EzeXtend Development Manual

Anthony Mesa

Nov 09, 2022

CONTENTS

CHAPTER

ONE

INSTALL

To work with ezeXtend, first you must install the current ezeXtend codebase in your development workspace. The current way to do this is to get the pre-setup project provided in the subversion repository for Mcube version 4.5.0.0. The subversion repository link is:

```
http://svnmcube.tcgdigital.com/svn/MCube_Implementation/Development_Artefacts/Dev-Area/4.
↪5.0.0/
```

Clone the repository to a folder of your choice, such as your user home folder.

Note: Use our Subversion guide (here) if you are unfamiliar with retrieving projects using TCG Digital's Subversion repository.

Once you have cloned the Subversion repository, the codebase for ezeXtend can be found within:

```
4.5.0.0/Code/module_designer/
```

Going forward, this `module_designer` folder will be referred to as `/EZEXTEND_ROOT`.

1.1 NVM

Node Version Manager is essential when working within a NodeJS environment as it allows us to download and use specific versions of NodeJS and Node Package Manager (NPM) given that many different projects may require different versions. The installation instructions for NPM can be found on their website

After installing, tell NVM to download the specific version we need (If you already have this version installed, you can skip the installation command):

```
nvm install 10.15.3
```

1.2 Nginx

Minimum Version: 14+ (v16.13.2 reccomended)

Nginx is a server software that we will be using as a simple reverse proxy for our development environment. This is required so that we can develop 'as if' we are working with a backend located outside of our development environment or on the cloud, etc. This is important for greater control in port allocation and bypassing any issues that might arise from violating Cross-Origin Resource Sharing (CORS) policy.

You will need to install the latest version of Nginx on your development environment. If you are using our Dockerized development container, then Nginx should alredy be installed.

This guide will only cover configuration for Linux based operating systems, so Windows or Mac users may need to adapt this guide to however Nginx differs on those systems (locations of config files, etc.).

Install Nginx with your package manager, e.g.:

```
sudo apt-get install nginx
```

CHAPTER

TWO

SETUP

2.1 NVM

Set NVM to use the NodeJs version required for this project:

```
nvm use 10.15.3
```

2.2 Nginx

ezeXtend will require an Nginx reverse proxy so that calls made to local routes will be redirected to their relevant services, whether those services are running on the local dev machine, or on a cloud-hosted VM.

To edit the Nginx configuration you are going to want to add a configuration file named `ezeXtend_proxy` inside the folder `/etc/nginx/sites-available/`. Within the file, you will need to define an Nginx server and the proxy route locations with minimal configuration, such as below:

```
server {
   location /module_designer/ {
      proxy_pass http://127.0.0.1:3001/;
   }

   location /elastic_api/ {
      proxy_pass http://127.0.0.1:9241/;
   }
}
```

The purpose of this configuration is to only register two url routes, `/module_designer/` and `/elastic_api/` that redirect calls for both the ezeXtend react page itself and the Elasticsearch database.

Now you can start Nginx with:

```
sudo nginx
```

Note: If Nginx starts successfully, you will see no errors or output, as it begins the server and runs it in the background. Because Nginx starts and runs in the background, before trying to run it, it is helpfull to use `htop` before hand to ensure that it isn't already running. If it us, kill the currently running Nginx with `sudo pkill -9 nginx`.

2.3 ezeXtend

Now we must prepare the ezeXtend project to be run. The ezeXtend project is split into two pieces, one inside of the other. The module designer project contains both the frontend for the ezeXtend page (in the `ui` folder) as well as backend code (in `server`) that helps it interface with Kibana/Elasticsearch behind the scenes.

Whenever we are extending the functionality of the front end (creating custom visualisations, etc.) we can run the front end in either a development mode which lacks an amount of full functionality, or a release mode that more closely emulates the full functionality of the page when working in tandem with Kibana/Elasticsearch. To achieve that full release functionality, you must build the `ui` react project inside `module_designer` and use the Nginx reverse proxy to provide Elasticsearch connection functionality. This may sound confusing but we will cover both aspects.

2.3.1 Development Mode

To run ezeXtend in development mode, execute:

```
cd /EZEXTEND_ROOT/ui
npm install
npm start
```

Assuming all went well, then the react project should start up and be available at `http://localhost:3000`.

2.3.2 Release Mode

To run ezeXtend in release mode, execute:

```
cd /EZEXTEND_ROOT/
npm install
cd /EZEXTEND_ROOT/ui
npm install
npm run build
cd ..
npm start
```

Again, assuming all went well, then the release version of the React project should be available at `http://<local domain>/module_designer` where the `<local domain>` can either be localhost, or any host that you have listed in your `/etc/hosts` file that redirect to `127.0.0.1`.

CHAPTER

THREE

CREATING A CUSTOM DASHBOARD COMPONENT

This guide explains the process of creating a custom component that would be included in the ezeXtend source code and built into MCube. In the future, we hope to create a system that allows clients and users to create dashboard components without having access to the MCube or ezeXtend source code, but until then, this is the best current solution for adding custom functionality to ezeXtend.

In this guide we will be making a custom Button element.

3.1 Add Entry in Components List Sidebar

Before creating the custom Button component itself, we need to provide details about our custom element to ezeXtend so that the custom component is displayed as an option in the component sidebar on the left side of the ezeXtend window. Doing this first will allow us to be able to test our Button in the UI and catch any errors in our Button development along the way.

First, open `/EZEXTEND_ROOT/ui/src/AppConstants/WidjetsMapping.js`. Inside of this file is a constant variable WidgetsMapping that contains a JSON object of the Labels to be displayed in the sidebar panel as available elemenents to use in the dashboard. Here we add a definition for our new element `CUSTOM_BUTTON` and the string label that will be shown for it.

Note: For the sake of brevity in the code examples, elipses are used to denote code that we are not concerned with for our example and thus does not need to be displayed in this tutorial.

```
1  export const WidgetsMapping = {
2      SHAPES: {
3          ...
4      },
5      CHARTS: {
6          ...
7      },
8      INPUTS: {
9          TEXTBOX: 'Text',
10         BUTTON: 'Button',
11         CUSTOM_BUTTON: 'Custom Button',
12         RADIO: 'Radio',
13         SELECT: 'Select',
14         MULTI_SELECT: 'Multi Select',
15         LABEL: 'Label',
16         IMAGE:'Img',
```

(continues on next page)

(continued from previous page)

```
    },
    OTHERS: {
        ...
    },
};
```

Next, we need to add the data for the component entry in the sidebar, that is, we need to specify things like the icon to be displayed, etc. To do this, first open `/EZEXTEND_ROOT/ui/src/Components/Sidebar/ComponentsData.js`. This file contains a constant variable `Groups` that contains a JSON object with lists of JSON descriptions of each of the sidebar components belonging to each group in the panel. Because we add our custom button to the `Inputs` section of the `WidgetsMapping` so too do we have to add a new sidebar component to the `Inputs` list within the JSON object:

```
export const Groups = {
   Shapes: [
      ...
   ],
   Charts: [
      ...
   ],
   Inputs: [
      {
            title: WidgetsMapping.INPUTS.TEXTBOX,
            icon: <BsInputCursor size={size} color={activeColor} />,
            active: true,
      },
      {
            title: WidgetsMapping.INPUTS.BUTTON,
            icon: <GiClick size={size} color={activeColor} />,
            active: true,
      },
      {
            title: WidgetsMapping.INPUTS.CUSTOM_BUTTON,
            icon: <HiCursorClick size={size} color={activeColor} />,
            active: true,
      },
      {
            title: WidgetsMapping.INPUTS.RADIO,
            icon: <IoMdRadioButtonOn size={size} color={activeColor} />,
            active: true,
      },
      {
            title: WidgetsMapping.INPUTS.SELECT,
            icon: <BiSelectMultiple size={size} color={activeColor} />,
            active: true,
      },
      {
            title: WidgetsMapping.INPUTS.LABEL,
            icon: <MdLabel size={size} color={activeColor} />,
            active: true,
      },
      {
```

(continues on next page)

(continued from previous page)

```
            title: WidgetsMapping.INPUTS.IMAGE,
            icon: <FaImages size={size} color={activeColor} />,
            active: true,
        },
    ],
    Others: [
        ...
    ],
};
```

Notice that we are referencing the title via the key-value pair that we entered into the WidgetsMapping object. For the icon, we are using one of the available click-related React icons that are freely available to React developers. You can find more of these icons at react-icons. We are using the `HiCursorClick` icon for our Button, so we will need to import that icon as a dependency at the top of the file:

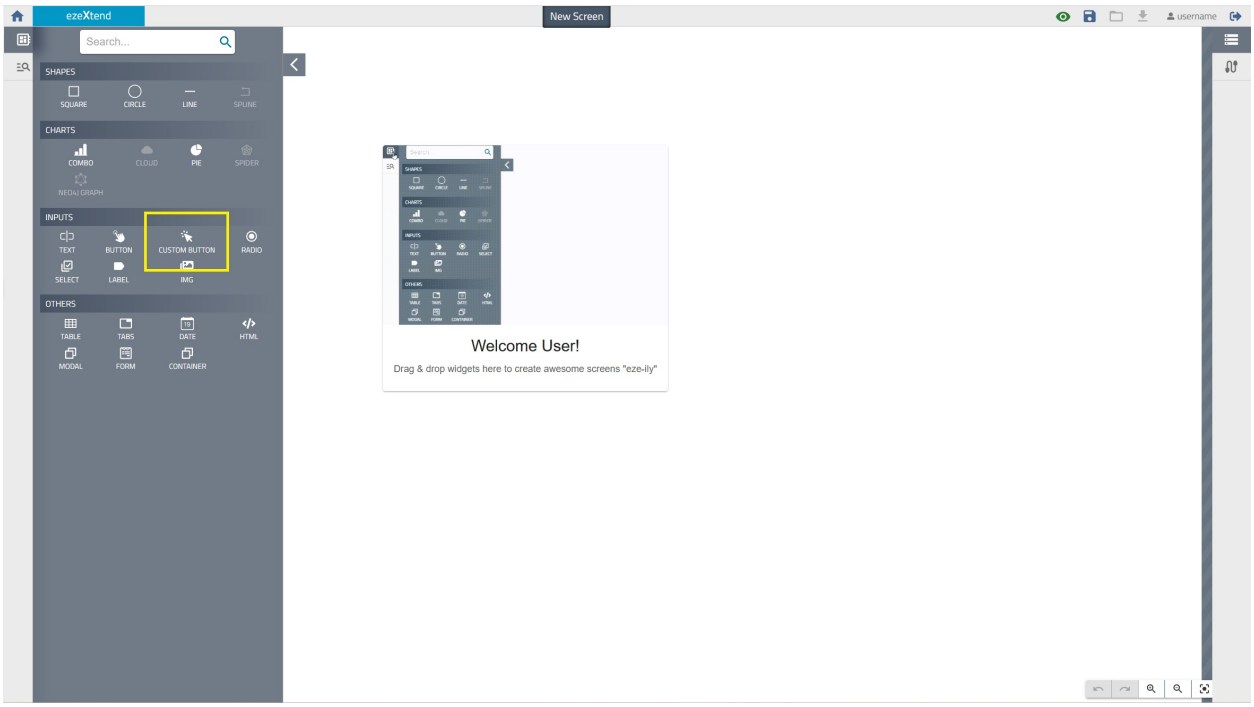
```
import {
    ...
} from 'react-icons/ai';

...
import { GrGraphQl } from "react-icons/gr";
import { BiSelectMultiple } from 'react-icons/bi';
import { RiCheckboxMultipleBlankLine } from 'react-icons/ri';
import { HiCursorClick } from 'react-icons/hi';
import { WidgetsMapping } from 'AppConstants';
const size = 20;
const color = 'rgb(203 203 203)';
...
```

If we run ezeXtend in Development Mode, we can see our component that has yet to be created listed as an option in the component sidebar:

3.2 Create Default Parameters for Component

Lorem ipsum et dolor



3.3 Create a Custom React Component

Lorem ipsum et dolor

3.4 Create Component Formatting Options

Lorem ipsum et dolor