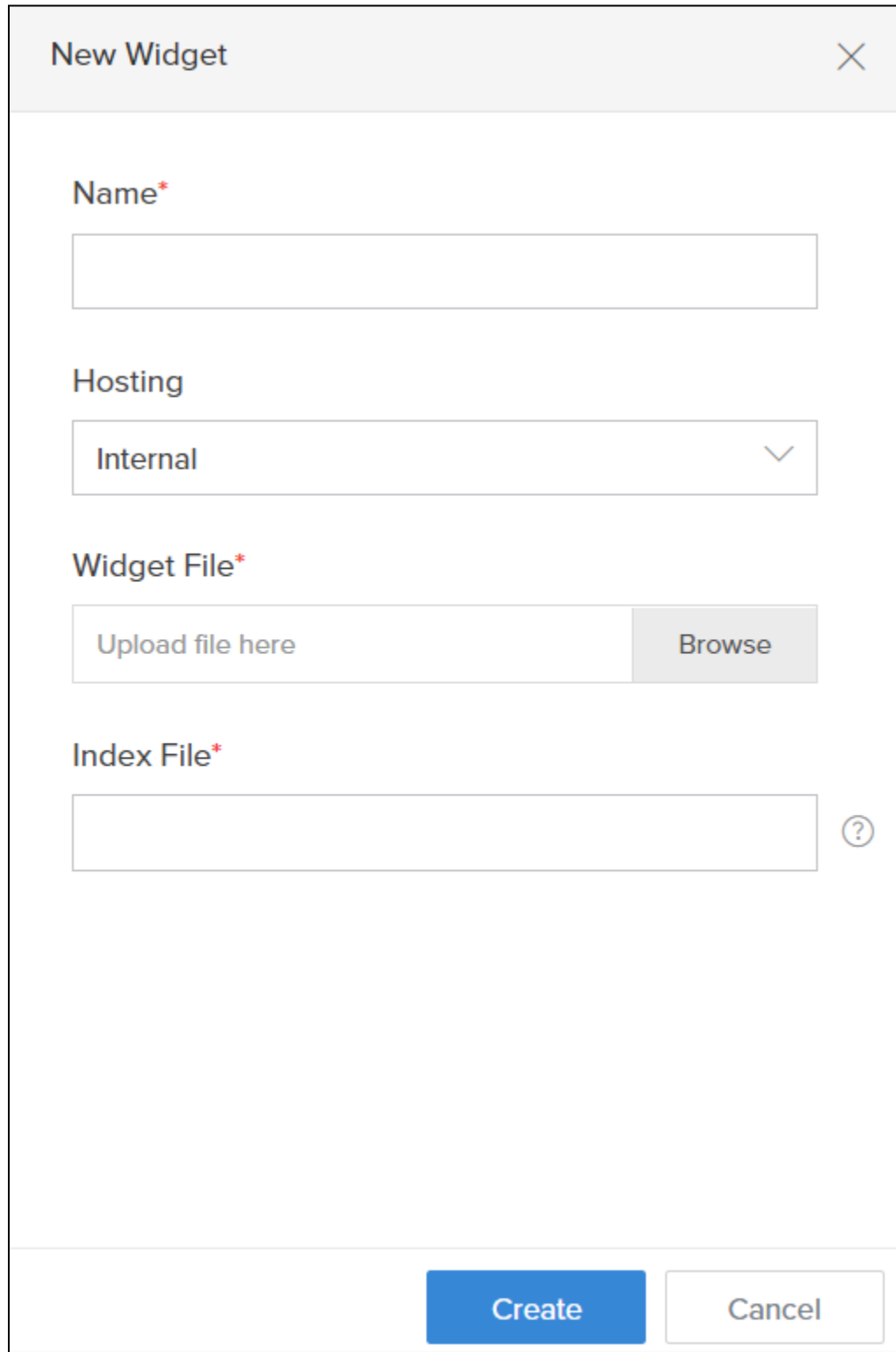


14. Navigate to the **Settings** page of the app in which you need to embed your widget. Select **Widgets**.

15. Click **Create** button.



16. The *New Widget* pane will appear.



The image shows a 'New Widget' dialog box with a title bar containing the text 'New Widget' and a close button (X). The dialog contains four main sections: 'Name*' with a text input field; 'Hosting' with a dropdown menu showing 'Internal'; 'Widget File*' with a text input field containing 'Upload file here' and a 'Browse' button; and 'Index File*' with a text input field and a help icon (?). At the bottom right are 'Create' and 'Cancel' buttons.

New Widget

Name*

Hosting

Internal

Widget File*

Upload file here Browse

Index File*

Create Cancel

17. Enter a name for your **Widget**.

18. Choose **Internal** option in the *Hosting* drop down list.
19. Upload the packed zip file from the **dist** folder in the **Widget File** field.
20. Enter **/widget.html** in the **Index File** field. The /widget.html file is the index file i.e., this file will be incorporated as the widget in your page.

Note:

- The widget ZIP folder contains the **widget.html** file inside the **App** folder by default.
- If you've created another folder inside the **App** folder and moved the widget.html file inside the new folder, then you need to specify the index file name in the following format:

/<folder-name>/<filename>.html

New Widget

Name*

Sample Widget

Hosting

Internal

Widget File*

Sample_Project.zip

Browse

Index File*

/widget.html

?

Create

Cancel

21. Click **Create**. The created widget is added to the **Widgets** page.

22. This widget will appear as a drag-and-drop element in Page builder for all the Pages present in the application.

External Hosting

The procedure given above pertains to the internal hosting of widgets (i.e.) the widgets are hosted within Creator. Alternatively the widgets can also be hosted externally and linked up with Creator.

The procedure for external hosting of widgets is as follows:

1. Go to the *Application Settings* page. Then click on **Widgets** option.
2. Now click on the **New Widget** button in the Widgets page. The New Widget pane will appear.

New Widget

Name*

Hosting

Internal

Widget File*

Upload file here

Browse

Index File*

?

Create

Cancel

3. Enter a name for your Widget.
4. Choose **External** option in the *Hosting* drop down list.

5. Enter the Index page URL of the externally hosted widget in the **Index File** field.

Install command line interface



This help page is for users in Creator 6. If you are in the older version (Creator 5), [click here](#). Know your [Creator version](#).

Zet is a command line interface that helps developers build and package widgets in Zoho Creator. The Command Line Interface, or CLI, is used to send commands to a software program as lines of texts in order to interact with it. This interaction includes typing commands into the interface and receiving responses to them. This text-based interface has its basis in the Graphical User Interface (GUI) but provides a mechanism that is easier to use. Developers use CLIs to create applications, software, and operating systems.

Zoho Creator enables you to create and integrate a widget using CLI.

Prerequisites to install CLI

- Download the nodejs source code from <https://nodejs.org/en/download/>
- Use the following command to verify the installation:

```
$ node -v
```

```
$ npm -v
```

Install CLI

1. Run the following command to install the zapps cli node package:

```
$ npm install -g zoho-extension-toolkit
```

2. Run the following command to ensure that the installation is successful:

```
$ zet // Help information about 'zet' command will be shown.
```

Create a project

1. Run the following command to create new project:

```
$ zet init
```

The list of Zoho Services you need to create a project template for will appear.

2. Select **Zoho Creator** and create a Project Folder.
3. Include resources: All the files that are required for rendering your widget will appear inside the "app" folder of your project.

Start server

1. Run the following command to start a local HTTP server that allows you to run your app locally and to test it in your sandbox instance:

```
$ zet run
```

This will run the HTTP server in your local machine in port number 5000. The port should not be occupied with any other process before starting the server.

2. To verify whether the server has started successfully, you need to open the following URL in your web browser: <http://127.0.0.1:5000/app/widget.html>

Validate and package the application

1. Run the following command to validate your application:

```
$ zet validate
```

This will validate your app package and identify any violations. They should be corrected before updating the zip in the developer console.

2. Run the following command to generate a zip of your application that can be uploaded:

```
$ zet pack
```

This will create a zip file of the application in the "dist" folder of your project directory. This zip file needs to be uploaded to the Zoho Creator.

Whitelisting of URLs



This help page is for users in Creator 6. If you are in the older version (Creator 5), [click here](#). Know your [Creator version](#).

By default, Widgets are allowed access only the Creator servers and subsequently all other URLs are blocked. To use any client library within widget the whitelisting of URLs should be done. Whitelisting is the process by which you can create a list of URLs that the widget can access apart from the default Creator URLs.

To whitelist URLs:

1. Install the Command Line Interface(CLI). [Learn more](#)
2. Open your project folder. Find the *plugin-manifest.json* file in the folder.

3. Add the Domains to be whitelisted in the "plugin-manifest.json" file.
4. Pack the ZIP using the CLI client. [Learn more](#)
5. Upload this ZIP to Creator.

Syntax

```
{  
  
  "service": "CREATOR",  
  
  "cspDomains": {  
  
    "connect-src": [  
  
      "domain1",  
  
      "domain2",  
  
      "domain3"  
  
    ]  
  
  }  
  
}
```

Example

```
{  
  
  "service": "CREATOR",  
  
  "cspDomains": {  
  
    "connect-src": [  
  

```

```
"wss://*.zohopublic.com",  
  
"https://*.zoho.com",  
  
"https://*.zohopublic.com"  
  
]  
  
}  
  
}
```

JS API documentation



This help page is for users in Creator 6. If you are in the older version (Creator 5), [click here](#) . Know your Creator [version](#) .

Download the latest JS SDK library from the CDN URL given below and link it using the script tag.

- CDN URL - <https://js.zohostatic.com/creator/widgets/version/1.0/widgetsdk-min.js>

Initialize the SDK using the below method and use then block to write the logic

```
1. ZOHO.CREATOR.init()  
2.     .then(function(data) {  
3.         //Code goes here
```

```
4.    });
```

To get the value from the parameters configured in pages, use the below method

```
1. var queryParams = ZOHO.CREATOR.UTIL.getQueryParams();
```

To get the login user's email address, use the below method

```
1. var initparams = ZOHO.CREATOR.UTIL.getInitParams();
```

Sample response:

```
1. {  
2.   "scope": "zylkercorp",  
3.   "envUrlFragment": "/environment/development",  
4.   "appLinkName": "widgetapp",  
5.   "loginUser": " john@zylker.com "  
6. }
```

The values of the image field will be an API endpoint in case the image is uploaded from computer or phone, use the below method to convert the endpoint to image data and assign the value to the source of the image tag. In another case, the image can be a public image link, and this method works for both cases.

```
1. var imageTag = document.getElementById("imgtag");  
2. var imageURL =  
   "/api/v2/zylker/Contacts/view/All_Contacts/66359000000015039/Upload/download";
```

```
3. ZOHO.CREATOR.UTIL.setImageData(imageTag, imageURL,  
  callbackFunction);
```

Widget API allows customers to **push** and **pull** data from other apps in their Creator account. The success of the API call to push or pull data in other apps depends on the [permissions](#) provided for users of that app.

Record APIs

These APIs help you perform different tasks with the records of the application. The various Record APIs are as follows:

Note : *appName* needs to be passed only when you need to push or pull data from other apps in your account.

- [Add Record](#)
- [Update Record](#)
- [Get All Records](#)
- [Get Specific Record Using ID](#)
- [Get Record Count](#)
- [Delete Record](#)
- [Upload File](#)

Add Record

Method : `addRecord`

Name	Description

formName	Link name of the form to which the records to be added. Refer this page to view and change the link name.
data	Form data in json format, where the format for all the fields has been specified here.
appName	Link name of the application from which data needs to be pushed to.

Syntax :

```

1. //configuration json
2. config = {
3.     appName : "${APPNAME}",
4.     formName : "${FORMNAME}",
5.     data : "${FORMDATA}"
6. }
7. //add record API
8. ZOHO.CREATOR.API.addRecord(config).then(function(response){
9. //callback block
10. });

```

Example :

```

1. //configuration json

```



```
2. var config = {
3.     appName : "zylker",
4.     formName : "timesheet",
5.     data : formData
6. }
7. //add record API
8. ZOHO.CREATOR.API.addRecord(config).then(function(response)
9. {
10.     if(response.code == 3000){
11.         console.log("Record added successfully");
12.     }
13. });
```

Form data :

```
1. formData = {
2.     "data" : {
3.         "Name": {
4.             "first_name": "James",
5.             "last_name": "Kyle",
6.         },
7.         "Address": {
8.             "address_line_1": "Dari Mart",
9.             "address_line_2": "Hayden Bridge Rd",
```

```
10.         "district_city" : "Springfield",
11.         "state_province" : "Oregon",
12.         "postal_code" : "97477",
13.         "country" : "United States"
14.     },
15.     "Rich_Text" : "The phone number is
    <b>+1541-726-4051</b>",
16.     "Phone_Number": "+15417264051",
17.     "Single_Line": "A simple one line text",
18.     "Email": " james@zylker.com ",
19.     "Dropdown" : "Open",
20.     "Radio" : "Closed",
21.     "Multi_Select" : ["New", "Used"],
22.     "Checkbox" : ["Refurbished", "Used"],
23.     "Number": "1000",
24.     "Decimal" : "1000.00",
25.     "Percent": "60.00",
26.     "Date_field" : "10-Jan-2020",
27.     "Date_Time" : "23-Mar-2020 12:25:43 PM",
28.     "Image" : "
    https://www.zylker.com/sites/default/profile-01.jpg",
29.     "Ur1": {
30.         "value": "Dari Mart",
```

```
31.         "url": " www.darimart.com",
32.         "title": "Dari mart provisional store"
33.     },
34.     "Lookup_Single_Select" : "521270000078464",
35.     "Lookup_Multi_Select" :
36.         ["5212750000247932", "5212750000564867"],
37.     "SubForm" : [
38.         {
39.             "Single_Line" : "Service Equipment",
40.             "Dropdown" : "New Order"
41.         },
42.         {
43.             "Single_Line" : "Rotary Machine",
44.             "Dropdown" : "Replacement",
45.         }
46.     ]
47. }
```

Response :

```
1. response = {
2.     "code": 3000,
3.     "data": {
```

```
4.      "ID": "2000000245119"
5.    },
6.    "message": "success"
7.  }
```

Update Record

Method : updateRecord

Name	Description
reportName	Link name of the report through which the record to be updated. Refer here to get and update the link name of a report
id	ID value of the record to be updated
data	Form data in json format, where format for all the fields have been specified here
appName	Link name of the application in which data needs to be updated.

Syntax :

```
1. //configuration json
2. config = {
3.     appName : "${APPNAME}",
```

```
4.   reportName : "${REPORTNAME}",
5.   id: "${RECORDID}",
6.   data : "${FORMDATA}"
7. }
8.
9. //update record API
10. ZOHO.CREATOR.API.updateRecord(config).then(function(response){
11. //callback block
12. });
```

Example :

```
1. var config = {
2.   appName : "zylker"
3.   reportname : "All-Timesheet",
4.   id : "2000000245119"
5.   data : formData
6. }
7.
8. ZOHO.CREATOR.API.updateRecord(config).then(function(response){
9.   if(response.code == 3000){
10.     console.log("Record updated successfully");
11.   }
12. });
```

Response :

```
1. response = {
2.   "code": 3000,
```

```
3. "data": {
4.   "ID": "2000000245119"
5. },
6. "message": "success"
7. }
```

Get All Records

Method : `getAllRecords`

Configuration :

Name	Description
reportName	Link name of the report through which the record to be updated. Refer here to get and update the link name of a report
criteria (optional)	Condition to filter the record, which is applied along with the condition specified in the report filter . The criteria must be specified in the following format <code>is the link name of the field, which compares with the and the comparison is based on the . The supported operators are !=, <=, >=, <, >, ==</code> You can combine more than one criteria using logical AND (&&) and logical OR() Example : (Stage == "Open" && Score > 40)
page (optional)	Index or number of the page to be retrieved. The default page number is 1

pageSize (optional)	Total number of records to be retrieved in the specified page. The default size is 100 and the maximum size is 200
appName	Link name of the application from which data needs to be pulled from.

Note :

- The response will return the data that's displayed in both the Quick view and Detail view of the corresponding report
- When the Detail view sports a [related block](#) of data, which is related via a [lookup](#) field, this API's response will also include the data from that lookup field's [display fields](#).

Syntax :

```

1. //configuration json
2. config = {
3.     appName : "${APPNAME}",
4.     reportName : "${REPORTNAME}",
5.     criteria: "${CRITERIA}",
6.     page : "${PAGENUMBER}",
7.     pageSize : "${PAGESIZE}"
8. }
9.
10. //get all records API
11. ZOHO.CREATOR.API.getAllRecords(config).then(function(response){
12.     //callback block

```

```
13. });
```

Example :

```
1. var config = {
2.   appName : "zylker"
3.   reportName : "All-Timesheet",
4.   criteria : "(Status == \"Open\" && Priority = \"High\")",
5.   page : 1,
6.   pageSize : 10
7. }
8.
9. ZOHO.CREATOR.API.getAllRecords(config).then(function(response){
10.   var recordArr = response.data;
11.   for(var index in recordArr){
12.     console.log(recordArr[index]);
13.   }
14. });
```

Response :

```
1. response = {
2.   "code": 3000,
3.   "data": [
4.     {
5.       "ID": "52129000000146011",
6.       "Name": {
7.         "first_name": "James",
8.         "last_name": "Kyle",
9.         "display_value": "James Kyle"
```



```
10.   },
11.   "Address": {
12.     "address_line_1": "Dari Mart",
13.     "address_line_2": "995 Hayden Bridge Rd",
14.     "district_city": "Springfield",
15.     "state_province": "Oregon",
16.     "postal_code": "97477",
17.     "country": "United States",
18.     "latitude": "33.9307727",
19.     "longitude": "-116.8767541",
20.     "display_value": "Dari Mart, 995 Hayden Bridge Rd, Springfield, Oregon, United
    States"
21.   },
22.   "Rich_Text": "The phone number is <b>+1541-726-4051</b>",
23.   "Phone_Number": "+15417264051",
24.   "Single_Line": "A simple one line text",
25.   "Email": " james@zylker.com ",
26.   "Number": "23",
27.   "Decimal": "1000.00",
28.   "Percent": "60.00",
29.   "Date_field": "10-Jan-2020",
30.   "Date_Time": "23-Mar-2020 12:25:43 PM",
31.   "Dropdown": "Open",
32.   "Radio": "Closed",
33.   "Multi_Select": [
34.     "New",
35.     "Used"
36.   ],
```

```
37.   "Checkbox": [  
38.     "Refurbished",  
39.     "Used"  
40.   ],  
41.   "Image": " https://www.zylker.com/sites/default/profile-01.jpg",  
42.   "File_Upload":  
      "/api/v2/zylker/Contacts/view/Contacts/635900380/Upload/download",  
43.   "Ur1": {  
44.     "value": "Dari Mart",  
45.     "url": " www.darimart.com",  
46.     "title": "Dari mart provisional store"  
47.   },  
48.   "Lookup_Single_Select": {  
49.     "display_value": "James - 4269",  
50.     "ID": "521270000078464"  
51.   },  
52.   "Lookup_Multi_Select": [  
53.     {  
54.       "display_value": "Door Step Delivery",  
55.       "ID": "5212750000247932"  
56.     },  
57.     {  
58.       "display_value": "Cash on Delivery",  
59.       "ID": "5212750000564867"  
60.     }  
61.   ],  
62.   "Subform": [  
63.     {
```

```
64.     "display_value": "Service Equipment,New Order",
65.     "ID": "5212750000433263"
66. },
67. {
68.     "display_value": "Rotary Machine,Replacement",
69.     "ID": "5212750000090778"
70. }
71. ]
72. },
73. {
74.     //Record 2
75. },
76. {
77.     //Record 3
78. }
79. ]
80. }
```

Get Specific Record Using ID

Method : `getRecordById`

Configuration :

Name	Description

reportName	Link name of the report through which the record to be updated. Refer here to get and update the link name of a report
id	ID value of the record to be retrieved
appName	Link name of the application from which data needs to be pulled from.

Note:

- The response will return the data that's displayed in both the Quick view and Detail view of the corresponding report
- When the Detail view sports a [related block](#) of data, which is related via a [lookup](#) field, this API's response will also include the data from that lookup field's [display fields](#).

Syntax :

```

1. //configuration json
2. config = {
3.     appName : "${APPNAME}",
4.     reportName : "${REPORTNAME}",
5.     id: "${RECORDID}"
6. }
7.
8. //get specific record API
9. ZOHO.CREATOR.API.getRecordById(config).then(function(response){
10.     //callback block

```

```
11. });
```

Example :

```
1. var config = {  
2.     appName : "zylker"  
3.     reportName : "All-Timesheet",  
4.     id : "2000000193171"  
5. }  
6.  
7. ZOHO.CREATOR.API.getRecordById(config).then(function(response){  
8.     console.log(response.data);  
9. });
```

Response :

```
1. response = {  
2.     "code": 3000,  
3.     "data": {  
4.         "ID": "52129000000146011",  
5.         "Name": {  
6.             "first_name": "James",  
7.             "last_name": "Kyle",  
8.             "display_value": "James Kyle"  
9.         },  
10.    "Address": {  
11.        "address_line_1": "Dari Mart",  
12.        "address_line_2": "995 Hayden Bridge Rd",
```

```
13.         "district_city" : "Springfield",
14.         "state_province" : "Oregon",
15.         "postal_code" : "97477",
16.         "country" : "United States"
17. "latitude": "33.9307727",
18. "longitude": "-116.8767541",
19. "display_value": "Dari Mart, 995 Hayden Bridge Rd, Springfield, Oregon, United
    States"
20. },
21. "Rich_Text" : "The phone number is <b>+1541-726-4051</b>",
22.     "Phone_Number": "+15417264051",
23.     "Single_Line": "A simple one line text",
24.     "Email": " james@zylker.com ",
25. "Number": "23",
26.     "Decimal" : "1000.00",
27. "Percent": "60.00",
28.     "Date_field" : "10-Jan-2020",
29.     "Date_Time" : "23-Mar-2020 12:25:43 PM",
30. "Dropdown" : "Open",
31.     "Radio" : "Closed",
32.     "Multi_Select" : ["New", "Used"],
33.     "Checkbox" : ["Refurbished", "Used"],
34.     "Image" : " https://www.zylker.com/sites/default/profile-01.jpg",
35.     "File_Upload" :
        "/api/v2/zylker/Contacts/view/Contacts/635900380/Upload/download",
36. "Ur1": {
37.     "value": "Dari Mart",
38.     "url": " www.darimart.com",
```

```
39.         "title": "Dari mart provisional store"
40.     },
41.     "Lookup_Single_Select" : {
42.         "display_value" : "James - 4269",
43.         "ID" : "521270000078464"
44.     },
45. },
46. "Lookup_Multi_Select" : [{
47.     "display_value" : "Door Step Delivery",
48.     "ID" : "5212750000247932"
49. },
50. {
51.     "display_value" : "Cash on Delivery",
52.     "ID" : "5212750000564867"
53. }],
54. "Subform" : [{
55.     "display_value" : "Service Equipment,New Order",
56.     "ID" : "5212750000433263"
57. },
58. {
59.     "display_value" : "Rotary Machine,Replacement",
60.     "ID" : "5212750000090778"
61. }]
62. }
63. }
```

Get Record Count

Method : `getRecordCount`

Configuration :

Name	Description
reportName	Link name of the report for which the record count is required. Refer here to get and update the link name of a report
appName	Link name of the application from which data needs to be pulled from and counted.
criteria (optional)	Condition to filter the record, which is applied along with the condition specified in the report filter . The criteria must be specified in the following format is the link name of the field, which compares with the and the comparison is based on the . The supported operators are !=, <=, >=, <, >, == You can combine more than one criteria using logical AND (&&) and logical OR() Example : (Stage == \"Open\" && Score > 40)

Syntax :

```
1. //configuration json
2. config = {
3.   appName: "first-application",
4.   reportName: "Form_Report",
5.   criteria: "(Single_Line = \"High\")" //optional
6. }
7.
8. ZOHO.CREATOR.API.getRecordCount(config).then(function (response) {
```



```
9.    //your code goes here.  
10.});
```

Example:

```
1. var config = {  
2.     appName : "zylker"  
3.     reportName : "All-Timesheet",  
4. }  
5.  
6. ZOHO.CREATOR.API.getRecordCount(config).then(function (response){  
7.     console.log(response);  
8. });
```

Response:

```
1. response =  
2. {  
3.     "result": {  
4.         "records_count": "4"  
5.     },  
6.     "code": 3000  
7. }
```

Delete Record

Method : deleteRecord

Configuration :

Name	Description
reportName	Link name of the report through which the record to be updated. Refer here to get and update the link name of a report
criteria	<p>Condition to filter the record, which is applied along with the condition specified in the report filter. The criteria must be specified in the following format</p> <p><i><Field Name> <Operator> <Value></i></p> <p><Field name> is the link name of the field, which compares with the<Value> and the comparison is based on the <Operator>. The supported operators are !=, <=, >=,<, >, ==</p> <p>You can combine more than one criteria using logical AND (&&) and logical OR()</p> <p>Example : (Stage == "Open" && Score > 40)</p>
appName	Link name of the application from which data needs to be deleted.

Note: The maximum of 200 records can be deleted for every API call and you will receive error when the number of records exceeds 200.

Syntax :

```
1. //configuration json
2. config = {
3.     appName : "${APPNAME}",
4.     reportName : "${REPORTNAME}",
5.     criteria: "${CRITERIA}"
6. }
7.
8. //delete record API
9. ZOHO.CREATOR.API.deleteRecord(config).then(function(response){
10.     //callback block
11. });
```

Example :

```
1. var config = {
2.     appName : "zylker"
3.     reportName : "All-Timesheet",
4.     criteria : "(Status == \"Invalid\" && Date '01-01-2019')\"
5. }
6.
7. ZOHO.CREATOR.API.deleteRecord(config).then(function(response){
8.     console.log("Record has been deleted");
9. });
```

Response :

```
1. response = {
2.     "result": [{
```

```
3.     "code": 3000,
4.     "data": {
5.         "ID": "66359000000021016"
6.     },
7.     "message": "success"
8. },
9. {
10.    "code": 3000,
11.    "data": {
12.        "ID": "66359000000024309"
13.    },
14.    "message": "success"
15. }
16. ],
17. "code": 3000
18. }
```

Upload File

Method : uploadFile

Configuration :

Name	Description
reportName	Link name of the report through which the record to be updated. Refer here to get and update the link name of a report

id	ID value of the record where the file to be uploaded
fieldName	<p>Link name of the field to which the file has to be uploaded. Refer here to get the link name of the field. The allowed field types are Image, Signature, File Upload, Audio, and Video.</p> <p>In case the file has to be uploaded to a field which has been added inside a subform, the fieldName will be subform field link name followed by the field link name of the field to be uploaded with (dot) in between. Example: \${SUBFORMFIELDNAME.FIELDNAME}</p>
file	It is a javascript File object to be uploaded. Refer here to know more about javascript File API
parentId	In case the file has to be uploaded to a field which has been added inside a subform, then the parentId is the ID value of the parent record, i.e the main form record in which the subform has been embedded.
appName	Link name of the application in which data needs to be uploaded.



Note: The file can be uploaded only after submitting a record and this API will upload one file to one field at a time. In case of more number of fields in a form, this API need to call for each field separately.

Syntax :

```
1. //configuration json
2. config = {
3.     appName : "${APPNAME}",
4.     reportName : "${REPORTNAME}",
5.     id: "${RECORDID}",
6.     fieldName : "${SUBFORMFIELDNAME.FIELDNAME}",
7.     parentId : "${RECORDID}",
8.     file : "${FILEOBJECT}"
9. }
10.
11. //upload file API
12. ZOHO.CREATOR.API.uploadFile(config).then(function(response){
13. //callback block
14. });
```

Example:

```
1. var fileObject = document.getElementById("fileInput").files[0];
2.
3. var config = {
4.     appName : "zylker"
5.     reportName : "All-Timesheet",
```

```
6.   id : "2000000193171",
7.   fieldName : "Attachments",
8.   file : fileObject
9. }
10.
11. ZOHO.CREATOR.API.uploadFile(config).then(function(response){
12. console.log("File uploaded successfully");
13. });
```

Response :

```
1. response = {
2.   "code": "3000",
3.   "data": {
4.     "filename": "Rental Agreement-March.pdf",
5.     "filepath": "1587402896598_Rental_Agreement-March.pdf",
6.     "message": "success"
7.   }
8. }
```

Understand navigateParentURL() method



This method is applicable to [widgets](#).

Widgets are used to extend the capabilities of your Zoho Creator application, helping you perform tasks that couldn't be accomplished using the built-in features, as well as enhance the front-end capabilities of your app. You can create a feature, configure it as a [widget](#), and add it to your [page](#) to attain the kind of usability you are looking for.

You can also incorporate third-party apps that cater to your Creator app's requirements in your pages as widgets. In these widgets, you might want to configure certain **navigation actions** to be performed upon clicking any of the buttons. For example, upon clicking the **Previous** button, users must be redirected to the previous URL; upon clicking the **Cancel** button, the widget popup must be closed while users remain in the parent page. These actions will be executed when the domains of both the parent and child windows are the same. Currently, if the parent and child windows operate in different domains, then the respective actions will not be executed due to cross-domain restrictions.

With the **navigateParentURL()** method, you can perform an action in the **parent window** from the **child window** itself. In other words, this method can be used to control only the parent container and the actions executed for the parent URL.



How to identify parent window and child windows?

The parent window is the main window on which your user has currently landed and will perform any operation. For example, the Dashboard page in your application is the parent window. All the windows that open inside your main window are the child windows. For example, an Email Subscribe popup that opens inside your Dashboard page is a child window.



Note: There can be multiple child windows in your parent window.

Let's say you've created an application named *Food Delivery Management*, in which you've built a form titled *User Details*. Users will enter their details and, upon checking the **Verify Details** field (decision box), a *Verify Phone Number* popup will open. This is actually a third-party service incorporated as a widget. Users can either enter their phone number and click **Verify** or close the popup.

User Details

The screenshot displays a web application interface for 'User Details'. The top navigation bar includes 'Quiz Management' and 'User Details'. The left sidebar lists various subjects: 'Subject Dashboard', 'Python', 'Java', 'Java Scripts', 'jQuery', and 'User Details' (which is highlighted). The main content area contains a form with the following fields: 'Full Name' (split into 'First Name' and 'Last Name'), 'Email', 'Phone' (with a '+91' country code dropdown), and 'Address' (split into 'Address Line 1' and 'Address Line 2'). Below the address fields are 'City / District', 'State / Province', 'Postal Code', and 'Country' dropdowns. A dark grey popup window is overlaid on the form, containing the text 'VERIFY PHONE NUMBER' and 'CONFIRM ADDRESS'. Below the form are 'Submit' and 'Reset' buttons. At the bottom right, there are two mobile device screens: one showing a phone verification screen with the text 'Please verify your phone number.' and a 'VERIFY' button, and another showing a map for location selection with the text 'Select delivery location' and a 'Confirm location' button.

After closing the popup window, we need to explicitly switch back to the parent window (here, its a form). In this case, since we've used a third-party service that belongs to a different domain, the close action will not be executed. This is because widgets cannot

execute URL actions when the parent and child windows operate in different domains. This is where **navigateParentURL()** method comes into the picture.

Syntax

```
1. var param = {  
2.   action:${ACTIONVALUE}  
3.   url: ${URL}  
4.   window: ${WINDOWTYPE}  
5. }  
6. ZOHO.CREATOR.UTIL.navigateParentURL(param)
```



Note: This method is **case-sensitive**. In case of two or more words as its value, camel case is accepted.

For example, the above syntax can be used as follows.

To open a URL

```
1. ZOHO.CREATOR.init().then(function (data) {  
2.   var param = {  
3.     action: "open",url: "https://www.zoho.com/widgets.html", window: "new"  };  
4.   ZOHO.CREATOR.UTIL.navigateParentURL(param);  
5. });
```

To close a URL

```
1. ZOHO.CREATOR.init().then(function (data) {  
2.   var param = {action: "close" };  
3.   ZOHO.CREATOR.UTIL.navigateParentURL(param);  
4. });
```

Label	Value
Action	<ul style="list-style-type: none">• open - to open the specified URL in new/parent window . If values are not passed for 'url' or 'window' parameters, then the specified action will not be executed.• reload - To load the previously visited page.• back - To reloads the parent window.• close - To close the last opened parent container.• close all - To close all open dialog parent containers in the parent window.

URL

You can specify any URL link here. Also, you can include Creator component URIs appended to hash (#).

Example: #Report: Report_Name

Example:

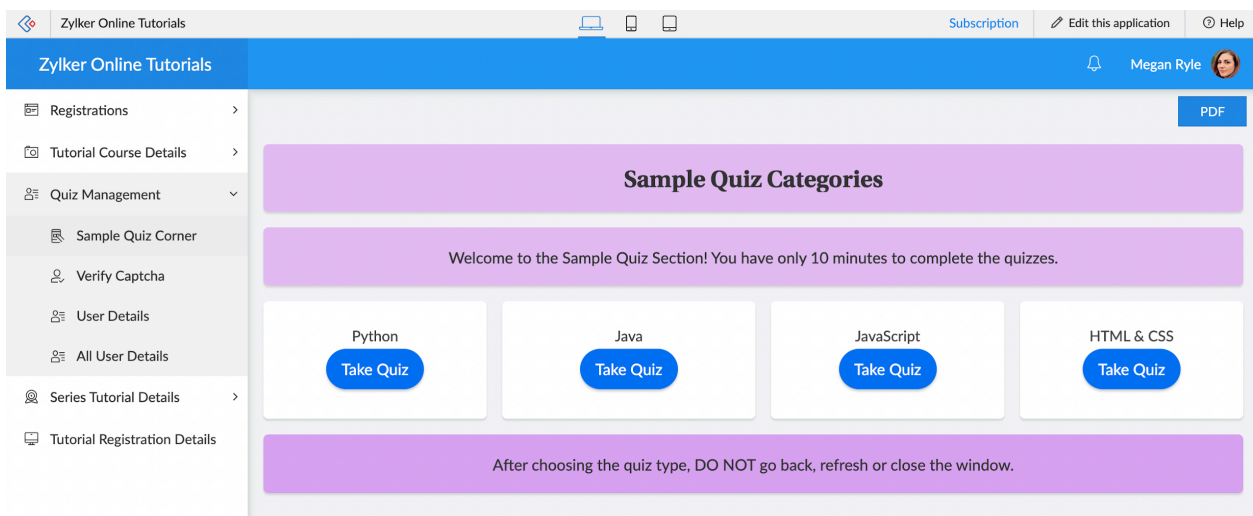
https://www.google.co.in/?client=safari&channel=mac_bm

Window type

"new" , "same"

Example

Let's assume you've created a quiz management app named **Zylker Tutorials**, in which users can attend quizzes on topics of their choice.



Once a user completes the quiz and clicks the **Next** button, they'll be taken to the *Verify Captcha* page that contains a widget. This third-party widget is configured to display images from which users need to check the relevant options to prove that the quiz answers haven't been automated (entered by a robot). If the wrong option is checked, an error will be displayed on the same page. After checking the correct option, users will be redirected from the widget to the *User Details* form. This redirection is achieved using the **navigateParentURL()** method configured to perform an open action (opens a form). Upon entering the required details, users can submit the form.



Info: Additionally, you can add a decision box field named **Captcha Verified** and enable it to be visible only to the users. You can configure a workflow to run a check and ensure that the entered email address matches with the logged-in user's email address. Upon verification, the Captcha Verified field's value can be set to true.

To learn how to configure the above use case, [click here](#).

Limitations

Currently, while specifying the window type, **popup window** is not supported.

Configure navigateParentURL() method

- To learn more about the navigateParentURL() method, click [here](#).
- This method can be configured in page elements such as [widgets](#).
- This method is applicable to both C5 and C6 versions of Creator. Click [here](#) to find your current version.

To use navigateParentURL() method in a widget

- [Create a widget](#)
- [Add widget to a page](#)

Create a widget

1. Download and install **Node.js** source code from this link <https://nodejs.org/en/download/>
2. Run the following commands to confirm that the installation was successful. The commands return the version of the package.



Note: When entering commands, there is no need to enter the dollar sign (\$) explicitly; it will already be present by default. Commands can be run in Terminal for Mac OS and in Command Prompt for Windows.

1. \$ node -v
2. \$ npm -v

3. Run the following command to install the [zapps cli node package](#).

1. \$ npm install -g zoho-extension-toolkit



Note: Use the following command in the event of any permission issues:

```
sudo npm install -g zoho-extension-toolkit
```

4. Run the following command to confirm that the installation was successful. It will return the version of the package.

```
1. $ zet -v
```

5. Create a project using the following command:

```
1. $ zet init
```



Note: Use the following command in the event of permission issues:

```
sudo zet init
```

6. Select **Zoho Creator**.

7. Specify a project name. Hit **Enter**.



Note:

The project is basically a folder which is used to store files that are required to render the widget. It can contain only alphanumeric and underscore characters.

8. Type `cd <project/folder_name>` and hit **Enter** to navigate to the project directory.

9. Run the following command.

```
1. $ zet run
```



Note: This will run the HTTP server in your local machine in port number 5000. The port should not be occupied with any other process before starting the server. A relevant note may appear as a result of the above command.

10. To verify if the server has started successfully, navigate to *widget.html* and open it in a browser. The *widget.html* file will be present in the folder (project) created earlier.



Note:

By default, this file will be present in the App folder.

You can also access the *widget.html* file by following the below steps:

1. Open the local host URL (<https://127.0.0.1:5000>) in a new tab and authorize the connection by clicking **Advanced->Proceed to 127.0.0.1**.

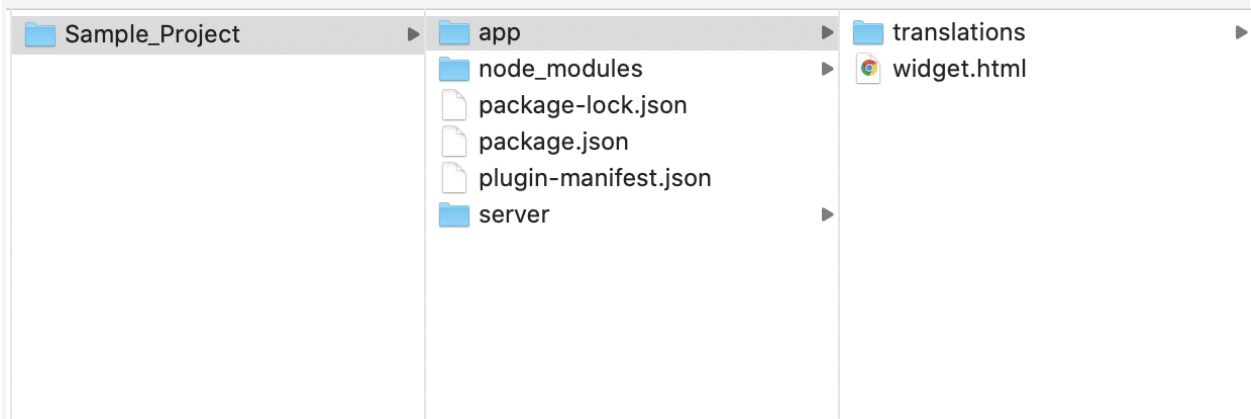
2. Append **/app/widget.html** to the local host URL as follows:

<https://127.0.0.1:5000/app/widget.html>

You should see the following screen:

This is a sample Widget built using Zoho Extension toolkit.

11. Open the app folder in your project folder. Find the *widget.html* file in the same folder.



12. Enter your code in the *widget.html* file, save and download it. Any third-party app, for example **Sublime**, **VS Code**, or **Eclipse**, can be used to add or modify code in the html file. The *widget.html* file contains the structure, design, and components of the widget.



You can copy the below sample code in the *widget.html* file to navigate to the *User Details* form upon choosing the correct option inside the **Verify widget** in [this](#) example.

```
1. function addData() {  
2.  
3.   ZOHO.CREATOR.init()  
4.     .then(function() {  
5.       config = {  
6.         action : "open",  
7.         url :  
           "https://creatorapp.zoho.com/megan/zylker-online-tutorials/#Form:User_Details",  
8.         window : "same"  
9.       }  
10. ZOHO.CREATOR.UTIL.navigateParentURL(config);  
11.     });  
12. }
```

13. Navigate to the *Settings* page of the app in which you need to embed your widget. Select **Widgets**.

14. Click **Create** button.

15. The *New Widget* pane will appear.

16. Enter a name for your Widget.

17. Choose **Internal** option in the *Hosting* drop down list.

18. Upload the packed zip file from the **dist** folder in the *Widget File* field.

19. Enter **/widget.html** in the *Index File* field. The **/widget.html** file is the index file i.e., this file will be incorporated as the widget in your page.

New Widget



Name*

Submit

Hosting

Internal



Widget File*

Navigate_parenturl.zip

Browse

Index File*

/widget.html



20. Click **Create**. The created widget is added to the *Widgets* page.

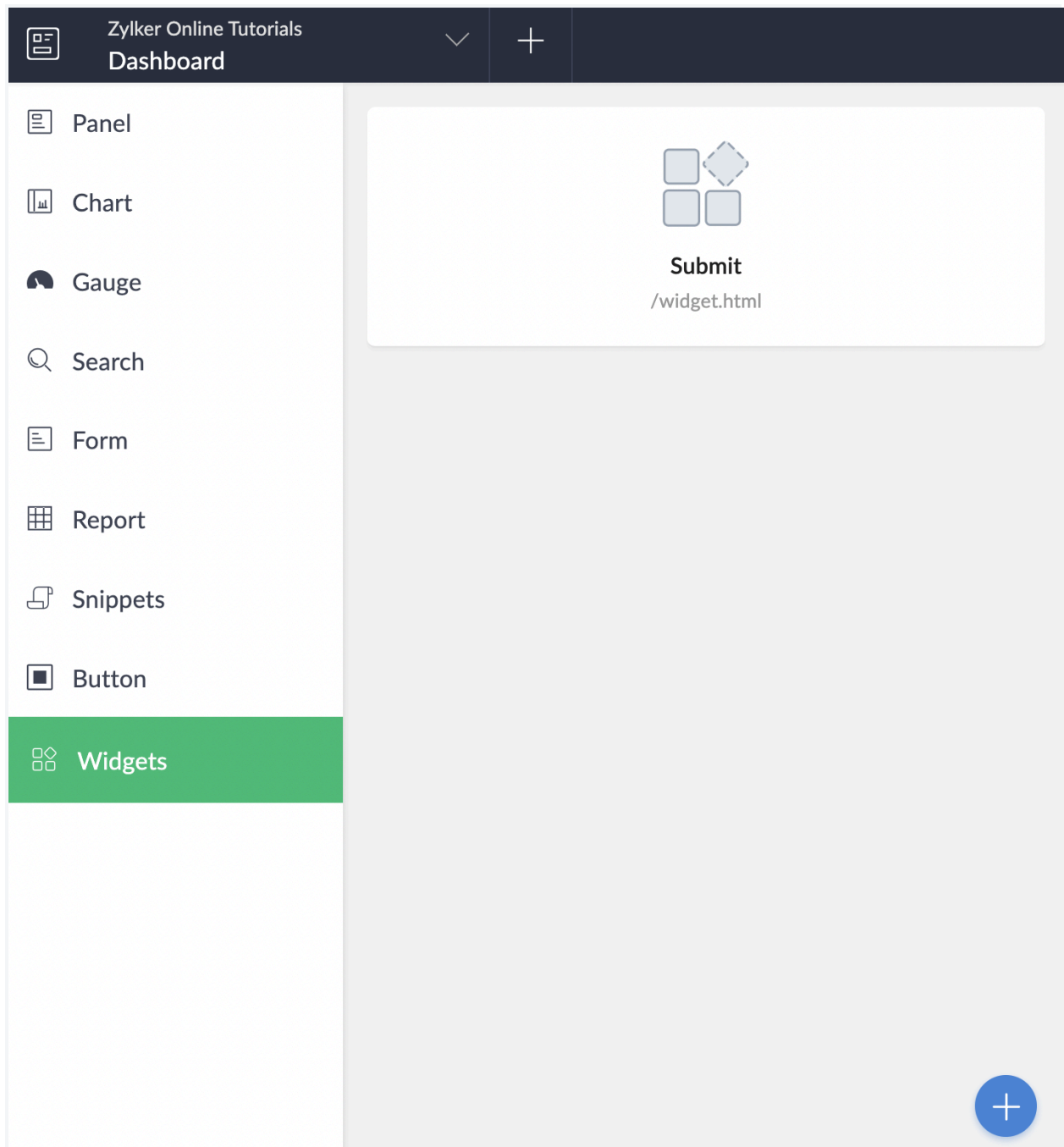
21. This widget will appear as a drag-and-drop element in the page builder for all the pages present in your application.

Add widget to a page

1. Navigate to the *Edit* mode of your app.

2. Open the **builder** of the required page.

3. Select **Widgets** from the left pane. The created widget will appear.



4. **Drag and drop** the widget into the builder area.

5. Click **Done** to exit the builder.

Get started with sample data

To help you get started quickly, we've provided [sample data](#) that you can download and use to start creating widgets!

Related Topics

- [Understand navigateparentURL\(\) method](#)
- [Understand widgets](#)
- [JS API documentation](#)

Helpful?20

Share :

New Widget



Name*

Sample Widget

Hosting

External



Index File*

<https://www.myapphosting.com/app/index.html>



Create

Cancel

6. Click **Create**. The created widget is added to the **Widgets** page.
7. This widget will appear as a drag-and-drop element in Page builder for all the Pages present in the application.

Importing a Figma file

You can convert your Figma design into a widget in Creator. To start with, you need to access the [Create Widgets from Figma UI Kit](#), read the instructions about using this kit, and then proceed to design your file in Figma. After the design has been finalized, copy and paste the [frame URL](#) and [access token](#) to generate widgets directly from the design in your Figma file. [Learn how](#)



To know more about the list of components supported in the UI kit, click [here](#).

External framework support

Creator Widget now supports external frameworks like React, Vue, etc. You can include the respective framework's **CDN URLs** in your widget app's index **HTML** files and code your app with the framework.

Note: It is suggested to use minified **production-ready CDN URLs** (available separately for each framework) when deploying to prevent the slowing down of your application.

Alternatively, you can use the respective framework's **NPM, CLI** method to create production-ready bundles and include them in your widget app. No additional inclusion of external URLs is required in this method.

Below is an example for using the React framework,

Users can follow the below format for other frameworks i.e you can include the script tag along with the respective framework's **CDN URLs**.

While in **development**, use the **development** URL to get meaningful error descriptions and suggestions.

```
<script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin></script>
```

```
<script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"
crossorigin></script>
```

While **deploying** use **minified version** URL to increase the loading speed of your application.

```
<script src="https://unpkg.com/react@17/umd/react.production.min.js" crossorigin></script>

<script src="https://unpkg.com/react-dom@17/umd/react-dom.production.min.js"
crossorigin></script>
```