Crema

About Crema

Designed and developed with high expertise, Crema is fully-featured React based Admin template that is configured with all the latest and trending libraries and technologies like Material-UI, Redux, React Hooks etc.

Crema has two dashboards viz. CRM and CRYPTO, lots of widgets and metrics, four fully functional apps (Todo, Mail, Contact and Scrumboard) and a lot of ready to use pages.

Key Features:-

- Three Modes - Light, Semi-Dark and Dark.
- Lot of Color combinations to beautify the Template.
- Six Languages Supported.
- Code Splitting and Reusability.

Crema is loaded with :-

- React Hooks
- React Redux
- Material UI
- Google Maps
- React Calendar
- Drag N Drop
- ReCharts
- React Player
- React Colors
- React Beautiful DND
- Dropzone
- React Timeline
- React Table
- Material UI Tables
- React Notifications
- React Library
- Axios-mock-adaptor
- Material Icons

Product Overview

Upon downloading the zip folder from the themeforest, you will get following content:-

1. source

It contains the source code of the template.

2. build

It contains the code for offline demo.

3. starter-template

This can be used as the starting point of a new project. It is integrated with **jwt-auth** for authentication. This is the ideal place to kickstart the project with all the necessary settings already done.

4. starter-template-aws

This can be used as the starting point of a new project. It is integrated with **aws** for authentication. This is the ideal place to kickstart the project with all the necessary settings already done.

5. starter-template-firebase

This can be used as the starting point of a new project. It is integrated with **firebase** for authentication. This is the ideal place to kickstart the project with all the necessary settings already done.

6. starter-template-auth0

This can be used as the starting point of a new project. It is integrated with **auth0** for authentication. This is the ideal place to kickstart the project with all the necessary settings already done.

7. document

It is the pdf of the online documentation of the template.

Folder Structure

The folder structure is simple and easy to understand. When you unzip the folder, you will find the following folders/files:-

1. **public**

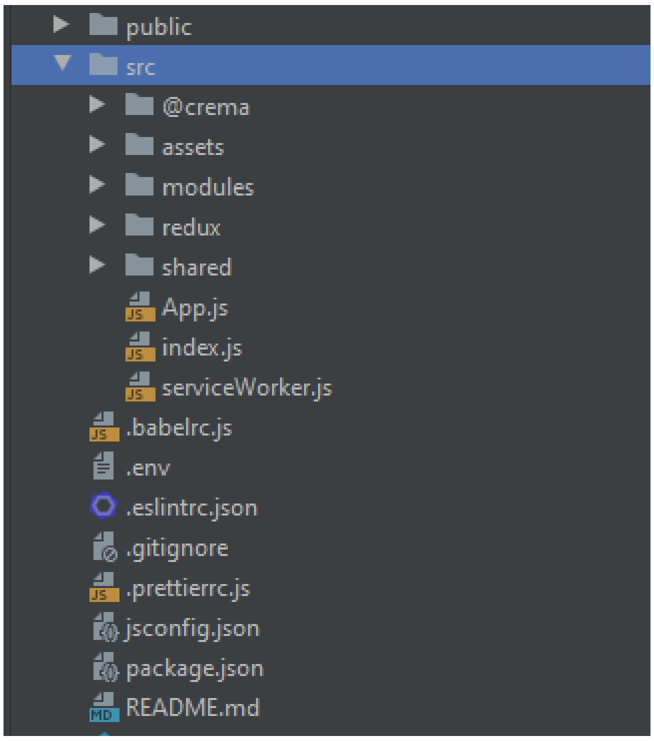
This folder contains the public assets and JavaScript used for rendering of Application on the browser.

2. **src**

The src contains the code of all the pages in the Crema. This folder is the heart of this project. This is the folder where the magic happens. Basically, this is the folder in which the user do all the work. The src folder structure is as follows:-

- @crema
    - core
      It contains all the general and common components that are used throughout the template.
    - services
      It contains database and authentication service providers settings and other data and files related to fake APIs.
    - utility
      It contains files all the important files related to the template.
- assets
  This folder contains all the images and other raw materials used in the template.
- modules
  All the code related to dashboard, apps and other pages lies in this folder. We have kept the folder structure very simple and easy to understand by breaking the folders into sub-folders and following proper naming. For example, modules folder contain a folder named 'dashboard' which contains all the files related to dashboard, similarly, a folder named 'apps' contains all the code related to apps included in the template.
- redux, As the name suggests, this folder contains the global state management related files. This folder is divided into three subfolders viz. actions, reducers, and store.
- shared

This folder contains the helping files. It contains language files, constants that are used in Redux and style related files.



```
▶ public
▼ src
    ▶ @crema
    ▶ assets
    ▶ modules
    ▶ redux
    ▶ shared
      App.js
      index.js
      serviceWorker.js
  .babelrc.js
  .env
  .eslintrc.json
  .gitignore
  .prettierrc.js
  jsconfig.json
  package.json
  README.md
```

Installation

This module explains the process of Setting up the Crema in your system for use. The following chapters in this module will explain the steps to Set up the Crema. You need to follow the steps one by one in the written order.

Pre-Requisites

To install Crema on the system, you need to develop a React Environment in the system. Following steps should be followed to develop React Environment:-

1. **Node.js**

Node.js is a JavaScript run time built needed to run a React App. You can install the recommended version of Node.js from the official website https://nodejs.org/en/

2. **Yarn**

Yarn is a project manager responsible for adding and installing all the dependencies required for running the Crema on the system. You can install the latest version of Yarn from the official website https://yarnpkg.com/

Installing Crema

Once you have followed the installation pre-requisite steps, You can follow the following steps to install and run the Crema:-

1. To run the Crema, Open the Crema folder in the editor of your choice. After following the folder, open the terminal and run the following command:-

```
yarn
```

this command will automatically install the required dependencies.

2. Now, run the following command to run the project in the browser.

```
yarn start
```

This command will start the project in the browser in development mode i.e. the project will run on localhost://300X URL.

3. To add the project to the live server, you need to make the Build of the project. Run the following command to make the Build of the project.

```
yarn build
```

After successful completion of build command, you will find following instructions on the terminal:-

The project was built assuming it is hosted at the server root. You can control this with the homepage field in your package.json. For example, add this to build it for GitHub Pages:

"homepage" : "http://myname.github.io/myapp",

The build folder is to ready be deployed. You may serve it with a static server:

```
yarn global add serve
serve -s build
```

Find out more about deployment here

https://create-react-app.dev/docs/deployment/

In case you want to deploy on another server please change accordingly.

Customization

Crema React Admin Template can be customized very easily. The template style, mode, layout, color combinations, direction can be customized easily either during run time or in development mode. We will talk about all of them individually in next chapters.

Template Style

We have included two template Styles in the Crema Admin Template. First is Material-UI based Standard theme style and Second is Modern Style. By Default, you will get Standard style, however, switching between the two is very easy. You can change theme style during run-time or during development mode.

Changing Template Style in run time:-

Click on the Settings button on the right corner of the screen just below the notification icon, a drawer will open in which you will find Theme Style Option and from there, you can choose between Standard and Modern theme style.

Changing Template Style in development mode (setting default Style) :-

To set the default template style, go to the following file:-

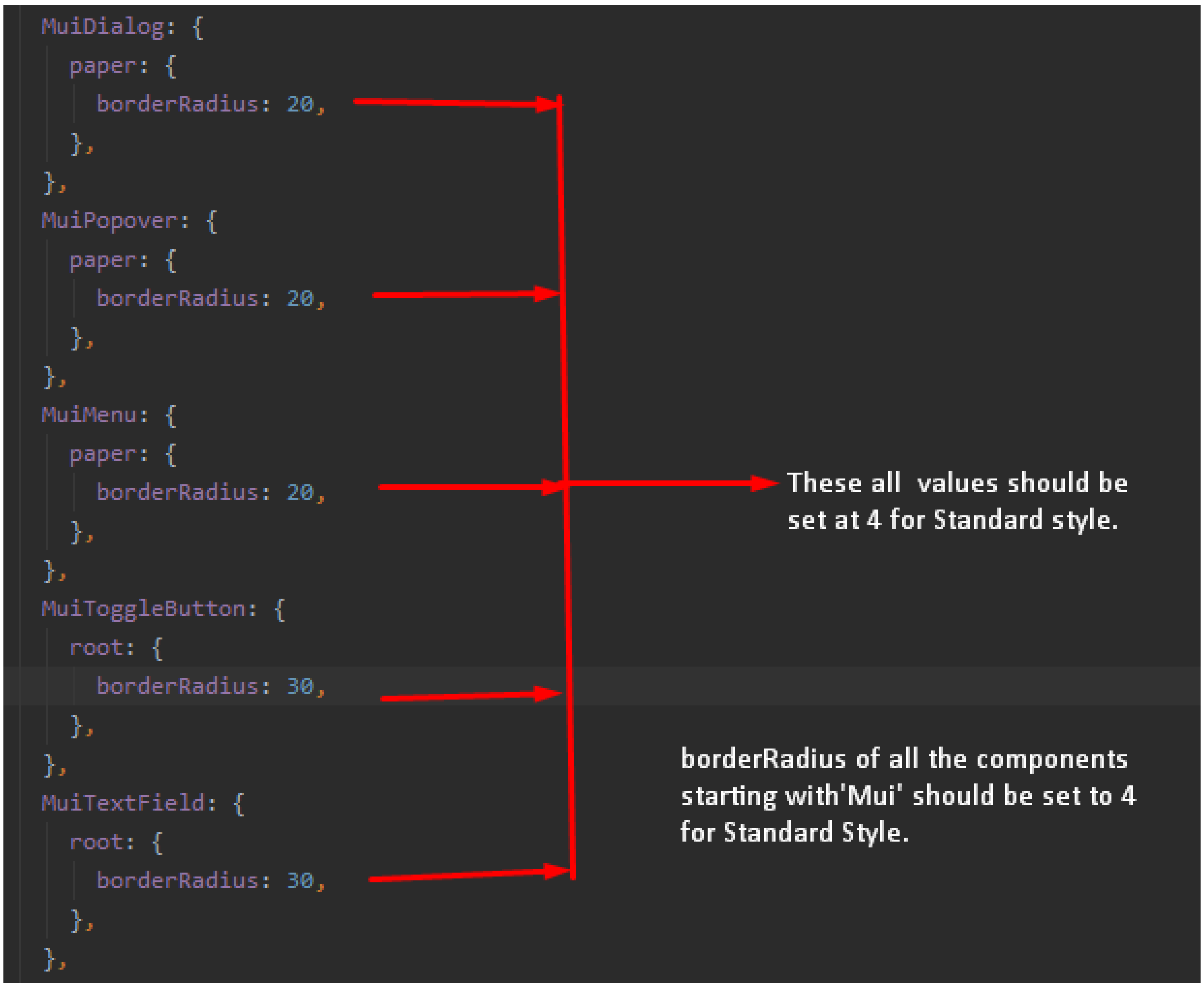
```
src/@crema/utility/ContextProvider/defaultConfig.js
```

In this file, you will find a property named 'themeStyle'. You need to set the value of 'themeStyle' according to your choice from the accepted values. You can find the accepted values of 'themeStyle' in the file with the path:-

```
src/shared/constants/AppEnums.js
```

After setting the value of 'themeStyle', you need to set the 'borderRadius' of all the muiComponents present in the 'overrides' property in the defaultConfig.js file.

The value of borderRadius for 'Standard Style' should be fixed at '4' for all the components starting with the word 'Mui'. You don't need to set the value for 'Modern Style' as by default all the values are set according to 'Modern Style'.



```
MuiDialog: {
  paper: {
    borderRadius: 20,
  },
},
MuiPopover: {
  paper: {
    borderRadius: 20,
  },
},
MuiMenu: {
  paper: {
    borderRadius: 20,
  },
},
MuiToggleButton: {
  root: {
    borderRadius: 30,
  },
},
MuiTextField: {
  root: {
    borderRadius: 30,
  },
},
```

These all values should be set at 4 for Standard style.

borderRadius of all the components starting with'Mui' should be set to 4 for Standard Style.

Template Mode

Three template modes are available in the Crema i.e. Light, Semi-Dark and Dark. You can choose any one of the three according to your choice. Switching between the Modes is very easy.

Changing Template Mode in run time:-

Click on the Settings button on the right corner of the screen just below the notification icon, a drawer will open in which you will find Template Mode Option and from there, you can choose between Light, Semi-Dark and Dark Mode.

Changing Template Mode in development mode (setting default Mode) :-

To set the default Template Mode, go to the following file:-

```
src/@crema/utility/ContextProvider/defaultConfig.js
```

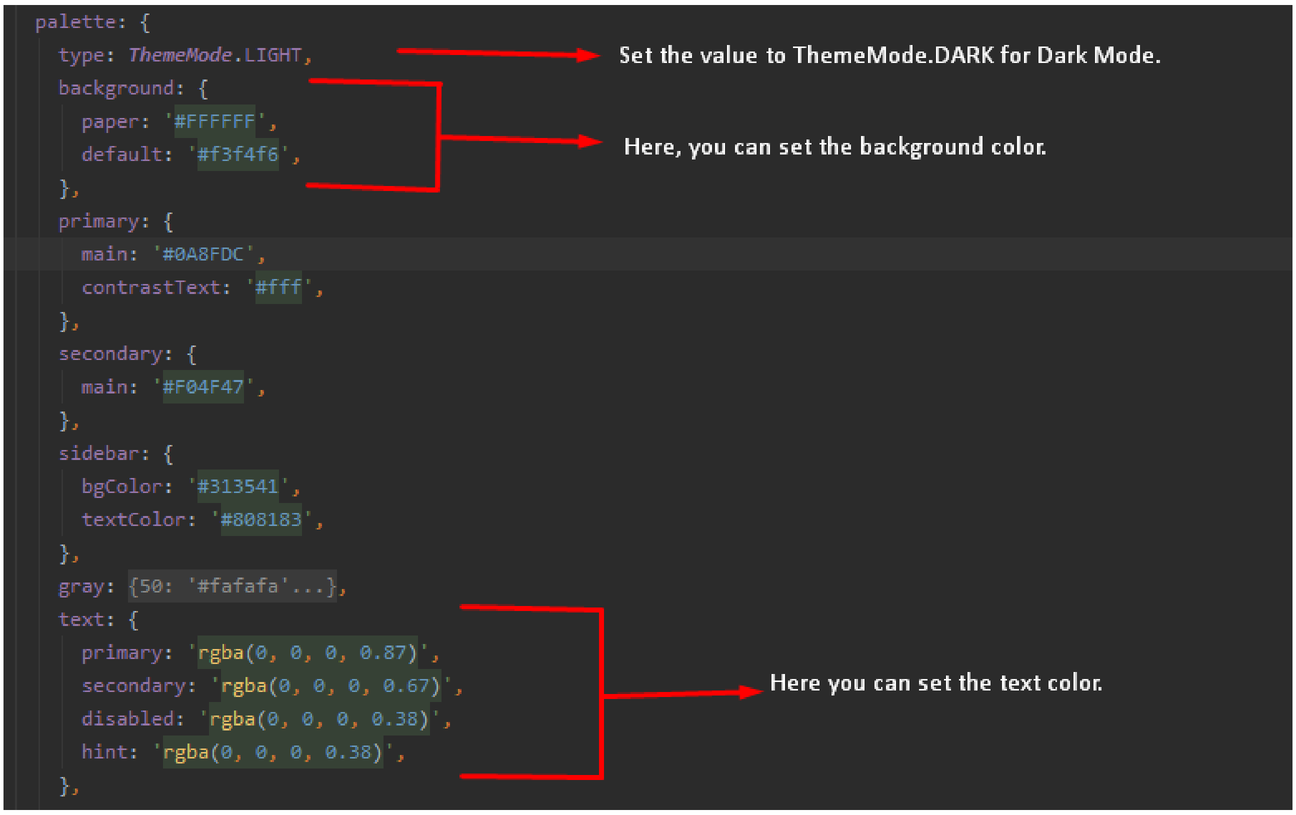
In this file, you will find a property named 'themeMode'. You need to set the value of 'themeMode' according to your choice from the accepted values. You can find the accepted values of 'themeMode' in the file with the path:-

```
src/shared/constants/AppEnums.js
```

For Setting Dark mode, you need to follow one more step :-

In the 'defaultConfig.js' file, there is a object type property named "pallete" which contains some template related properties. In the pallete object, there is a property named 'type', you need to set the value of type to 'ThemeMode.DARK'. After that, you need to set the background color and text color present in the 'pallete' property according to your choice. In case, you don't want to set the color, simply remove the 'background' and 'text' property, then the template will take the background and text color according to Material UI default colors.

Look at the image for reference:-



```
palette: {
  type: ThemeMode.LIGHT,
  background: {
    paper: '#FFFFFF',
    default: '#f3f4f6',
  },
  primary: {
    main: '#0A8FDC',
    contrastText: '#fff',
  },
  secondary: {
    main: '#F04F47',
  },
  sidebar: {
    bgColor: '#313541',
    textColor: '#808183',
  },
  gray: {50: '#fafafa'...},
  text: {
    primary: 'rgba(0, 0, 0, 0.87)',
    secondary: 'rgba(0, 0, 0, 0.67)',
    disabled: 'rgba(0, 0, 0, 0.38)',
    hint: 'rgba(0, 0, 0, 0.38)',
  },
```

Set the value to ThemeMode.DARK for Dark Mode.

Here, you can set the background color.

Here you can set the text color.

Template and Sidebar Color

Users can customize the template color and sidebar color according to their choice and requirements. Thousands of color combinations can be made by using the color customizer option. We have provided some primary and secondary color combinations and also provided the customizer option to make your own.

Changing Template Color in run time:-

Click on the Settings button on the right corner of the screen just below the notification icon, a drawer will open in which you will find color option and from there, you can choose from PRESET color combinations or create your own by selecting the CUSTOM option.

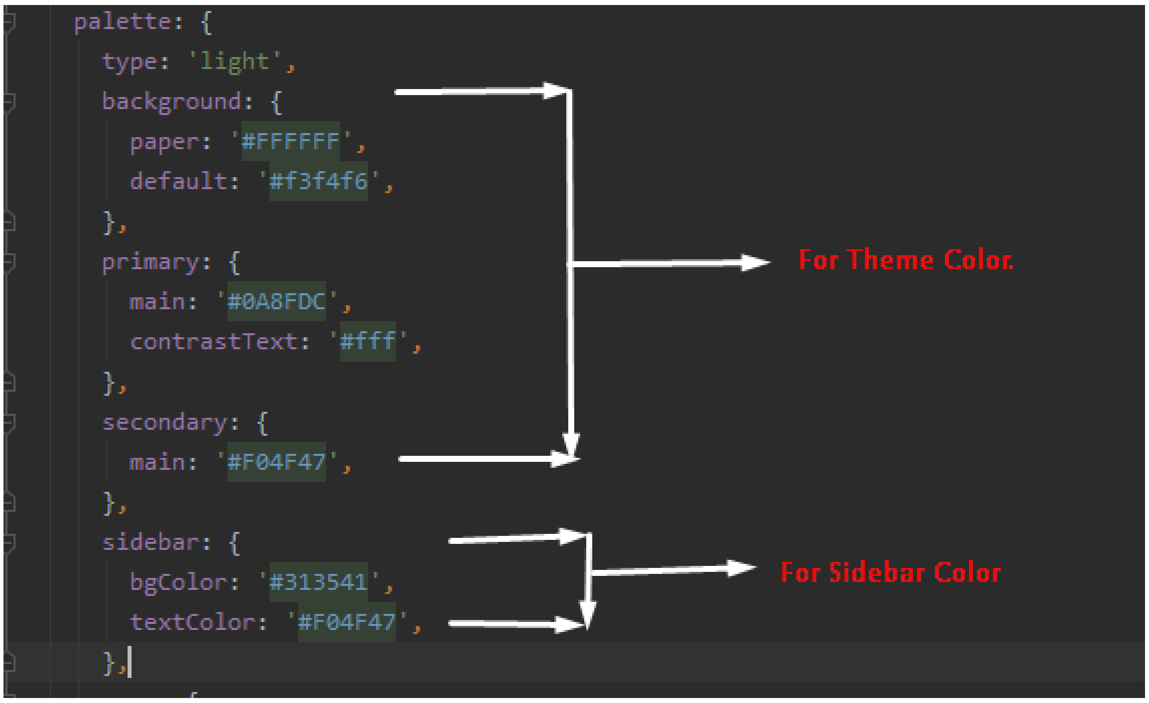
Changing Template Color and Sidebar Color in development mode (setting default Template Color and Sidebar Color):-

To set the default **Template Color and Sidebar Color**, go to the following file:-

```
src/@crema/utility/ContextProvider/defaultConfig.js
```

In this file, you will find an object type property named 'theme' which has a 'palette' object and in that, you can define the primary, secondary and other colors.

A screenshot is attached for the reference.



```
palette: {
  type: 'light',
  background: {
    paper: '#FFFFFF',
    default: '#f3f4f6',
  },
  primary: {
    main: '#0A8FDC',
    contrastText: '#fff',
  },
  secondary: {
    main: '#F04F47',
  },
  sidebar: {
    bgColor: '#313541',
    textColor: '#F04F47',
  },
```

Tempate Direction

The Crema template is fully RTL supported. On one toggle button, the user can switch between 'Right to left' direction or 'Left to Right' direction.

Changing Template Direction in run time:-

Click on the Settings button on the right corner of the screen just below the notification icon, a drawer will open in which you will find RTL support toggle button. By using that button, you can enable or disable RTL support.

Changing RTL support in development mode (setting default RTL type):-

To set the default RTL type, go to the defaultConfig.js file, it can be found at the path:-

```
src/@crema/utility/ContextProvider/defaultConfig.js
```

In this file, you will find a property named 'direction' . You need to set the value of 'direction' according to your choice from the accepted values. You can find the accepted values of 'direction' in the file with the path:-

```
src/shared/constants/AppEnums.js
```

Since some languages are read from 'Right to Left', so whenever that language is chosen, the template direction changes to 'Right to Left'. This is managed by the property name "rtlLocale". This property can be found in the above file. It takes an array as the value and language codes for which RTL is to be enabled can be passed in it.

```
rtlLocale: ['ar'],
```

Currently, 'ar' which has been used for the Arabic language in the Crema has been passed in the array which means whenever the Arabic language is chosen, RTL will be enabled. Similarly, you can pass any other language locale code in the array to enable RTL on the selection of that language.

Navigation Style

We have included eight navigation styles in the Crema. You can choose the Navigation style according to your choice and requirement by following under-written steps.

Changing Navigation Style in run time:-

Click on the settings button on the right corner of the screen just below the notification icon, a drawer will open in which you will find navigation styles option and from there, you can navigation style of your choice.

Changing Navigation Style in development mode (setting default Navigation Style):-

To set the default Navigation Style, go to the following file:-

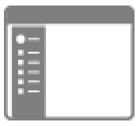
```
src/@crema/utility/ContextProvider/defaultConfig.js
```

In this file, you will find a property named 'navStyle' which accepts eighth values. You need to set the value of 'navStyle' according to your choice. Layouts and their accepted values(which you have to assign to 'navStyle') can be found in the file with path:-

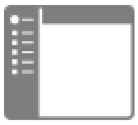
```
src/shared/constants/AppEnums.js
```

Layouts and their schematic designs are as follows:-

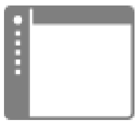
1. Default Layout



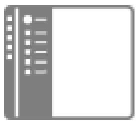
2. Standard Layout



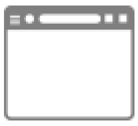
3. Mini Layout



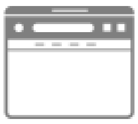
4. Drawer Layout



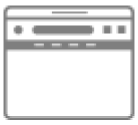
5. Bit Bucket Layout



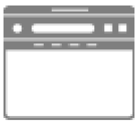
6. Horizontal Default Layout



7. Horizontal Light Navigation Layout



8. Horizontal Dark Navigation Layout



Footer

Whether the user want to keep the footer or not, this is totally up to the user's choice. We have provided the option of disabling or enabling the footer. You can enable or disable the footer by following the under-written steps.

Disabling or Enabling the Footer in Run Time:-

Click on the Settings button on the right corner of the screen just below the notification icon, a drawer will open in which you will find a Footer toggle button and by using that button, you can enable or disable Footer.

Enabling or Disabling in development mode (setting default Footer):-

To enable or disable the Footer, go to the following file:-

```
src/@crema/utility/ContextProvider/defaultConfig.js
```

In this file, you will find a property named 'footer'. You need to set the value of Footer to true, if you want to enable the Footer or false if you want to disable the Footer.

Footer Type

If the footer is enabled, then we have provided the option of setting the type of Footer. The user can set the type of footer during run-time and in development mode by following the under-written steps:-

Setting the Footer type in Run Time:-

Click on the settings button on the right corner of the screen just below the notification icon, a drawer will open in which you will find a footer type select dropdown and by using that dropdown, you can choose the footer type.

Setting the Footer Type in development mode (setting default Footer Type):-

To set the default footer type, go to the following file:-

```
src/@crema/utility/ContextProvider/defaultConfig.js
```

In this file, you will find a property named 'footerType'. You need to set the value of 'footerType' according to your choice from the accepted values. You can find the accepted values of 'footerType' in the file with the path:-

```
src/shared/constants/AppEnums.js
```

Route Transition

User can change the route transition during run time or can set the default route transition type by following these steps:-

Changing Route Transition in run time:-

Click on the settings button on the right corner of the screen just below the notification icon, a drawer will open in which you will find a route transition select dropdown and by using that dropdown, you can choose the desired route transition.

Changing Route Transition in development mode (setting default Route Transition):-

To set the default route transition, go to the following file:-

```
src/@crema/utility/ContextProvider/defaultConfig.js
```

In this file, you will find a property named 'rtAnim'. You need to set the value of 'rtAnim' according to your choice from the accepted values. You can find the accepted values of 'rtAnim' in the file with the path:-

```
src/shared/constants/AppEnums.js
```

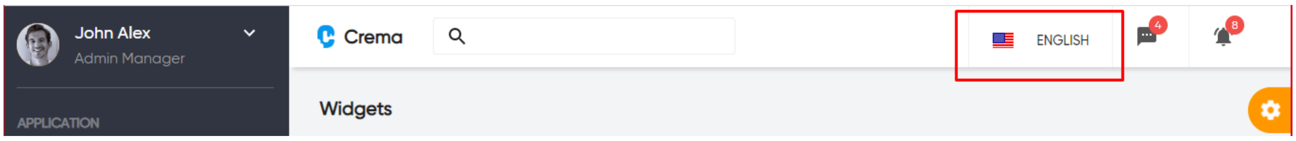
Multi-Lingual Support

As of now, Crema template supports six languages. These are English, Spanish, French, Italian, Arabic and Chinese. Users can select or change any of the given languages by following simple steps in following chapters.

Changing Language

Changing Language in run time:-

You will find a Language selector in the header, click on that and select the language of your choice.



Changing Language in development mode (setting default Language):-

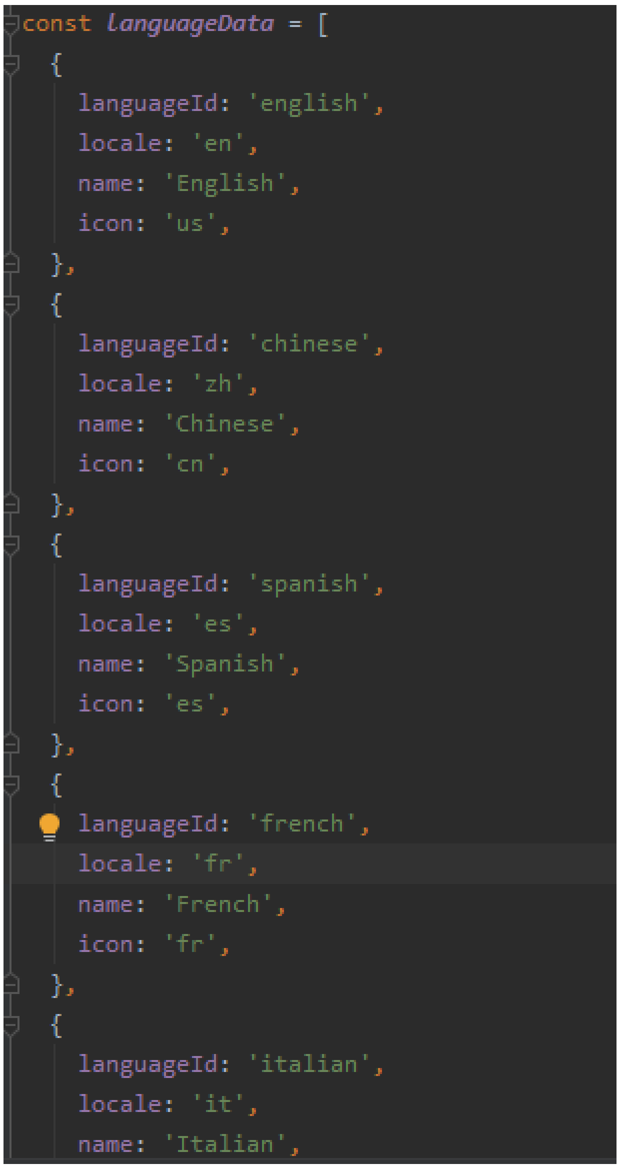
To set the default language, go to the following file:-

```
src/@crema/utility/ContextProvider/defaultConfig.js
```

In this file, you will find a property named 'locale' which is an object, you have to replace this object with any accepted value in order to set your language as the default language. You can get the list of accepted values by opening file having path:-

```
src/@crema/core/LanguageSwitcher/data.js
```

A screenshot of the data file is attached for the reference



```
const languageData = [
  {
    languageId: 'english',
    locale: 'en',
    name: 'English',
    icon: 'us',
  },
  {
    languageId: 'chinese',
    locale: 'zh',
    name: 'Chinese',
    icon: 'cn',
  },
  {
    languageId: 'spanish',
    locale: 'es',
    name: 'Spanish',
    icon: 'es',
  },
  {
    languageId: 'french',
    locale: 'fr',
    name: 'French',
    icon: 'fr',
  },
  {
    languageId: 'italian',
    locale: 'it',
    name: 'Italian',
```

Adding New Language

In order to add new language to the template, follow the following steps:-

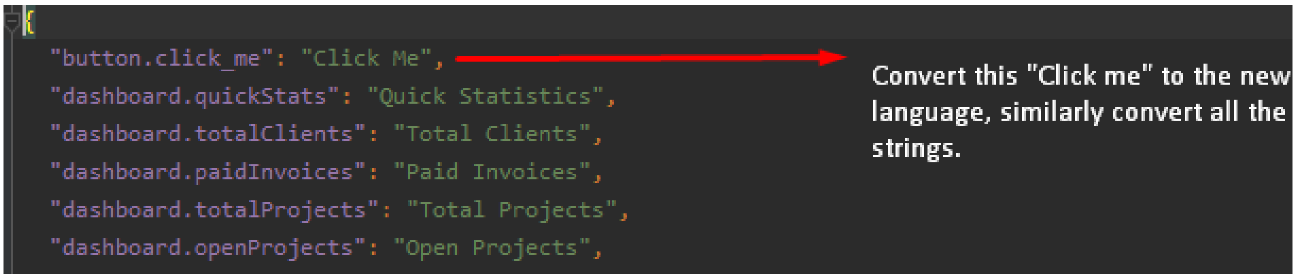
1. Make a new .json file in the folder with the path :-

```
src/shared/localization/locales
```

2. Copy the content of the file 'en-US.json' kept adjacent to the file you made and paste it into your file. The path of file 'en-US.json' is:-

```
src/shared/localization/locales/en_US.json
```

3. In the next step, you need to translate the values of translation variables into the new language.



```
{
  "button.click_me": "Click Me",
  "dashboard.quickStats": "Quick Statistics",
  "dashboard.totalClients": "Total Clients",
  "dashboard.paidInvoices": "Paid Invoices",
  "dashboard.totalProjects": "Total Projects",
  "dashboard.openProjects": "Open Projects",
```

Convert this "Click me" to the new language, similarly convert all the strings.

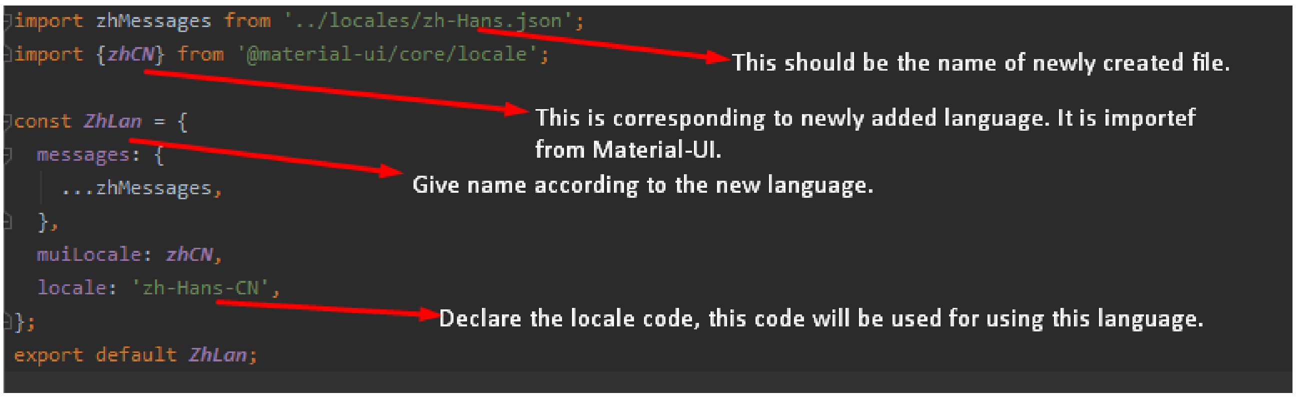
4. Now make a new file in the folder with the path:-

```
src/shared/localization/entries
```

The name of the file should be same as the above-created file, just the extension will be different, this will be a .js file, earlier we created a .json file.

Copy the following code, paste it into the newly made file and modify the content according to the changes suggested in the image given below the code.

```
import saMessages from '../locales/es_ES.json';
import {esES} from '@material-ui/core/locale';

const saLang = {
  messages: {
    ...saMessages,
  },
  muiLocale: esES,
  locale: 'es',
};
export default saLang;
```



```
import zhMessages from '../locales/zh-Hans.json';
import {zhCN} from '@material-ui/core/locale';

const ZhLan = {
  messages: {
    ...zhMessages,
  },
  muiLocale: zhCN,
  locale: 'zh-Hans-CN',
};
export default ZhLan;
```

This should be the name of newly created file.

This is corresponding to newly added language. It is importef from Material-UI.

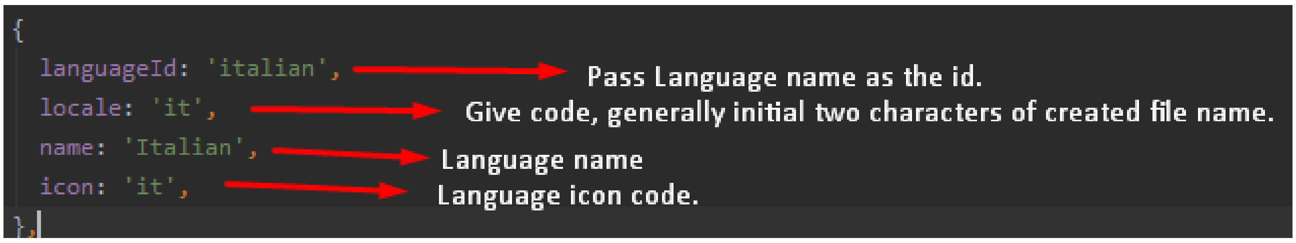
Give name according to the new language.

Declare the locale code, this code will be used for using this language.

5. Now go to the file with the following path:-

```
src/@crema/core/LanguageSwitcher/data.js
```

and add the new object corresponding to the new language, code should look something like this:-



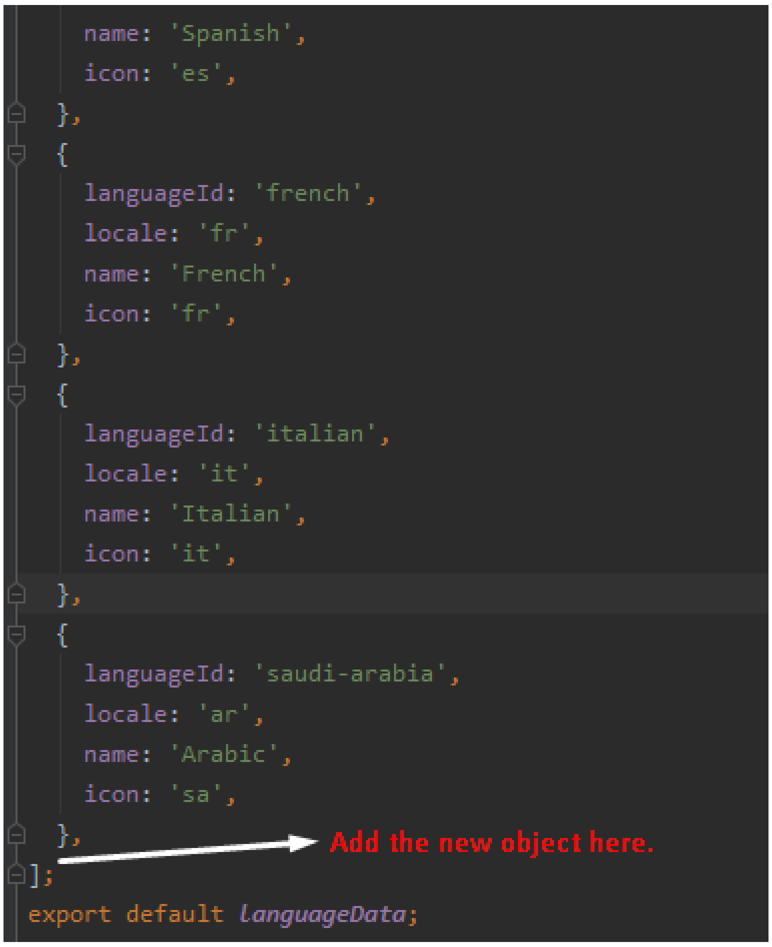
```
{
  languageId: 'italian',
  locale: 'it',
  name: 'Italian',
  icon: 'it',
},
```

Pass Language name as the id.

Give code, generally initial two characters of created file name.

Language name

Language icon code.

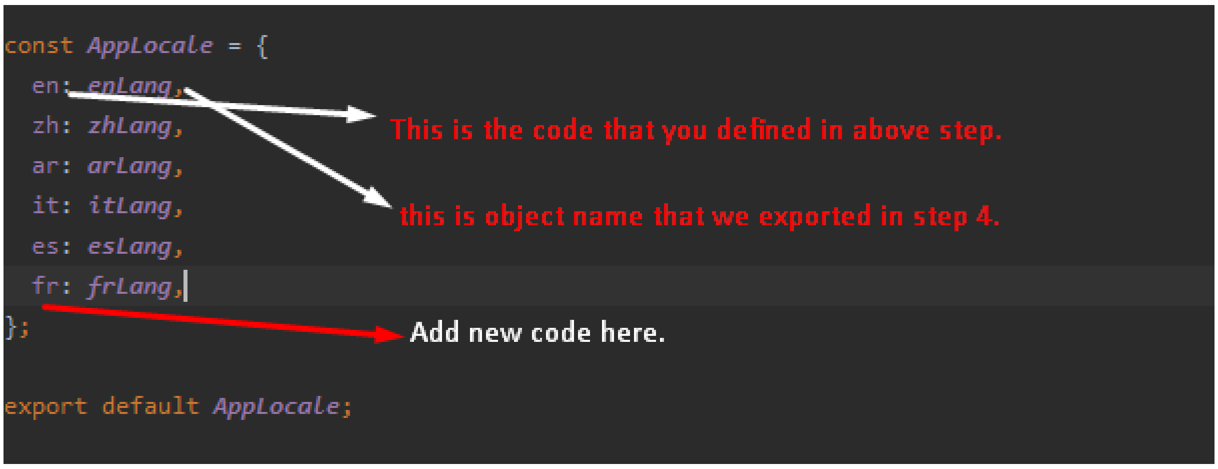


```
    name: 'Spanish',
    icon: 'es',
  },
  {
    languageId: 'french',
    locale: 'fr',
    name: 'French',
    icon: 'fr',
  },
  {
    languageId: 'italian',
    locale: 'it',
    name: 'Italian',
    icon: 'it',
  },
  {
    languageId: 'saudi-arabia',
    locale: 'ar',
    name: 'Arabic',
    icon: 'sa',
  },
];
export default languageData;
```

6. In the last step, open the file on the path:-

```
src/shared/localization/index.js
```

and add the language code defined in the above step and assign the value of the variable exported in step 4 to this code. Follow the following code for reference:-



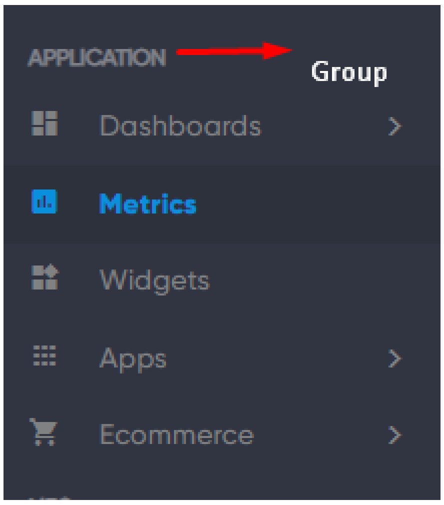
```
const AppLocale = {
  en: enLang,
  zh: zhLang,
  ar: arLang,
  it: itLang,
  es: esLang,
  fr: frLang,
};

export default AppLocale;
```

This is the code that you defined in above step.
this is object name that we exported in step 4.
Add new code here.

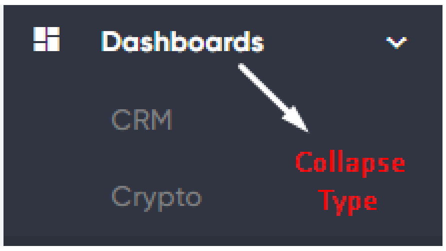
Sidebar Menu

In Crema, the menu are of three types:-

1. Group



2. Collapse type



3. Individual

Any Navigation item that is not associated with any group of navigations and does not have its children navigations is considered to be Individual navigation.

Adding New Menu

Adding a new menu in the Crema is a cup of cake. You don't have to add the menu separately for horizontal layouts and vertical layouts. You simply have to add a new menu to a single file and the menu will be added to all the layouts automatically.

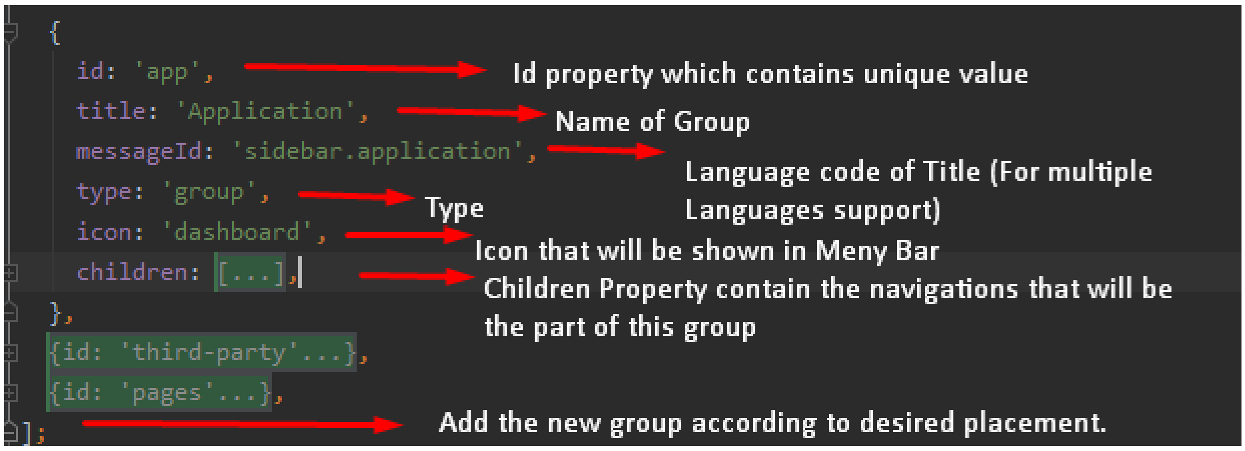
Steps to add a new menu:-

1. Go to the file with the path : -

```
src/modules/routesConfig.js
```

2. Adding a group of menu's:-

If you want to add a group of menu, Just add a new object in the above-said file with the following properties as shown in the below screenshot:-



```
{
  id: 'app',
  title: 'Application',
  messageId: 'sidebar.application',
  type: 'group',
  icon: 'dashboard',
  children: [...],
},
{id: 'third-party'...},
{id: 'pages'...},
];
```

Id property which contains unique value

Name of Group

Language code of Title (For multiple Languages support)

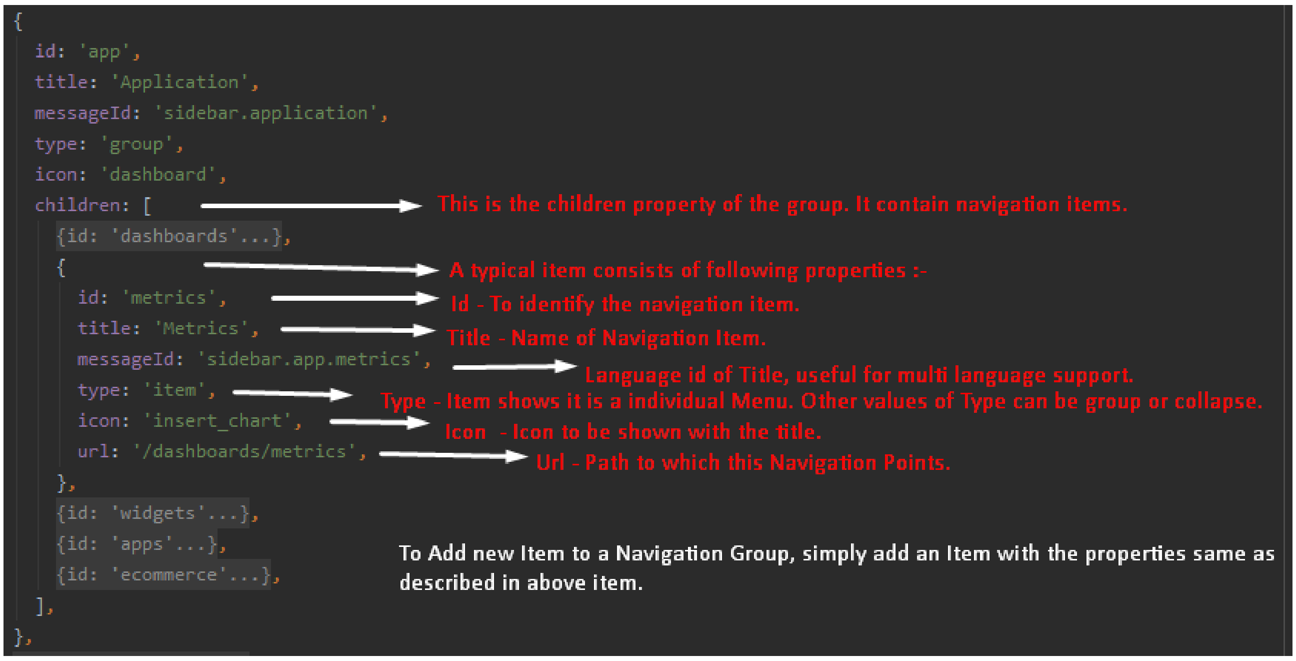
Type

Icon that will be shown in Meny Bar

Children Property contain the navigations that will be the part of this group

Add the new group according to desired placement.

The children's property of a group contains objects as shown below.



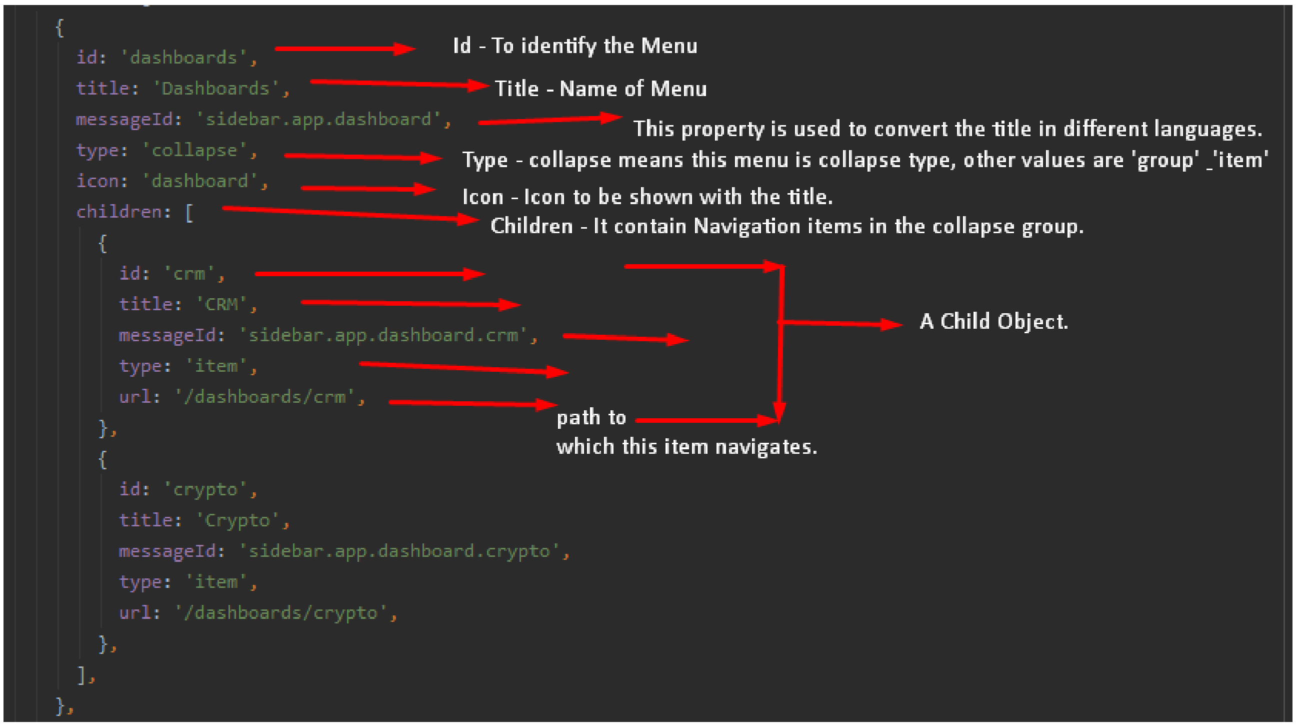
```
{
  id: 'app',
  title: 'Application',
  messageId: 'sidebar.application',
  type: 'group',
  icon: 'dashboard',
  children: [
    {id: 'dashboards'...},
    {
      id: 'metrics',
      title: 'Metrics',
      messageId: 'sidebar.app.metrics',
      type: 'item',
      icon: 'insert_chart',
      url: '/dashboards/metrics',
    },
    {id: 'widgets'...},
    {id: 'apps'...},
    {id: 'ecommerce'...},
  ],
},
```

This is the children property of the group. It contain navigation items.
A typical item consists of following properties :-
Id - To identify the navigation item.
Title - Name of Navigation Item.
Language id of Title, useful for multi language support.
Type - Item shows it is a individual Menu. Other values of Type can be group or collapse.
Icon - Icon to be shown with the title.
Url - Path to which this Navigation Points.

To Add new Item to a Navigation Group, simply add an Item with the properties same as described in above item.

3. Adding Collapse type Menu:-

To add a collapse type menu, go to the 'routesConfig' file (path given in step 1) and add a new object to the 'routesConfig' array. The new object show look like this:-

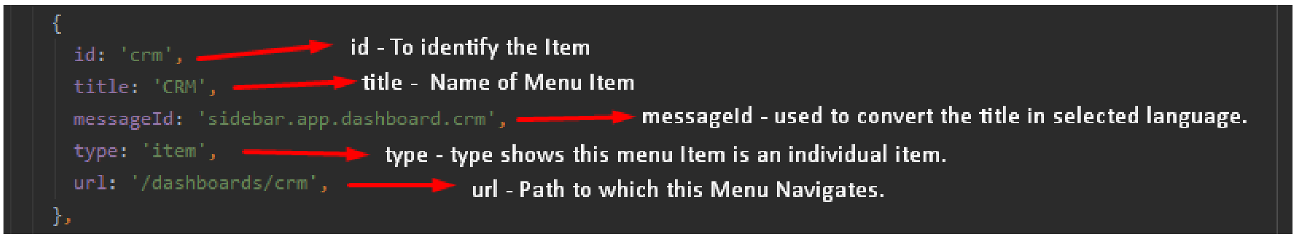


```
  {
    id: 'dashboards',
    title: 'Dashboards',
    messageId: 'sidebar.app.dashboard',
    type: 'collapse',
    icon: 'dashboard',
    children: [
      {
        id: 'crm',
        title: 'CRM',
        messageId: 'sidebar.app.dashboard.crm',
        type: 'item',
        url: '/dashboards/crm',
      },
      {
        id: 'crypto',
        title: 'Crypto',
        messageId: 'sidebar.app.dashboard.crypto',
        type: 'item',
        url: '/dashboards/crypto',
      },
    ],
  },
```

Id - To identify the Menu
Title - Name of Menu
This property is used to convert the title in different languages.
Type - collapse means this menu is collapse type, other values are 'group' _'item'
Icon - Icon to be shown with the title.
Children - It contain Navigation items in the collapse group.
A Child Object.
path to which this item navigates.

4. Adding a menu item:-

Adding a menu item is very easy, You just have to add an object in the 'routesConfig' array in the 'routesConfig' file (path given in step 1). The Item object should look contain the following properties as shown in the screenshot shown below.



```
{
  id: 'crm',
  title: 'CRM',
  messageId: 'sidebar.app.dashboard.crm',
  type: 'item',
  url: '/dashboards/crm',
},
```

id - To identify the Item
title - Name of Menu Item
messageId - used to convert the title in selected language.
type - type shows this menu Item is an individual item.
url - Path to which this Menu Navigates.

In the nutshell, You just have to add an object in order to array a new menu. The object contains a property named 'type' which can have three values i.e. 'group', 'collapse' and 'item'. You just have to pass the correct value to this 'type' property as per the type of menu you want to add. All other properties in the object remain the same.

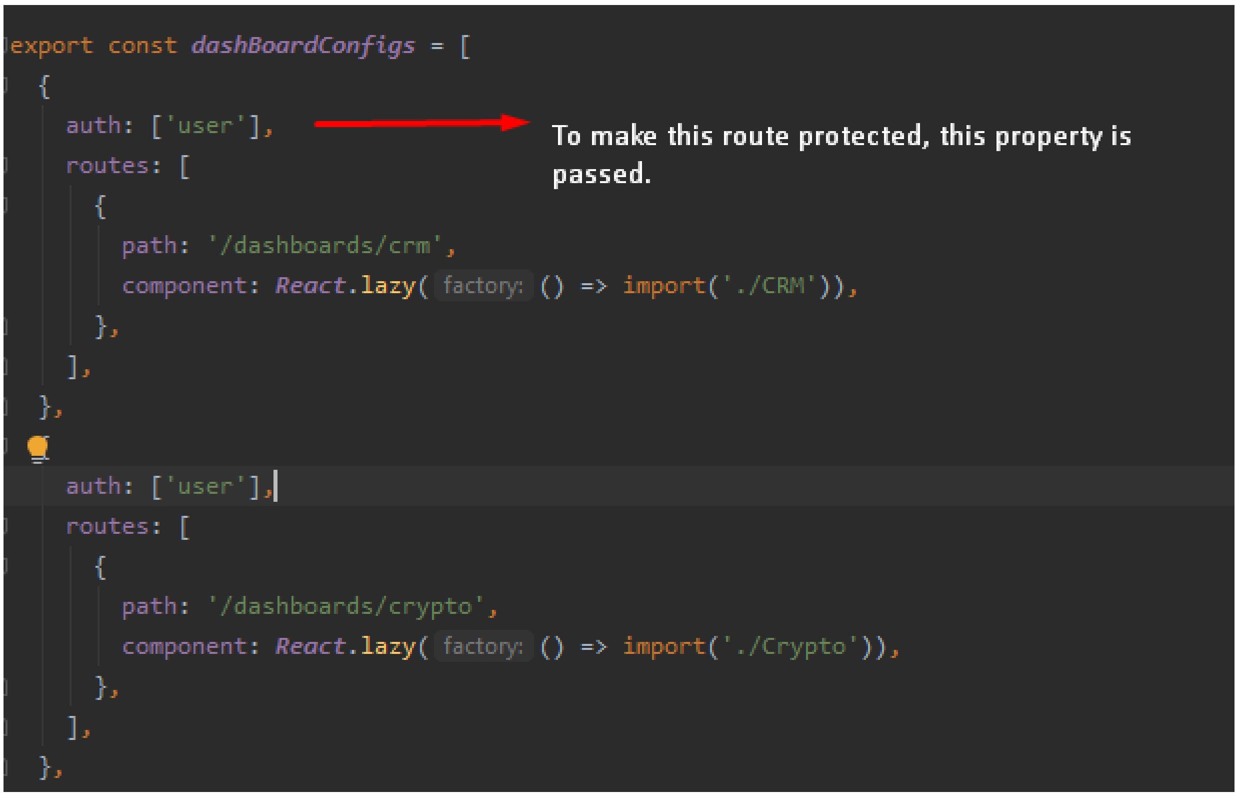
Route Protection

Route protection means protecting any defined path from unauthorized access. For example, Signin or Signup pages are common paths/unrestricted paths and any user can access it. However there are many paths or pages that only loggedin users can access, those paths are called Protected Routes. In Crema, we have provided the functionality of route protection, you have to follow the following steps to protect the route:-

Go to the file, where the routes have been declared. For example, let's go to the dashboard file where the dashboard related routes have been declared, the file can be found at:-

```
src/modules/dashboard/index.js
```

While declaring the route, you have to pass one extra property named "auth" in route object in order to make the route protected.



```
export const dashBoardConfigs = [
  {
    auth: ['user'],
    routes: [
      {
        path: '/dashboards/crm',
        component: React.lazy( factory: () => import('./CRM')),
      },
    ],
  },
  {
    auth: ['user'],
    routes: [
      {
        path: '/dashboards/crypto',
        component: React.lazy( factory: () => import('./Crypto')),
      },
    ],
  },
```

If this 'auth' property is added, this route will be protected and requires a loggedin user to access this path and if this property is not passed, then this path is directly accessible without any condition.

Loggedin User

To get the data of loggedin user wherever you need, simply write one line code i.e.

```
const user = useAuthUser();
```

you have to import useAuthUser from

```
src/@crema/utility/AppHooks.js
```

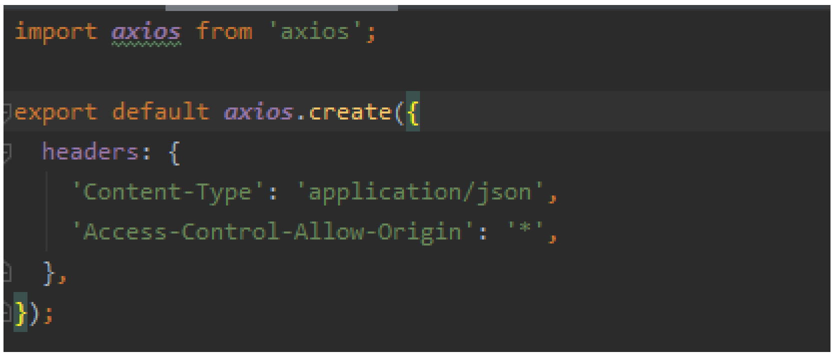
Axios Setup

In Crema, the Axios library has been used for API calls. You need to do the following settings for using it:-

Go to ApiConfig.js file, you can find this file at the path:-

```
src/@crema/services/ApiConfig.js
```

In this file, you will find the Axios setup, the content of the file looks like this:-



```
import axios from 'axios';

export default axios.create({
  headers: {
    'Content-Type': 'application/json',
    'Access-Control-Allow-Origin': '*',
  },
});
```

Here, you can define your base path Url and add more headers if required.