import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import { Provider } from "react-redux";

import store from "./redux/store";

import Dashboard from "./pages/Dashboard";

import Settings from "./pages/Settings";

import ThemeProvider from "./components/themes/ThemeProvider";

function App() {

return (

<Provider store={store}>

<ThemeProvider>

<Router>

<Routes>

<Route path="/" element={<Dashboard />} />

<Route path="/settings" element={<Settings />} />

</Routes>

</Router>

</ThemeProvider>

</Provider>

);

}

export default App;

To achieve this, we'll break the setup into two separate projects:

**1️⃣ Widget Library Component Library Project**

This will be a standalone React component library that includes all the widget types and utilities for creating widgets. It will expose components that can be imported and used in the main **dashboard app**.

**2️⃣ Dashboard Framework Web App Project**

This will be the main app that integrates the **widget library**, handles **Redux state**, **API integration**, and the **dashboard layout**. It will consume widgets from the component library.

**🛠️ Steps to Set Up**

1. **Widget Library Component Library (Separate Project)**
   * Set up a new React project for the widget library.
   * Create reusable widget components (charts, tables, maps, etc.).
   * Expose widgets as npm packages or local modules.
   * Implement **widget configuration** and **theme** support.
2. **Main Web App (Dashboard)**
   * Use **React + TypeScript** with **Redux** for state management.
   * **Widget Library** components will be imported from the widget library.
   * Integrate **API calls** to fetch data for widgets and dashboards.
   * Implement **theme switching** and **dashboard layout**.

**Widget Library Project (Separate)**

1. **Initial Setup**

bash

CopyEdit

npx create-react-library my-widget-library

cd my-widget-library

npm install

1. **Widget Components (e.g., ChartWidget.tsx)**
   * Create reusable components for different widget types like charts, tables, etc.
   * Make them configurable via props for flexibility.
2. **Export Components**
   * Ensure widgets are exported from the library for consumption in the dashboard app.

**⚙️ Main Dashboard Web App Project**

1. **Initial Setup**

bash

CopyEdit

npx create-react-app thingsboard-dashboard --template typescript

cd thingsboard-dashboard

npm install redux react-redux react-router-dom axios

1. **Redux Setup**  
   Set up Redux state management to handle widget configuration, dashboard settings, and user preferences.
2. **Widget Integration**  
   Import widgets from the **widget library** into the dashboard app.

**🔗 Integration Between Projects**

* The **widget library** can be either published as an npm package or managed locally as a dependency.
* The **main web app** will import the widgets and use them within the **dashboard layout**.

Would you like to proceed with detailed steps on setting up each of these projects, or do you need any specific part of the setup next?

Here’s a step-by-step guide to setting up both the **Widget Library** component library and the **Dashboard Web App** while ensuring **integration with the ThingsBoard backend APIs** for the dashboard and widget systems.

### ****Widget Library Component Library Project****

This is a standalone project that contains reusable React components for widgets. It will be integrated into the **Dashboard Web App**.

#### **Step 1: Create the Widget Library Project**

1. **Create the library** using the react-library template:

bash

CopyEdit

npx create-react-library my-widget-library

cd my-widget-library

1. **Install dependencies**:

bash

CopyEdit

npm install

#### **Step 2: Define Reusable Widget Components**

Create the widgets, for example, a ChartWidget.tsx.

tsx

CopyEdit

// src/widgets/ChartWidget.tsx

import React from "react";

interface ChartWidgetProps {

title: string;

data: number[];

}

const ChartWidget: React.FC<ChartWidgetProps> = ({ title, data }) => {

return (

<div className="chart-widget">

<h3>{title}</h3>

<div className="chart-container">

{/\* Implement chart rendering (use chart library like Chart.js or Recharts) \*/}

<p>Data: {JSON.stringify(data)}</p>

</div>

</div>

);

};

export default ChartWidget;

#### **Step 3: Export Widgets**

Make sure all widgets are exported from the library:

tsx

CopyEdit

// src/index.ts

import ChartWidget from './widgets/ChartWidget';

export { ChartWidget };

#### **Step 4: Build and Publish (Optional)**

To share your widget library, you can publish it on **npm** or keep it as a local dependency.

If you want to **publish** the package:

1. Add "main": "dist/index.js" to package.json.
2. Run npm run build to create the bundle.
3. Publish it to npm using npm publish.

For local development, you can link the library using npm link or add the local path in the package.json of the main app.

### ****2️⃣ Main Dashboard Web App Project****

This is the main web app that integrates the **Widget Library** and connects to the **ThingsBoard backend APIs**.

#### **Step 1: Create the Dashboard Web App Project**

1. **Create a React + TypeScript project**:

bash

CopyEdit

npx create-react-app thingsboard-dashboard --template typescript

cd thingsboard-dashboard

1. **Install dependencies**:

bash

CopyEdit

npm install redux react-redux react-router-dom axios @reduxjs/toolkit

#### **Step 2: Setup Redux Store**

Create a redux folder to manage app state.

1. **Define the store** in redux/store.ts:

tsx

CopyEdit

import { configureStore } from '@reduxjs/toolkit';

import widgetReducer from './widgetSlice';

const store = configureStore({

reducer: {

widget: widgetReducer

}

});

export default store;

1. **Create a slice for widget state management**:

tsx

CopyEdit

// redux/widgetSlice.ts

import { createSlice } from '@reduxjs/toolkit';

const widgetSlice = createSlice({

name: 'widget',

initialState: {

widgets: []

},

reducers: {

setWidgets: (state, action) => {

state.widgets = action.payload;

}

}

});

export const { setWidgets } = widgetSlice.actions;

export default widgetSlice.reducer;

#### **Step 3: Setup Routing**

1. Create routing structure in App.tsx:

tsx

CopyEdit

// src/App.tsx

import React from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import Dashboard from './pages/Dashboard';

import Settings from './pages/Settings';

import { Provider } from 'react-redux';

import store from './redux/store';

import ThemeProvider from './components/themes/ThemeProvider';

function App() {

return (

<Provider store={store}>

<ThemeProvider>

<Router>

<Routes>

<Route path="/" element={<Dashboard />} />

<Route path="/settings" element={<Settings />} />

</Routes>

</Router>

</ThemeProvider>

</Provider>

);

}

export default App;

#### **Step 4: Create Pages and Components**

1. **Dashboard Page** (Dashboard.tsx):

tsx

CopyEdit

// src/pages/Dashboard.tsx

import React, { useEffect } from 'react';

import { useDispatch } from 'react-redux';

import { setWidgets } from '../redux/widgetSlice';

import ChartWidget from 'my-widget-library'; // Import widget from library

const Dashboard: React.FC = () => {

const dispatch = useDispatch();

useEffect(() => {

// Simulate an API call to fetch widget data

const fetchWidgets = async () => {

const response = await fetch('https://your-thingsboard-api.com/widgets');

const data = await response.json();

dispatch(setWidgets(data));

};

fetchWidgets();

}, [dispatch]);

return (

<div>

<h1>Dashboard</h1>

<div className="widget-container">

{/\* Render widgets dynamically \*/}

<ChartWidget title="Sample Chart" data={[10, 20, 30]} />

</div>

</div>

);

};

export default Dashboard;

1. **Settings Page** (Settings.tsx):

tsx

CopyEdit

// src/pages/Settings.tsx

import React from 'react';

const Settings: React.FC = () => {

return (

<div>

<h1>Settings</h1>

{/\* Add Settings UI here \*/}

</div>

);

};

export default Settings;

#### **Step 5: API Integration**

1. Set up API calls using axios or fetch. For ThingsBoard, you'll need to integrate **REST** or **GraphQL** endpoints.

Example for fetching widgets:

tsx

CopyEdit

// Fetch data from ThingsBoard API

const fetchWidgets = async () => {

const response = await axios.get('https://thingsboard.example.com/api/widgets');

const widgets = response.data;

dispatch(setWidgets(widgets));

};

#### **Step 6: Themes**

1. Create a **ThemeProvider** component to switch between light and dark mode using context or styled-components.

Example of ThemeProvider.tsx:

tsx

CopyEdit

// src/components/themes/ThemeProvider.tsx

import React, { createContext, useContext, useState } from 'react';

const ThemeContext = createContext({});

export const ThemeProvider: React.FC = ({ children }) => {

const [theme, setTheme] = useState('light');

const toggleTheme = () => {

setTheme((prev) => (prev === 'light' ? 'dark' : 'light'));

};

return (

<ThemeContext.Provider value={{ theme, toggleTheme }}>

{children}

</ThemeContext.Provider>

);

};

export const useTheme = () => useContext(ThemeContext);

### ****3️⃣ Running the Complete Project****

1. **Running the Widget Library** (in development mode):
   * npm start inside my-widget-library to serve the widget library for testing.
2. **Running the Dashboard Web App**:
   * Inside the thingsboard-dashboard folder, run:

bash

CopyEdit

npm start

1. **Integrate with Backend**:
   * Make sure the **API URLs** in your app point to the **ThingsBoard** backend, such as /api/widgets and /api/dashboards.

### ****🔗 Summary****

* The **widget library** is a **separate React project** that is imported into the **main dashboard app**.
* The **main dashboard app** uses **Redux** for state management, integrates with **ThingsBoard backend APIs**, and handles **widget rendering**.
* The **theme system** can be toggled between light and dark mode, and widgets are dynamically loaded based on the API data.

Would you like me to assist with specific parts of this setup, or would you prefer more detailed examples of any part?

To help you convert the **ThingsBoard widget system** into **React + TypeScript** components, I'll break down the relevant widget components and provide detailed examples for each. This will include the following parts from the **ThingsBoard UI-ngx repo**:

1. **Widget Library**: List of available widgets.
2. **Widget Types**: Different types of widgets (charts, tables, etc.).
3. **Widget Core**: Logic for rendering and handling widget states.
4. **Widget Settings & Configuration**: Customizing widgets and saving settings.
5. **Widget Editor**: Interface for configuring widgets.

**1️⃣ Widget Library - WidgetLibrary.tsx**

This component will hold a list of widgets available for use.

tsx

CopyEdit

// src/components/widgets/WidgetLibrary.tsx

import React from 'react';

import { WidgetTypes } from './WidgetTypes'; // Import Widget Types from the widget types file

import { useDispatch } from 'react-redux';

import { addWidget } from '../../redux/widgetSlice'; // Assuming you have a Redux slice for widgets

const WidgetLibrary: React.FC = () => {

const dispatch = useDispatch();

const handleAddWidget = (widgetType: string) => {

// Dispatch action to add a widget to the dashboard

dispatch(addWidget({ widgetType }));

};

return (

<div className="widget-library">

<h3>Widget Library</h3>

<div className="widget-list">

{Object.keys(WidgetTypes).map((widgetType) => (

<div

key={widgetType}

className="widget-item"

onClick={() => handleAddWidget(widgetType)}

>

<h4>{WidgetTypes[widgetType].name}</h4>

<p>{WidgetTypes[widgetType].description}</p>

</div>

))}

</div>

</div>

);

};

export default WidgetLibrary;

**2️⃣ Widget Types - WidgetTypes.ts**

This file defines different widget types and their associated properties. You can add various types of widgets here, such as charts, tables, etc.

tsx

CopyEdit

// src/components/widgets/WidgetTypes.ts

export const WidgetTypes = {

chart: {

name: 'Chart Widget',

description: 'A widget to display charts (e.g., bar, line, pie)',

component: 'ChartWidget',

},

table: {

name: 'Table Widget',

description: 'A widget to display tabular data',

component: 'TableWidget',

},

// Add more widget types as needed

};

**3️⃣ Widget Core - WidgetCore.tsx**

This component defines the core behavior of a widget. It would manage properties like data binding, rendering, etc.

tsx

CopyEdit

// src/components/widgets/WidgetCore.tsx

import React, { useEffect, useState } from 'react';

interface WidgetCoreProps {

widgetType: string;

data: any[];

}

const WidgetCore: React.FC<WidgetCoreProps> = ({ widgetType, data }) => {

const [widgetData, setWidgetData] = useState<any[]>(data);

useEffect(() => {

// Logic to fetch or update widget data

setWidgetData(data);

}, [data]);

switch (widgetType) {

case 'chart':

return <div>{/\* Render Chart here \*/}</div>;

case 'table':

return <div>{/\* Render Table here \*/}</div>;

// Add more cases for other widget types

default:

return <div>No Widget Type Found</div>;

}

};

export default WidgetCore;

**4️⃣ Widget Settings & Configuration - WidgetConfiguration.tsx**

This component will handle the configuration of widgets, including customization of their appearance and behavior.

tsx

CopyEdit

// src/components/widgets/WidgetConfiguration.tsx

import React, { useState } from 'react';

interface WidgetConfigurationProps {

widgetType: string;

currentSettings: any;

onSaveSettings: (newSettings: any) => void;

}

const WidgetConfiguration: React.FC<WidgetConfigurationProps> = ({

widgetType,

currentSettings,

onSaveSettings,

}) => {

const [settings, setSettings] = useState(currentSettings);

const handleSaveSettings = () => {

// Save the new settings (could involve dispatching a Redux action)

onSaveSettings(settings);

};

return (

<div className="widget-config">

<h3>Configure {widgetType}</h3>

{/\* Form fields to configure widget settings \*/}

<div>

<label>

Widget Title

<input

type="text"

value={settings.title}

onChange={(e) => setSettings({ ...settings, title: e.target.value })}

/>

</label>

</div>

<div>

<button onClick={handleSaveSettings}>Save Settings</button>

</div>

</div>

);

};

export default WidgetConfiguration;

**5️⃣ Widget Editor - WidgetEditor.tsx**

This component provides an interface to edit widget properties, such as the title, description, and other settings.

tsx

CopyEdit

// src/components/widgets/WidgetEditor.tsx

import React, { useState } from 'react';

import { WidgetConfiguration } from './WidgetConfiguration';

import { WidgetCore } from './WidgetCore';

const WidgetEditor: React.FC = () => {

const [widgetType, setWidgetType] = useState('chart');

const [settings, setSettings] = useState({ title: 'My Chart Widget' });

const [data, setData] = useState([1, 2, 3, 4, 5]); // Example data

const handleSaveSettings = (newSettings: any) => {

setSettings(newSettings);

};

return (

<div className="widget-editor">

<h2>Edit Widget</h2>

<WidgetConfiguration

widgetType={widgetType}

currentSettings={settings}

onSaveSettings={handleSaveSettings}

/>

<WidgetCore widgetType={widgetType} data={data} />

</div>

);

};

export default WidgetEditor;

**6️⃣ Redux Integration - widgetSlice.ts**

This slice will handle the addition of widgets and their configurations using Redux.

tsx

CopyEdit

// src/redux/widgetSlice.ts

import { createSlice, PayloadAction } from '@reduxjs/toolkit';

interface WidgetState {

widgets: { type: string; settings: any }[];

}

const initialState: WidgetState = {

widgets: [],

};

const widgetSlice = createSlice({

name: 'widget',

initialState,

reducers: {

addWidget: (state, action: PayloadAction<{ widgetType: string }>) => {

state.widgets.push({ type: action.payload.widgetType, settings: {} });

},

updateWidgetSettings: (state, action: PayloadAction<{ index: number; settings: any }>) => {

state.widgets[action.payload.index].settings = action.payload.settings;

},

},

});

export const { addWidget, updateWidgetSettings } = widgetSlice.actions;

export default widgetSlice.reducer;

**7️⃣ Complete Dashboard Example - Dashboard.tsx**

tsx

CopyEdit

// src/pages/Dashboard.tsx

import React from 'react';

import { useDispatch, useSelector } from 'react-redux';

import { WidgetLibrary } from '../components/widgets/WidgetLibrary';

import { WidgetCore } from '../components/widgets/WidgetCore';

import { addWidget } from '../redux/widgetSlice';

const Dashboard: React.FC = () => {

const dispatch = useDispatch();

const widgets = useSelector((state: any) => state.widget.widgets);

const handleAddWidget = (widgetType: string) => {

dispatch(addWidget({ widgetType }));

};

return (

<div>

<h1>Dashboard</h1>

<WidgetLibrary onAddWidget={handleAddWidget} />

<div className="widget-container">

{widgets.map((widget: any, index: number) => (

<WidgetCore key={index} widgetType={widget.type} data={[]} />

))}

</div>

</div>

);

};

export default Dashboard;

**8️⃣ Running the Complete Project**

1. **Widget Library Project**:
   * Run the widget library in development mode: npm start inside the my-widget-library folder.
2. **Dashboard App**:
   * Inside the thingsboard-dashboard folder, run:

bash

CopyEdit

npm start

**🔗 Backend Integration**

To integrate with the **ThingsBoard backend**:

1. Use **REST API** calls or **WebSocket** for real-time data integration.
2. Implement the necessary API calls in the **Dashboard App** to fetch widgets and dashboard configurations.

Would you like help with connecting to the **ThingsBoard API** or with specific integrations? Let me know!

To render the **Chart Widget** using **Highcharts** in your React + TypeScript application, we need to replace the placeholder <div>{/\* Render Chart here \*/}</div> with actual code that uses the **Highcharts** library to display the chart. Below is the full setup and code to use **Highcharts** for rendering charts in your **React + TypeScript** app.

### ****Steps to Implement Highcharts in React****

1. **Install Highcharts and Highcharts React** First, you need to install highcharts and highcharts-react-official (the React wrapper for Highcharts).

bash

CopyEdit

npm install highcharts highcharts-react-official

1. **Creating the ChartWidget.tsx Component**

Now, let's create a complete chart widget using **Highcharts**.

tsx

CopyEdit

// src/components/widgets/ChartWidget.tsx

import React, { useState, useEffect } from 'react';

import Highcharts from 'highcharts';

import HighchartsReact from 'highcharts-react-official';

// Define the chart widget props

interface ChartWidgetProps {

data: number[]; // Data to be displayed in the chart

title: string; // Title of the chart

}

const ChartWidget: React.FC<ChartWidgetProps> = ({ data, title }) => {

const [chartOptions, setChartOptions] = useState<Highcharts.Options>({

chart: {

type: 'line', // Default chart type

},

title: {

text: title,

},

xAxis: {

categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul'], // Example months

},

series: [

{

name: 'Data Series',

data: data, // This will be dynamic

},

],

});

useEffect(() => {

// Example of updating the chart data dynamically (can be replaced with real data fetching)

setChartOptions((prevOptions) => ({

...prevOptions,

series: [

{

name: 'Updated Data Series',

data: data, // Dynamically updating the data

},

],

}));

}, [data]);

return (

<div className="chart-widget">

<HighchartsReact highcharts={Highcharts} options={chartOptions} />

</div>

);

};

export default ChartWidget;

### ****3️⃣ Integrating the**** ChartWidget.tsx ****into Your Dashboard****

You can now integrate the ChartWidget into your **Dashboard** or **Widget Editor** to render a chart with the given data.

#### Example Usage in a Dashboard or Widget Component

tsx

CopyEdit

// src/pages/Dashboard.tsx

import React, { useState } from 'react';

import ChartWidget from '../components/widgets/ChartWidget'; // Import the ChartWidget component

const Dashboard: React.FC = () => {

const [chartData, setChartData] = useState([1, 3, 2, 5, 6, 4, 7]); // Example data

return (

<div>

<h1>Dashboard</h1>

<div className="widget-container">

<ChartWidget title="Sales Overview" data={chartData} />

</div>

</div>

);

};

export default Dashboard;

### ****4️⃣ Highcharts Configuration (ChartWidget)****

To make this chart customizable, you can update the ChartWidget.tsx component to support multiple configurations such as different chart types, dynamic axes, etc.

Here’s a more advanced configuration for **Highcharts** with additional options.

tsx

CopyEdit

// src/components/widgets/ChartWidget.tsx

import React, { useState, useEffect } from 'react';

import Highcharts from 'highcharts';

import HighchartsReact from 'highcharts-react-official';

interface ChartWidgetProps {

data: number[];

title: string;

chartType?: string; // Optional, can be 'line', 'bar', etc.

categories?: string[]; // Optional categories for x-axis

}

const ChartWidget: React.FC<ChartWidgetProps> = ({

data,

title,

chartType = 'line',

categories = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul'],

}) => {

const [chartOptions, setChartOptions] = useState<Highcharts.Options>({

chart: {

type: chartType, // Dynamic chart type

},

title: {

text: title,

},

xAxis: {

categories: categories, // Dynamic categories

},

series: [

{

name: 'Data Series',

data: data,

},

],

});

useEffect(() => {

setChartOptions((prevOptions) => ({

...prevOptions,

series: [

{

name: 'Updated Data Series',

data: data,

},

],

}));

}, [data]);

return (

<div className="chart-widget">

<HighchartsReact highcharts={Highcharts} options={chartOptions} />

</div>

);

};

export default ChartWidget;

### ****5️⃣ Final Integration in a React Dashboard****

Now that you have the ChartWidget, you can render this widget in your **Dashboard** page. Below is an example of how you can integrate and display different widgets dynamically.

tsx

CopyEdit

// src/pages/Dashboard.tsx

import React, { useState } from 'react';

import ChartWidget from '../components/widgets/ChartWidget'; // Import ChartWidget

const Dashboard: React.FC = () => {

const [chartData, setChartData] = useState([5, 3, 7, 10, 12, 6, 8]); // Example chart data

return (

<div>

<h1>ThingsBoard Dashboard</h1>

<div className="dashboard-widgets">

<ChartWidget title="Sales Growth" data={chartData} chartType="line" categories={['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul']} />

</div>

</div>

);

};

export default Dashboard;

### ****6️⃣ Example Styling for Widgets (Optional)****

To style the chart widgets, you can add the following CSS (just for demonstration):

css

CopyEdit

/\* src/styles/widgetStyles.css \*/

.chart-widget {

width: 100%;

height: 400px;

margin: 20px 0;

background-color: #f4f4f4;

border-radius: 8px;

}

.dashboard-widgets {

display: flex;

flex-wrap: wrap;

gap: 20px;

justify-content: space-between;

}

.dashboard-widgets .chart-widget {

flex: 1 1 calc(33% - 20px);

}

### ****7️⃣ Highcharts Full Configuration Example****

Here’s an example of a more complex **Highcharts** configuration with multiple series, different chart types, and additional settings:

tsx

CopyEdit

// src/components/widgets/ChartWidget.tsx

import React, { useState, useEffect } from 'react';

import Highcharts from 'highcharts';

import HighchartsReact from 'highcharts-react-official';

interface ChartWidgetProps {

data: number[];

title: string;

chartType?: string;

categories?: string[];

}

const ChartWidget: React.FC<ChartWidgetProps> = ({

data,

title,

chartType = 'line',

categories = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul'],

}) => {

const [chartOptions, setChartOptions] = useState<Highcharts.Options>({

chart: {

type: chartType,

},

title: {

text: title,

},

xAxis: {

categories: categories,

},

yAxis: {

title: {

text: 'Value',

},

},

series: [

{

name: 'Data Series',

data: data,

type: chartType,

},

// Additional series can be added here if needed

],

tooltip: {

shared: true,

valueSuffix: ' units',

},

legend: {

layout: 'horizontal',

align: 'center',

verticalAlign: 'bottom',

},

});

useEffect(() => {

setChartOptions((prevOptions) => ({

...prevOptions,

series: [

{

name: 'Updated Data Series',

data: data,

},

],

}));

}, [data]);

return (

<div className="chart-widget">

<HighchartsReact highcharts={Highcharts} options={chartOptions} />

</div>

);

};

export default ChartWidget;

### ****Next Steps:****

1. **Test the Widgets**: You can now integrate this widget code into your larger dashboard and test the widget rendering.
2. **Dynamic Data**: Update the data prop in real-time to fetch actual data using **API calls** (e.g., **ThingsBoard backend APIs** for dynamic dashboard data).
3. **Advanced Configurations**: Add more interactivity, like different chart types, tooltips, data points, and chart styles.

Let me know if you need further integration with the **ThingsBoard backend** or additional configurations for the widgets!