X-Dashboard Documentation V1.0

The XDashboard plugin is a powerful and versatile Dashboard component plugin designed for seamless development. It offers a wide range of features to enhance dashboard display and manipulation while cutting down development time by more than 50%.

Quick Start:

To utilize XDashboard:

1. Follow the instructions for installing the package provided by the admin document.
2. Once the package is installed, you can directly import any XDashboard component like any other ESM import.

Feature Configuration:

XDashboard comes jam packed with a bunch of components to cut down development time, and so are the options to customize them as per your requirements.

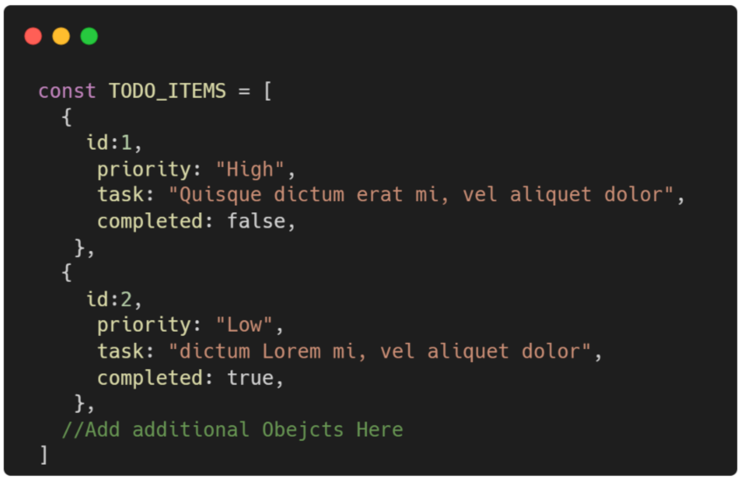
TODO:

XDashboard comes loaded with a To-Do's component for managing your important task along with the ability to not just see but also change and update priority, edit details and marking them as completed.

ToDo's take object array as an input for showing data, the object has following props:

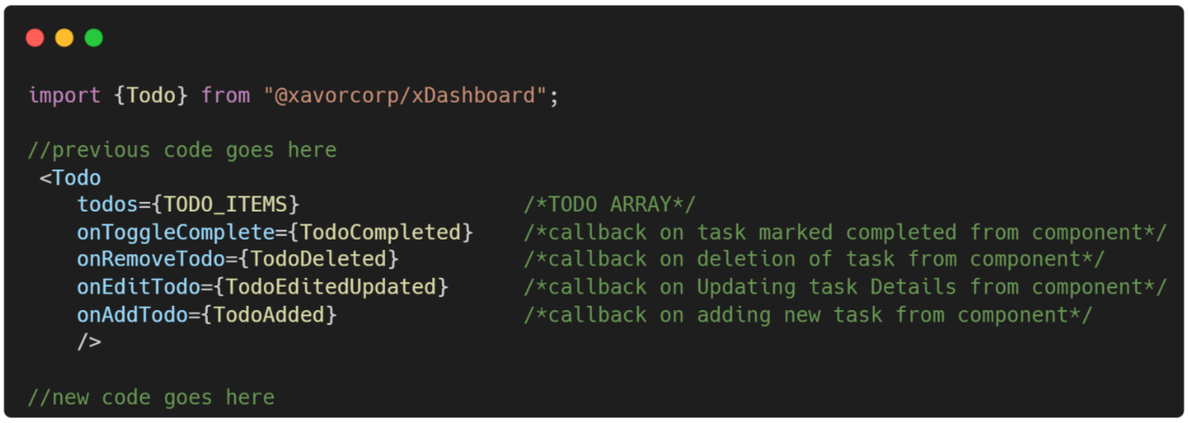
Props:	Type:	Values:	Description:
Id	Number	Unique integer	Should be unique id
Task	String	String value	The task to be added in list
Completed	Boolean	True/False	When the task is marked completed
Priority	String	Medium/High/Low	The task priority assigned to each task item

Input Data Sample:



```
const TODO_ITEMS = [
  {
    id:1,
     priority: "High",
     task: "Quisque dictum erat mi, vel aliquet dolor",
     completed: false,
   },
  {
    id:2,
     priority: "Low",
     task: "dictum Lorem mi, vel aliquet dolor",
     completed: true,
   },
  //Add additional Obejcts Here
]
```

Usage Example:



```
import {Todo} from "@xavorcorp/xDashboard";

//previous code goes here
 <Todo
    todos={TODO_ITEMS}                      /*TODO ARRAY*/
    onToggleComplete={TodoCompleted}        /*callback on task marked completed from component*/
    onRemoveTodo={TodoDeleted}              /*callback on deletion of task from component*/
    onEditTodo={TodoEditedUpdated}          /*callback on Updating task Details from component*/
    onAddTodo={TodoAdded}                   /*callback on adding new task from component*/
    />

//new code goes here
```

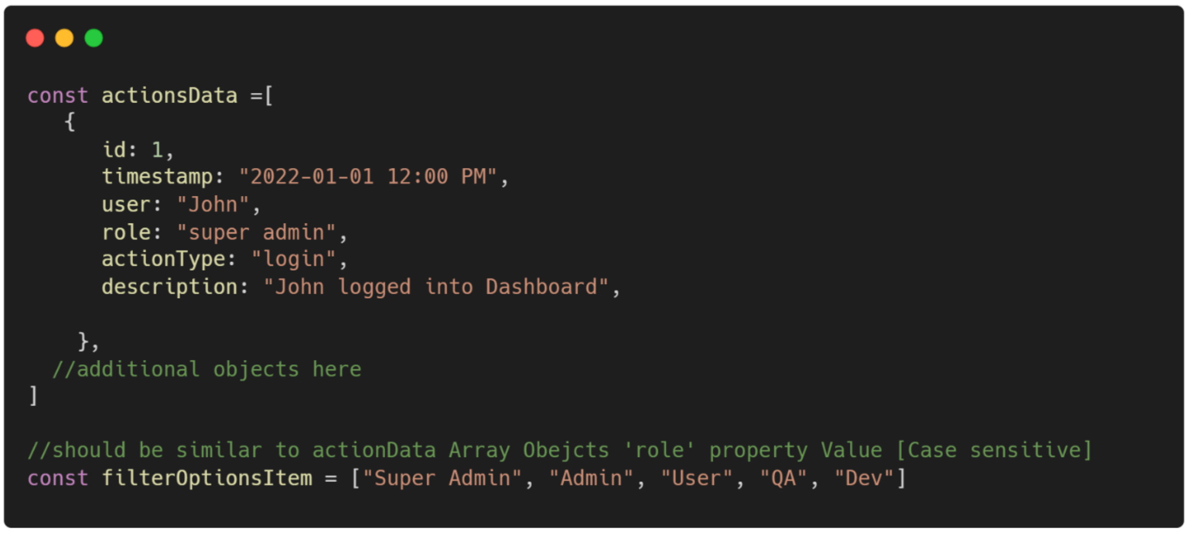
Action Logger:

XDashboard Action Logger component allows for keeping track of your important system logs along with the ability to not just see but also categories and filter data on multiple parameters and marking them as completed.

Actions Logger take object array as an input for showing data, the object has following props:

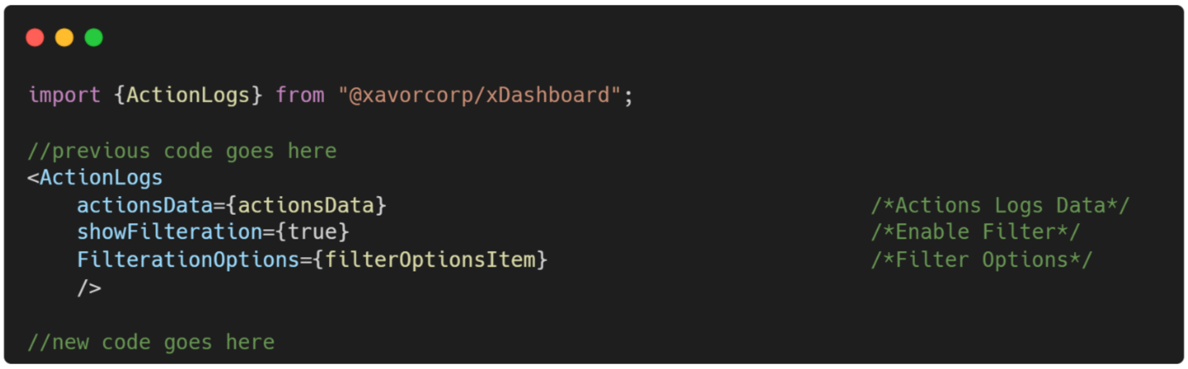
Props:	Type:	Values:	Description:
Id	Number	Unique integer	Should be unique id
timestamp	String	Date Time Stamp	The Date and time of the event
user	String	String	Which user performed the task
role	String	Users Role	The user’s role in the system
actionType	String	Users Action	The action category triggered
description	String	Action Log	Log statement

Input Sample:



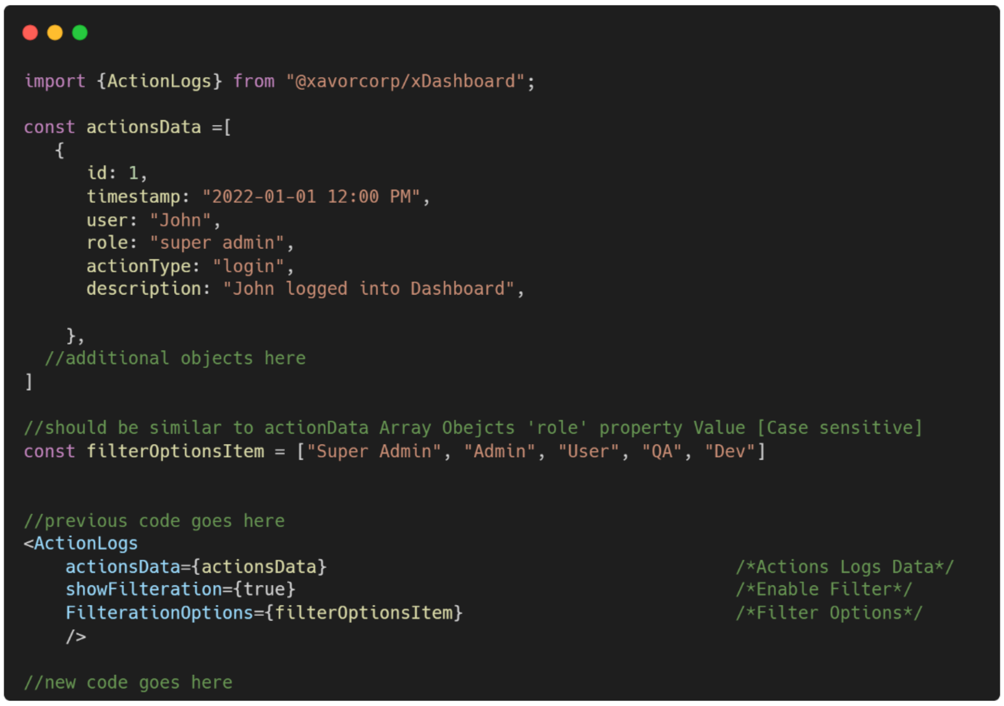
```
const actionsData =[
   {
      id: 1,
      timestamp: "2022-01-01 12:00 PM",
      user: "John",
      role: "super admin",
      actionType: "login",
      description: "John logged into Dashboard",

    },
  //additional objects here
]

//should be similar to actionData Array Obejcts 'role' property Value [Case sensitive]
const filterOptionsItem = ["Super Admin", "Admin", "User", "QA", "Dev"]
```

Usage Example:



```
import {ActionLogs} from "@xavorcorp/xDashboard";

//previous code goes here
<ActionLogs
    actionsData={actionsData}                                       /*Actions Logs Data*/
    showFilteration={true}                                          /*Enable Filter*/
    FilterationOptions={filterOptionsItem}                          /*Filter Options*/
    />

//new code goes here
```

Complete Code Snippet:



```
import {ActionLogs} from "@xavorcorp/xDashboard";

const actionsData =[
   {
      id: 1,
      timestamp: "2022-01-01 12:00 PM",
      user: "John",
      role: "super admin",
      actionType: "login",
      description: "John logged into Dashboard",

    },
  //additional objects here
]

//should be similar to actionData Array Obejcts 'role' property Value [Case sensitive]
const filterOptionsItem = ["Super Admin", "Admin", "User", "QA", "Dev"]


//previous code goes here
<ActionLogs
    actionsData={actionsData}                                    /*Actions Logs Data*/
    showFilteration={true}                                       /*Enable Filter*/
    FilterationOptions={filterOptionsItem}                       /*Filter Options*/
    />

//new code goes here
```

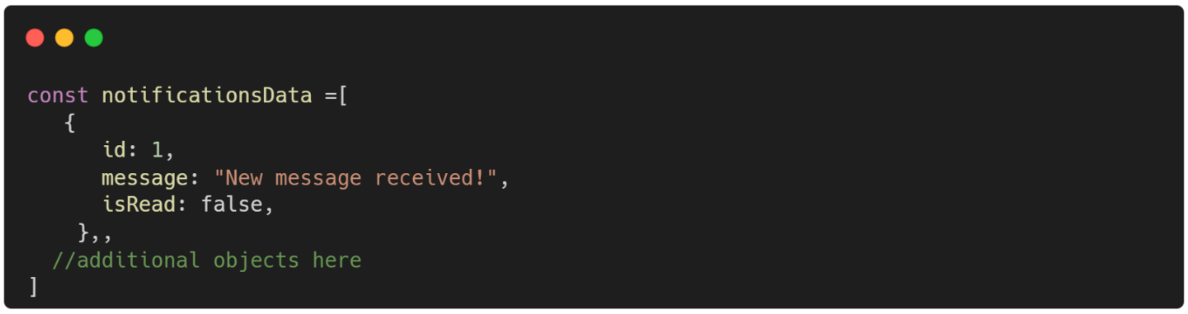
Notifications:

XDashboard Notifications component allows for keeping track of your important Notifications along with the ability to not just see but also mark them as read, and remove them along with badge indicators, indicating unread notifications.

notifications take object array as an input for showing data, the object has following props:

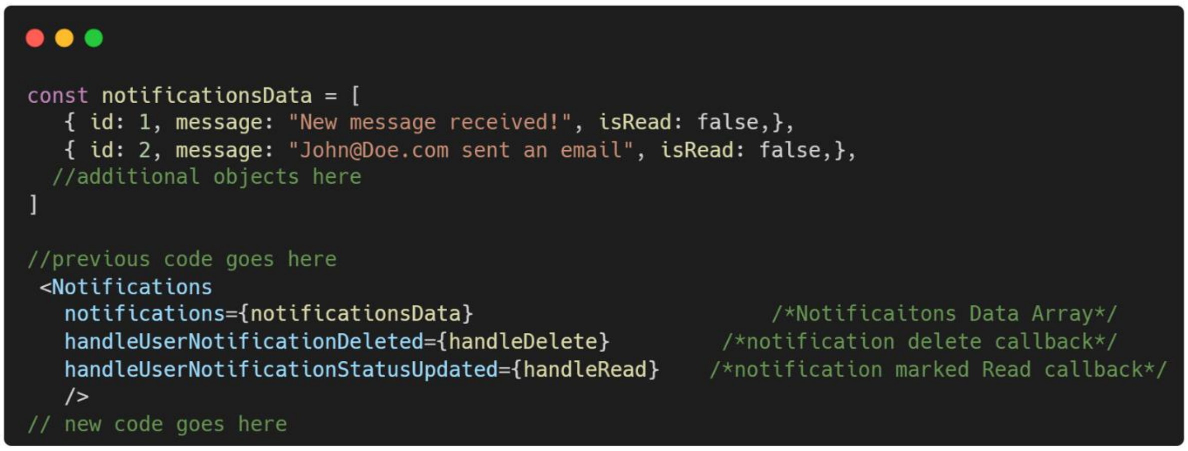
Props:	Type:	Values:	Description:
Id	Number	Unique integer	Should be unique id
Message	String	Notification	The notification message
isRead	Boolean	True OR False	Mark notification as read

Sample Input:



```
const notificationsData =[
   {
      id: 1,
      message: "New message received!",
      isRead: false,
    },,
  //additional objects here
]
```

Usage Example:



```
const notificationsData = [
   { id: 1, message: "New message received!", isRead: false,},
   { id: 2, message: "John@Doe.com sent an email", isRead: false,},
  //additional objects here
]

//previous code goes here
 <Notifications
   notifications={notificationsData}                          /*Notificaitons Data Array*/
   handleUserNotificationDeleted={handleDelete}           /*notification delete callback*/
   handleUserNotificationStatusUpdated={handleRead}    /*notification marked Read callback*/
   />
// new code goes here
```

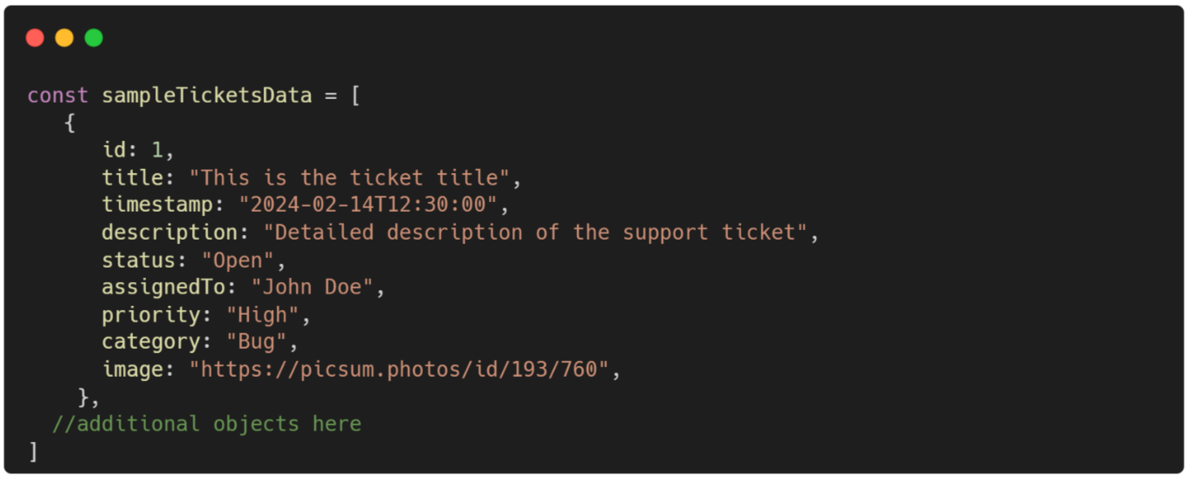
Support Tickets:

XDashboard Support Tickets component allows for keeping track of your tickets along with the ability to not just see but also mark them as resolved or remove them along with filter options on ticket list.

Support Tickets Component take object array as an input for showing data, the object has following props:

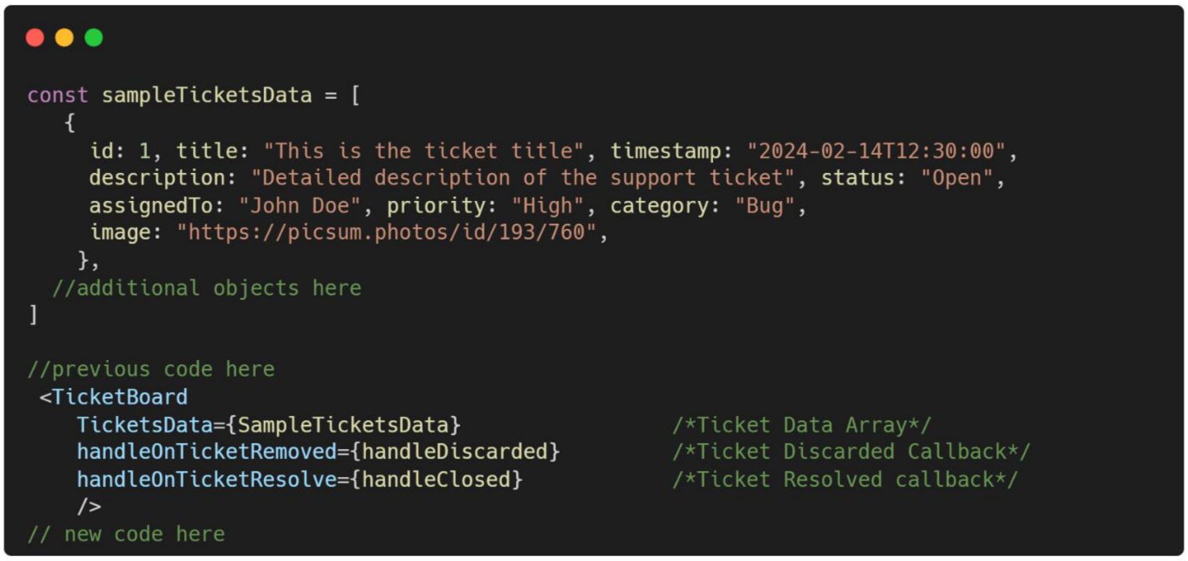
Props:	Type:	Values:	Description:
Id	Number	Unique integer	Should be unique id
timestamp	String	Date Time Stamp	The Date and time of the event
Description	String	String	Issue details
Status	String	Ticket status	Ticket open or closed
assignedTo	String	Username	Support providing user
priority	String	Priority	Priority of tickets
category	String	ticket Category	Category assigned to ticket
image	String	Image path/url	Path of the attached image or url

Sample Input:



```
const sampleTicketsData = [
   {
      id: 1,
      title: "This is the ticket title",
      timestamp: "2024-02-14T12:30:00",
      description: "Detailed description of the support ticket",
      status: "Open",
      assignedTo: "John Doe",
      priority: "High",
      category: "Bug",
      image: "https://picsum.photos/id/193/760",
    },
  //additional objects here
]
```

Usage Example:



```
const sampleTicketsData = [
   {
     id: 1, title: "This is the ticket title", timestamp: "2024-02-14T12:30:00",
     description: "Detailed description of the support ticket", status: "Open",
     assignedTo: "John Doe", priority: "High", category: "Bug",
     image: "https://picsum.photos/id/193/760",
    },
  //additional objects here
]

//previous code here
 <TicketBoard
    TicketsData={SampleTicketsData}                /*Ticket Data Array*/
    handleOnTicketRemoved={handleDiscarded}        /*Ticket Discarded Callback*/
    handleOnTicketResolve={handleClosed}           /*Ticket Resolved callback*/
    />
// new code here
```

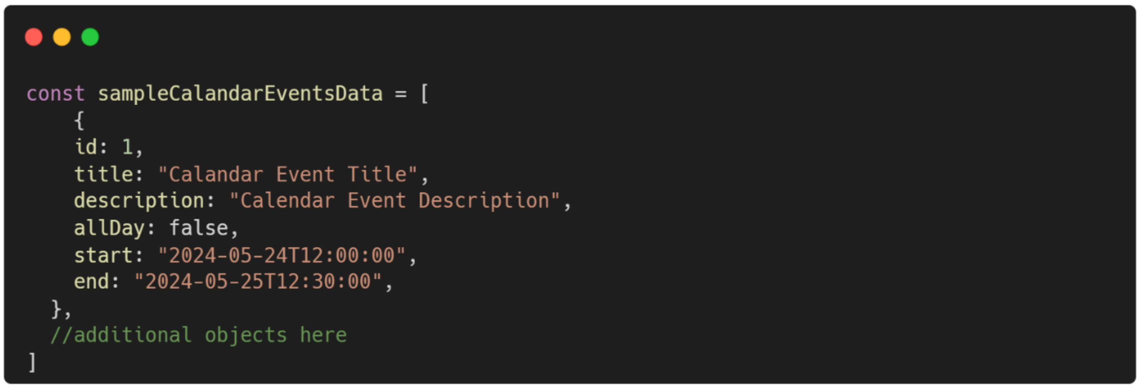
Calendar:

XDashboard Calendar component allows for keeping track of your Events along with the ability to Summarize all events, which can be toggled to or hide based on preference, along with the option to show/hide weekends, not just that but also, options for changing calendar views into monthly, daily, weekly, and event list format, along with the added ability to quickly jump onto present day, in case you got lost into dates, and last but not the least, the ability to quickly add or manipulate calendar events by clicking, dragging and dropping.

Calendar Component take object array as an input for showing data, the object has following props:

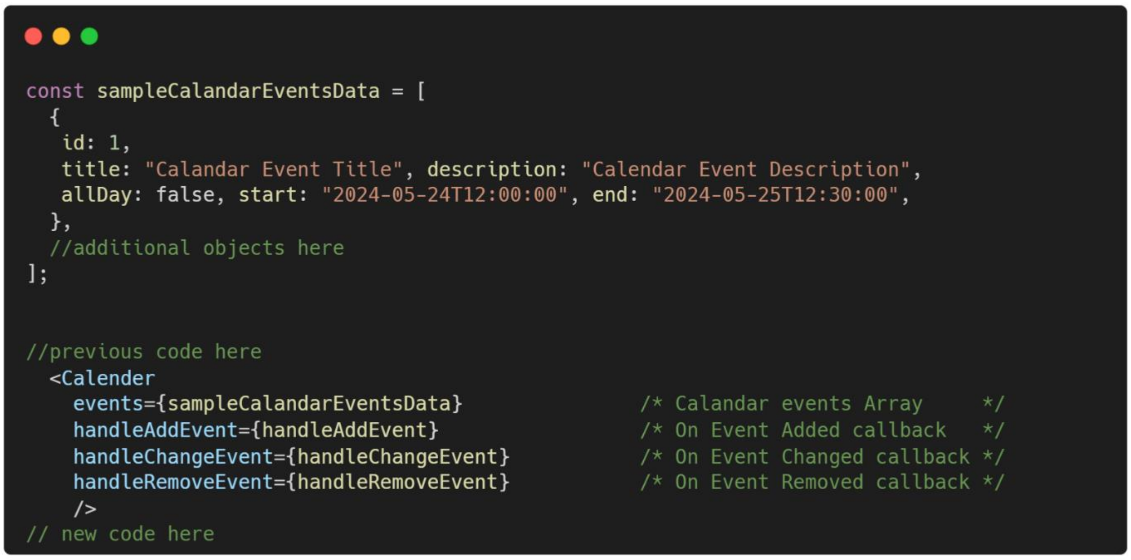
Props:	Type:	Purpose:	Description:
Id	string	Event identity	Should be unique id
description	String	Event Details	The details of event
start	String	Starting Date	Event Start date defined
startTime	String	Starting time slot	Event Starts at the time defined
end	String	Ending date	Event End Date defined
endTime	String	Ending time slot	Event ends at the time defined
allDay	boolean	Full day Event	Is it a Complete day event
Title	String	Event Title	Event title description

Sample Input:



```
const sampleCalandarEventsData = [
    {
    id: 1,
    title: "Calandar Event Title",
    description: "Calendar Event Description",
    allDay: false,
    start: "2024-05-24T12:00:00",
    end: "2024-05-25T12:30:00",
  },
  //additional objects here
]
```

Example Usage:



```
const sampleCalandarEventsData = [
  {
   id: 1,
   title: "Calandar Event Title", description: "Calendar Event Description",
   allDay: false, start: "2024-05-24T12:00:00", end: "2024-05-25T12:30:00",
  },
  //additional objects here
];


//previous code here
  <Calender
    events={sampleCalandarEventsData}              /* Calandar events Array     */
    handleAddEvent={handleAddEvent}                /* On Event Added callback   */
    handleChangeEvent={handleChangeEvent}          /* On Event Changed callback */
    handleRemoveEvent={handleRemoveEvent}          /* On Event Removed callback */
    />
// new code here
```

Charts:

XDashboard provide more than eight charts out of the box which can be accessed by a chart Provider component:

- Bar Chart (horizontal, vertical, stacked)
- Line Chart

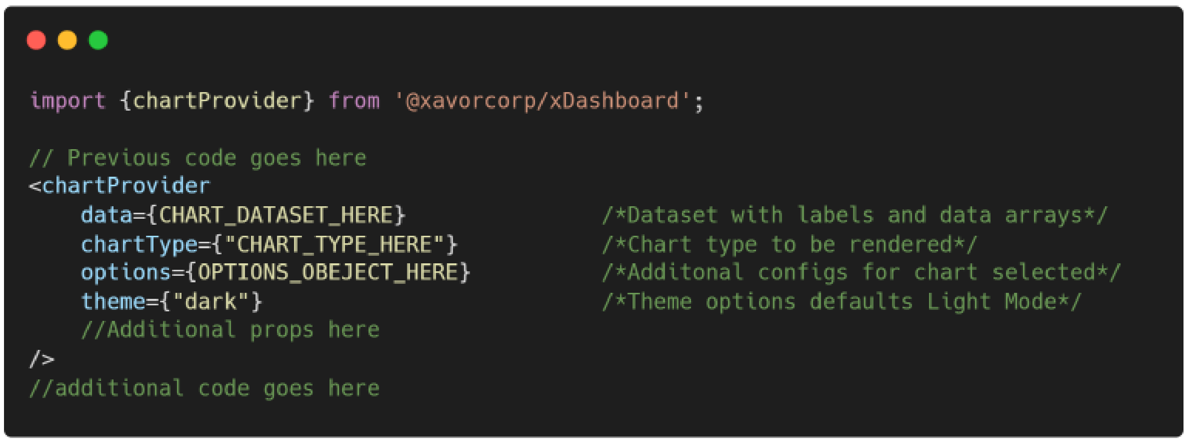
- Area Chart
- Scatter Chart
- Pie Chart
- Radar Chart
- Doughnut Chart
- Bubble Chart

Chart Provider:

Chart Provider requires four paramers, where data and chart-type are the two mandatory properties for rendering of charts, Additionally, options and theme properties can be specified:

- The ‘data’ parameters intakes Dataset of that need to be charted.
- ‘chartType’ specifies the type of chart to be rendered.
- ‘options’ property provides additional configurations for appearance and sub types of charts
- ‘theme’ property can be utilized for configurating light and dark theme of chart component on individual level.

Sample Usage:



```
import {chartProvider} from '@xavorcorp/xDashboard';

// Previous code goes here
<chartProvider
    data={CHART_DATASET_HERE}                /*Dataset with labels and data arrays*/
    chartType={"CHART_TYPE_HERE"}            /*Chart type to be rendered*/
    options={OPTIONS_OBEJECT_HERE}           /*Additonal configs for chart selected*/
    theme={"dark"}                           /*Theme options defaults Light Mode*/
    //Additional props here
/>
//additional code goes here
```

Each provider property can further be configured as follows:

Data Property:

data property is a required field and It should be structured as an object of two arrays **labels** and **datasets**.

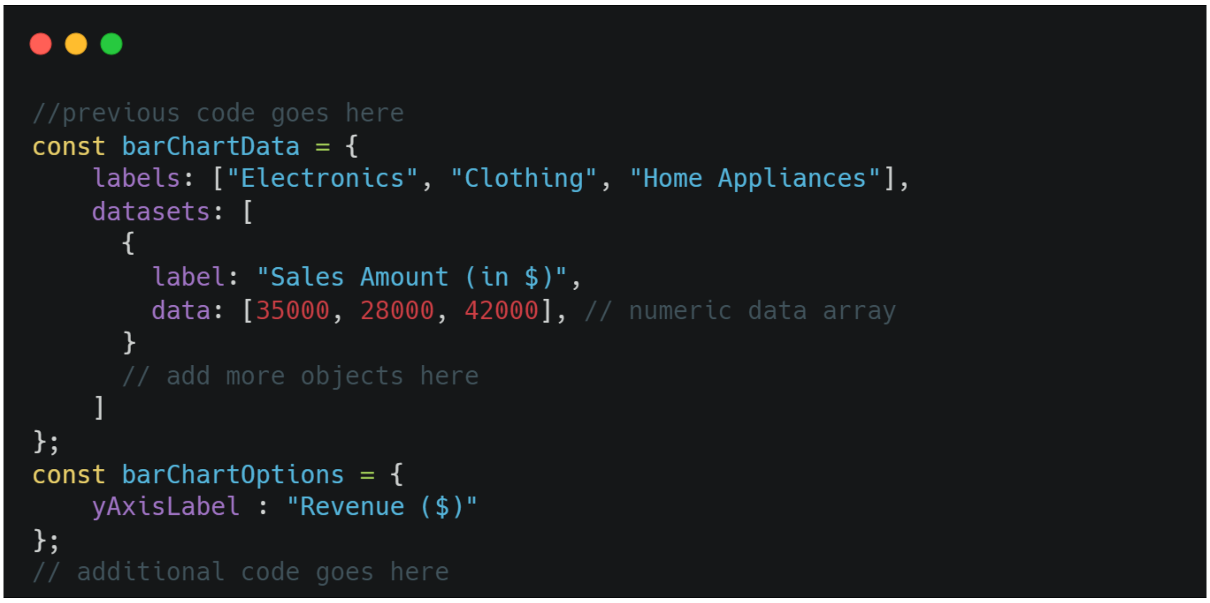
While structuring the data object:

- **datasets** array is required, while other fields can be made optional based on the chart being accessed.
- **datasets,** is an array of objects, where each object must have a two properties, data, and label.
    - **data** property will be an array of numeric values.
    - **label** property will act as legend for the charts.

- The Axes can be additionally specified for the data via options property of chart-Provider.
- The **labels** property of chart and **Data** property of Datasets array must have equal length otherwise labels will be cycled from starting index for the remaining data points.

Following examples shows how data should be passed to chart provider based on above key points.

Example:



```
//previous code goes here
const barChartData = {
    labels: ["Electronics", "Clothing", "Home Appliances"],
    datasets: [
      {
        label: "Sales Amount (in $)",
        data: [35000, 28000, 42000], // numeric data array
      }
      // add more objects here
    ]
};
const barChartOptions = {
    yAxisLabel : "Revenue ($)"
};
// additional code goes here
```

Chart-Type Property:

chartType property is mandatory for determining the type of chart.

Following table can be referred for choosing the right chart type input.

Chart	chartType value
Bar Chart	bar
Line Chart	line
Area Chart	area
Pie Chart	pie
Scatter Chart	scatter
Radar/Spider Chart	radar
Doughnut Chart	doughnut
Bubble Chart	bubble

Provider 'Options' Property configuration:

Options is an optional property that has following effect:

Options property provides two main accessibilities.

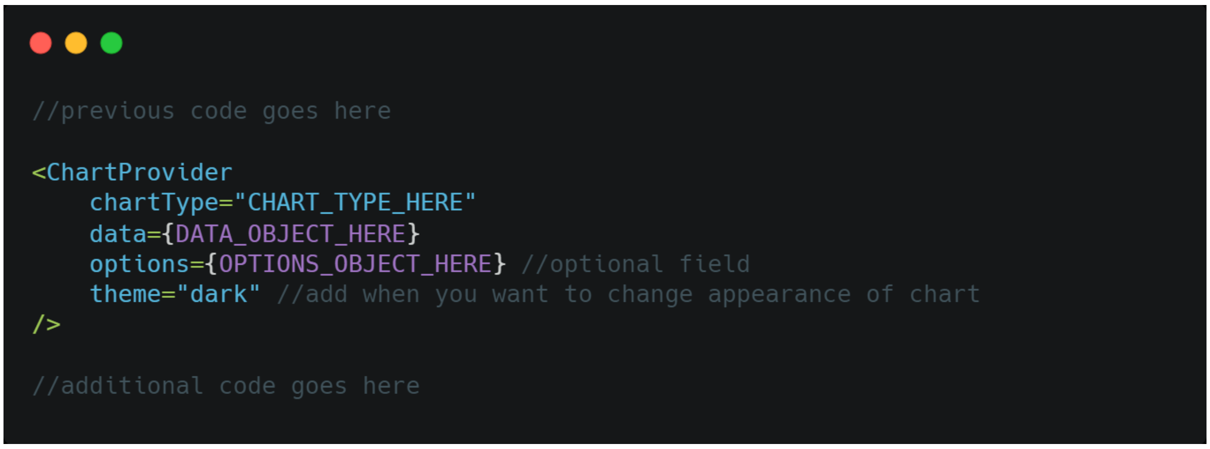
- **options** property is responsible for customization of chart. Chart customization can include customizing background, colour, border, border radius along with other comprehensive customizations, refer to the table below for available options.

- The **barChartType** property can be specified in options object which is responsible for displaying bar charts horizontally or vertically. To show stacked charts, the **barChartStacked** property can also be specified in options object.

options prop	**type**	**description**
barChartType	string	Shows the bars in Bar Charts in vertical or horizontal direction. Value of barChartType can be either **vertical** or **horizontal**. By default, bars are in vertical direction.
barChartStacked	boolean	Makes the datasets in Bar Chart stacked. The default value is **false**.
borderRadius	number	Changes radius of the border
backgroundColor	string, string[]	Changes the background color of chart
borderWidth	number	Changes the border width
fillColor	string, string[]	Changes the color of area in Area Chart
tension	float, number	Makes the line/border curve smooth
hoverRadius	number	Changes the radius on hover (usually used in bubble or scatter chart)
borderColor	string	Changes the color of border/line
xGridColor	string	Changes the color of grid line on x-axis
yGridColor	string	Changes the color of grid line on y-axis
hoverBackgroundColor	string	Changes the color of arc on hover (usually used in Doughnut and Pie Charts)
xTicksColor	string	Changes the color of ticks on x-axis
yTicksColor	string	Changes the color of ticks on y-axis
radius	number	Changes the size/radius of the bubble
xAxisLabel	string	Assigns label name on x-axis according to chart data
yAxisLabel	string	Assigns label name on y-axis according to chart data
barThickness	number	Changes the thickness of bar (used in Bar Charts)

Themes:

Chart provider provides both dark and light theme controls out of the box, which can be utilised by passing the theme props. Chart themes defaults as light, For dark theme, the ‘dark’ keyword can be passed as theme prop value.



```
//previous code goes here

<ChartProvider
    chartType="CHART_TYPE_HERE"
    data={DATA_OBJECT_HERE}
    options={OPTIONS_OBJECT_HERE} //optional field
    theme="dark" //add when you want to change appearance of chart
/>

//additional code goes here
```

Bar Charts:

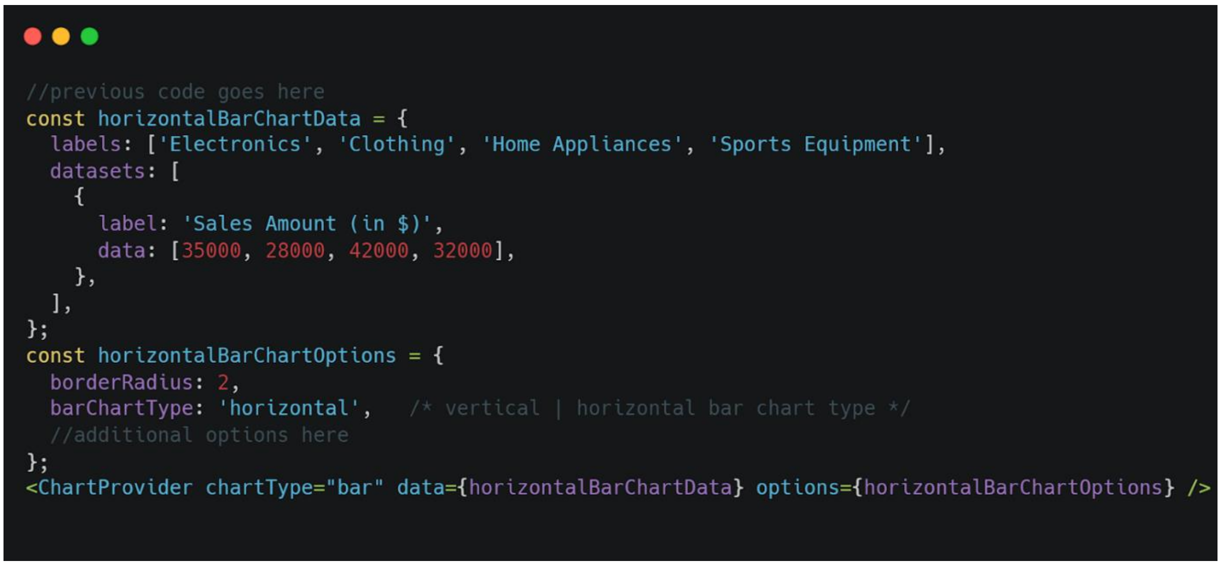
XDashboard provides four bar chart types out of the box:

- Horizontal Bar chart.
- Stacked Horizontal Bar chart.
- Vertical Bar chart.
- Stacked Vertical Bar chart.

Each bar chart can be accessed explicitly based on the options provided to the chart provider component.

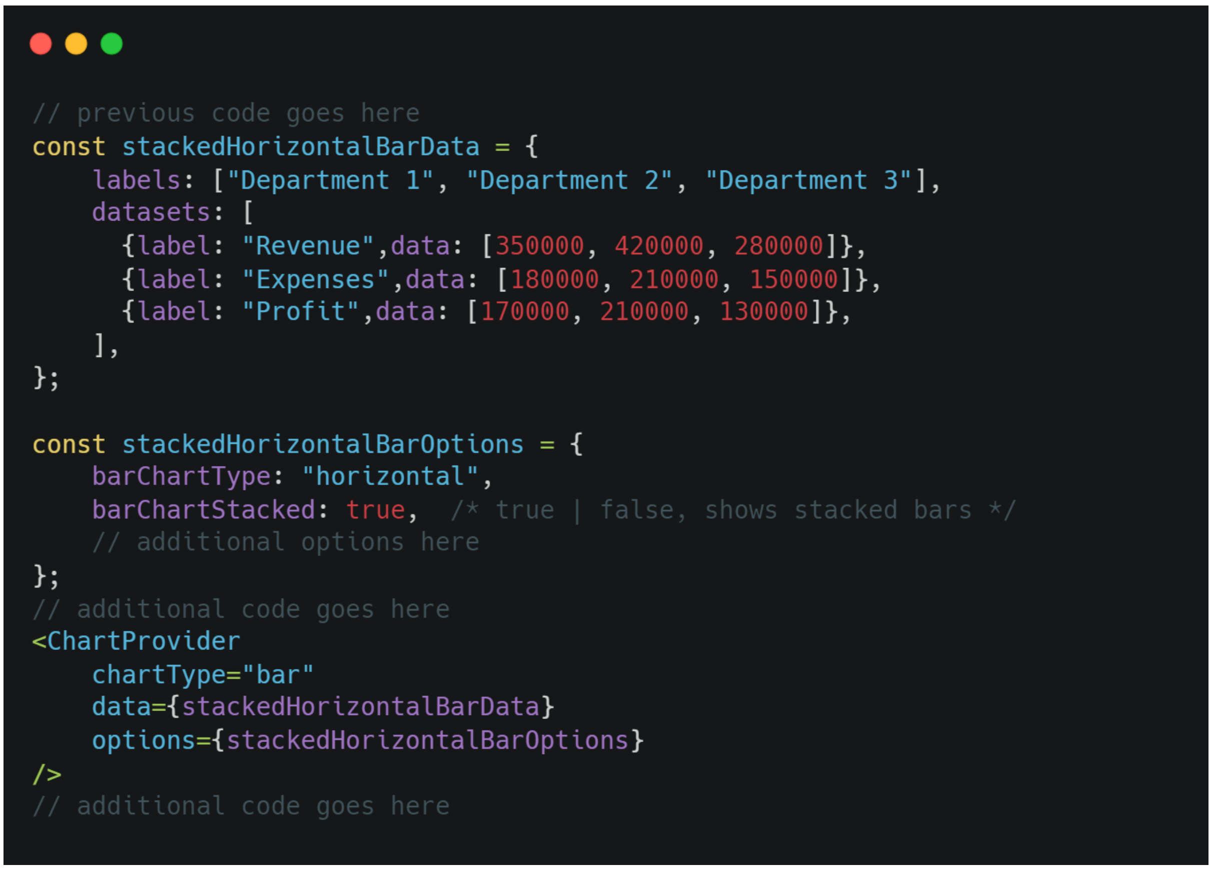
Following are examples of how each chart can be used in a react project.

Horizontal Bar chart:



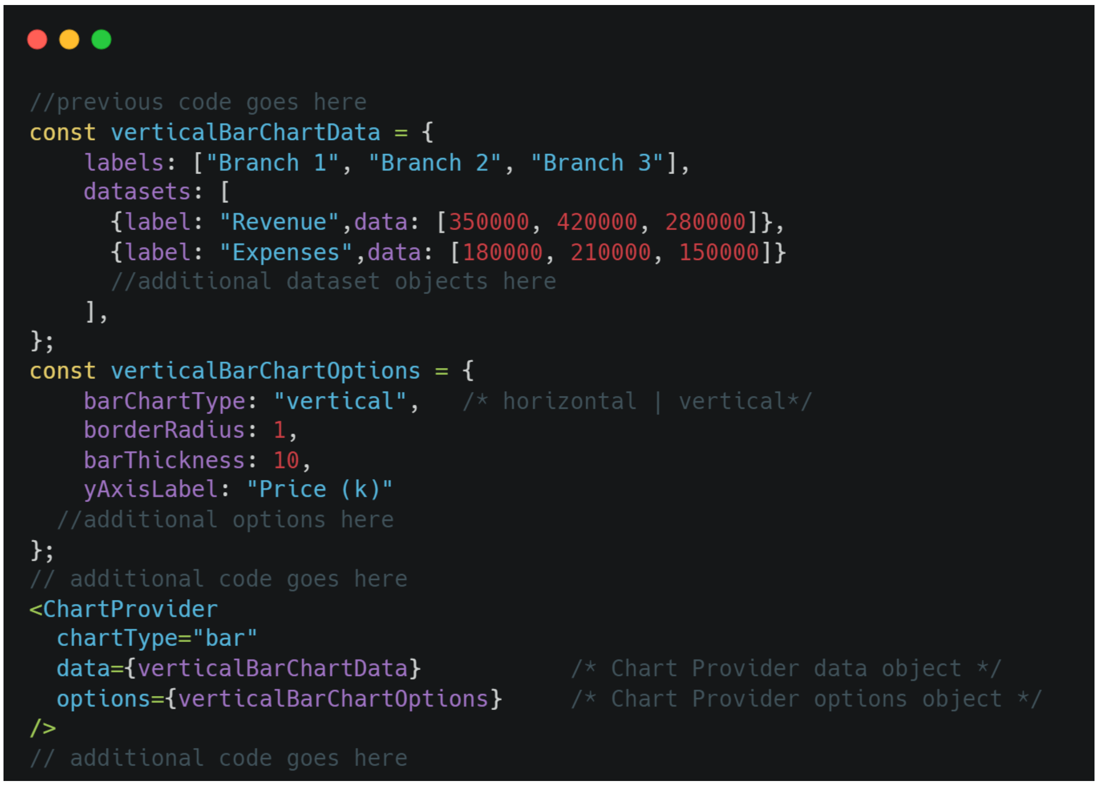
```
//previous code goes here
const horizontalBarChartData = {
  labels: ['Electronics', 'Clothing', 'Home Appliances', 'Sports Equipment'],
  datasets: [
    {
      label: 'Sales Amount (in $)',
      data: [35000, 28000, 42000, 32000],
    },
  ],
};
const horizontalBarChartOptions = {
  borderRadius: 2,
  barChartType: 'horizontal',   /* vertical | horizontal bar chart type */
  //additional options here
};
<ChartProvider chartType="bar" data={horizontalBarChartData} options={horizontalBarChartOptions} />
```

Stacked Horizontal Bar chart:



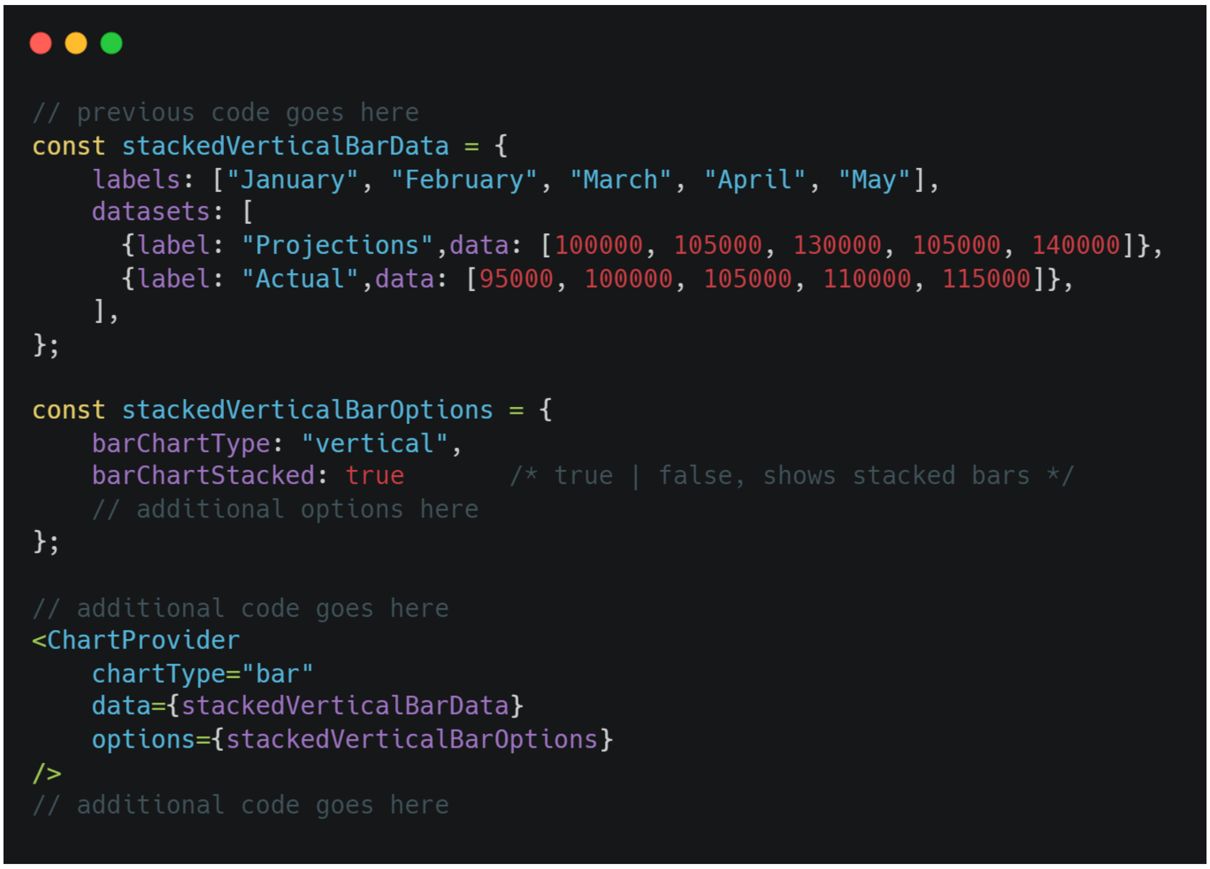
```
// previous code goes here
const stackedHorizontalBarData = {
    labels: ["Department 1", "Department 2", "Department 3"],
    datasets: [
      {label: "Revenue",data: [350000, 420000, 280000]},
      {label: "Expenses",data: [180000, 210000, 150000]},
      {label: "Profit",data: [170000, 210000, 130000]},
    ],
};

const stackedHorizontalBarOptions = {
    barChartType: "horizontal",
    barChartStacked: true,  /* true | false, shows stacked bars */
    // additional options here
};
// additional code goes here
<ChartProvider
    chartType="bar"
    data={stackedHorizontalBarData}
    options={stackedHorizontalBarOptions}
/>
// additional code goes here
```

Vertical Bar Chart:



```
//previous code goes here
const verticalBarChartData = {
    labels: ["Branch 1", "Branch 2", "Branch 3"],
    datasets: [
      {label: "Revenue",data: [350000, 420000, 280000]},
      {label: "Expenses",data: [180000, 210000, 150000]}
      //additional dataset objects here
    ],
};
const verticalBarChartOptions = {
    barChartType: "vertical",   /* horizontal | vertical*/
    borderRadius: 1,
    barThickness: 10,
    yAxisLabel: "Price (k)"
  //additional options here
};
// additional code goes here
<ChartProvider
  chartType="bar"
  data={verticalBarChartData}          /* Chart Provider data object */
  options={verticalBarChartOptions}     /* Chart Provider options object */
/>
// additional code goes here
```

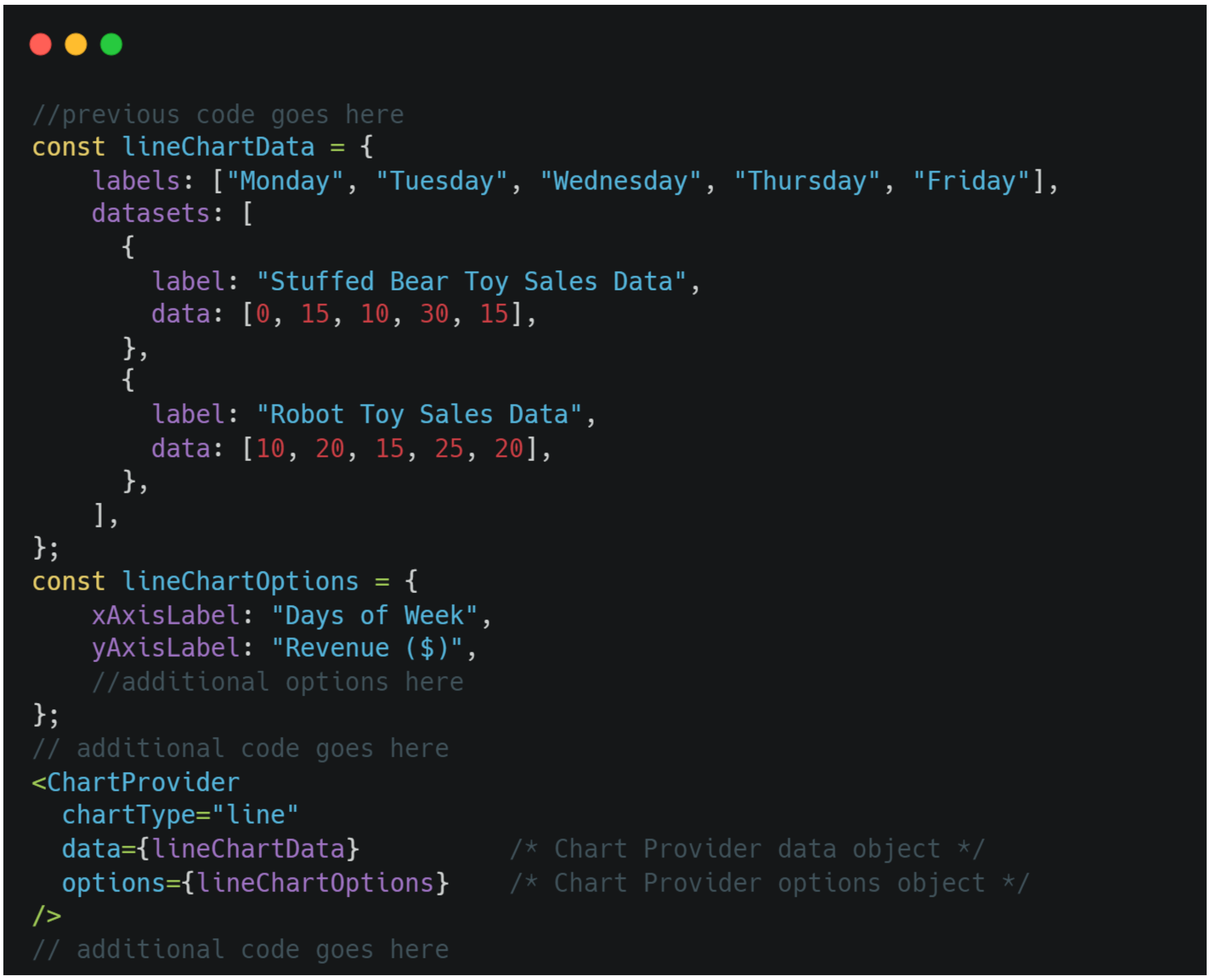
Stacked Vertical Bar chart:



```
// previous code goes here
const stackedVerticalBarData = {
    labels: ["January", "February", "March", "April", "May"],
    datasets: [
      {label: "Projections",data: [100000, 105000, 130000, 105000, 140000]},
      {label: "Actual",data: [95000, 100000, 105000, 110000, 115000]},
    ],
};

const stackedVerticalBarOptions = {
    barChartType: "vertical",
    barChartStacked: true       /* true | false, shows stacked bars */
    // additional options here
};

// additional code goes here
<ChartProvider
    chartType="bar"
    data={stackedVerticalBarData}
    options={stackedVerticalBarOptions}
/>
// additional code goes here
```

Line Chart:

Line Charts adhere to same structure and parameter requirements as Bar Charts, ensuring seamless integration and consistent usage across different chart types.

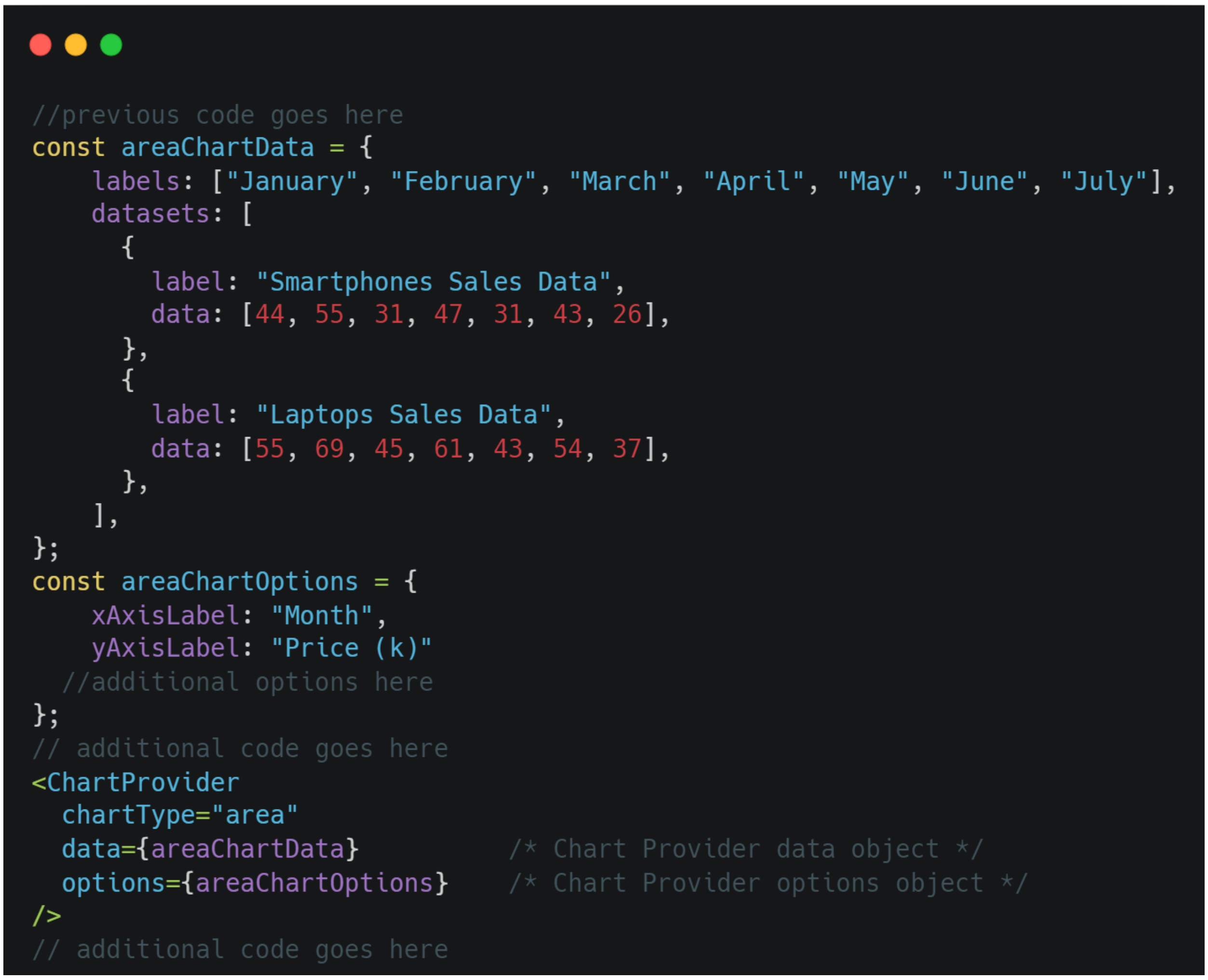


```
//previous code goes here
const lineChartData = {
    labels: ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"],
    datasets: [
      {
        label: "Stuffed Bear Toy Sales Data",
        data: [0, 15, 10, 30, 15],
      },
      {
        label: "Robot Toy Sales Data",
        data: [10, 20, 15, 25, 20],
      },
    ],
};
const lineChartOptions = {
    xAxisLabel: "Days of Week",
    yAxisLabel: "Revenue ($)",
    //additional options here
};
// additional code goes here
<ChartProvider
  chartType="line"
  data={lineChartData}          /* Chart Provider data object */
  options={lineChartOptions}    /* Chart Provider options object */
/>
// additional code goes here
```

Area Chart:

Area Charts follow the same guidelines as Line Charts and Bar Charts for accessing charts through the *Chart Provider*. By following the established guidelines, you can seamlessly integrate Area Charts into their visualizations alongside Line Charts and Bar Charts, facilitating a cohesive and standardized approach to data representation.

Example:

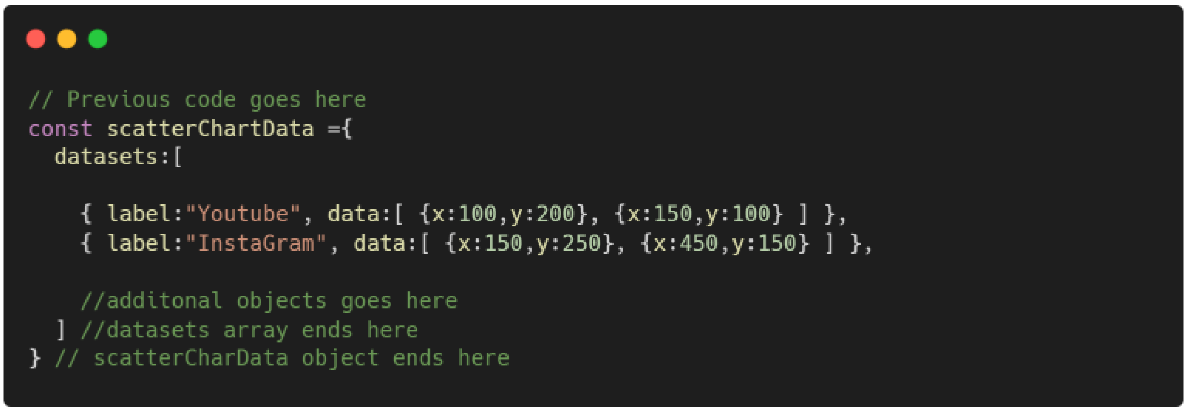


```
//previous code goes here
const areaChartData = {
    labels: ["January", "February", "March", "April", "May", "June", "July"],
    datasets: [
      {
        label: "Smartphones Sales Data",
        data: [44, 55, 31, 47, 31, 43, 26],
      },
      {
        label: "Laptops Sales Data",
        data: [55, 69, 45, 61, 43, 54, 37],
      },
    ],
};
const areaChartOptions = {
    xAxisLabel: "Month",
    yAxisLabel: "Price (k)"
  //additional options here
};
// additional code goes here
<ChartProvider
  chartType="area"
  data={areaChartData}           /* Chart Provider data object */
  options={areaChartOptions}     /* Chart Provider options object */
/>
// additional code goes here
```

Scatter Chart:

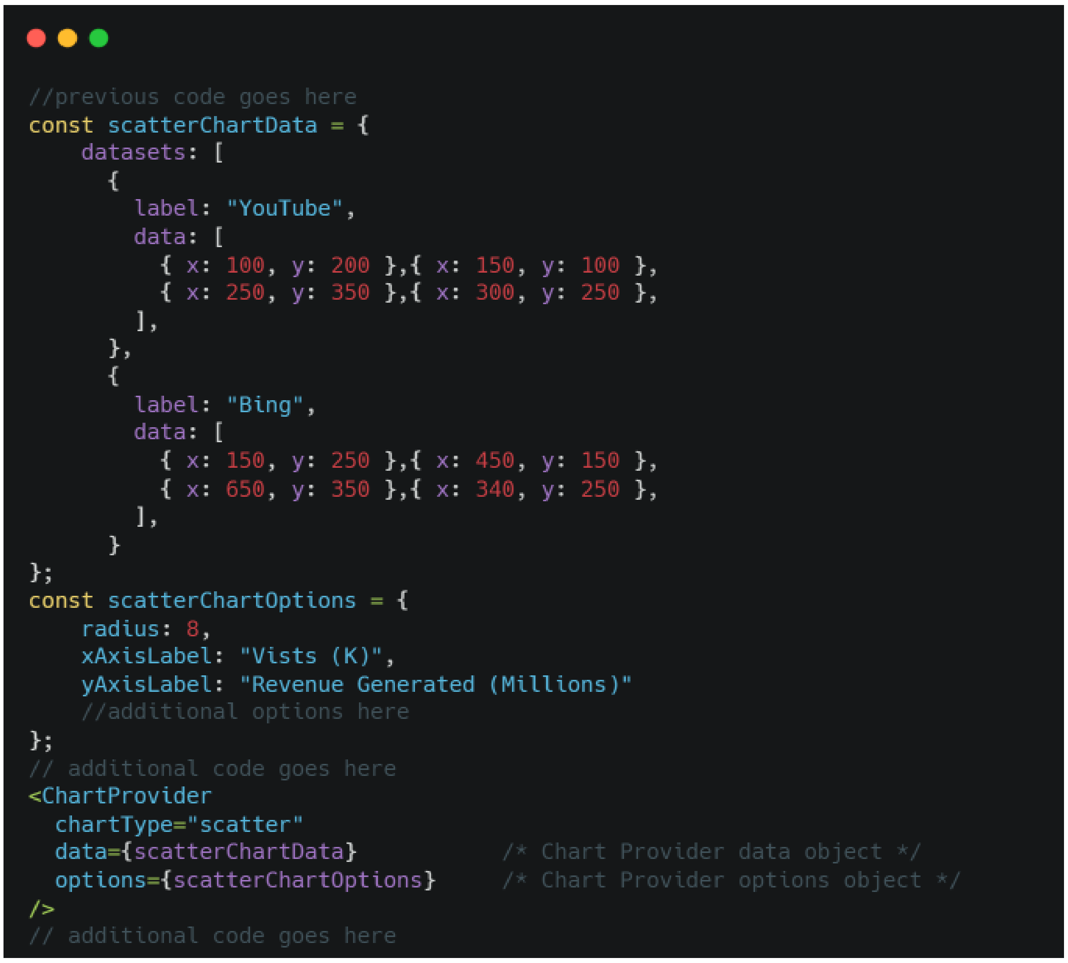
Scatter Charts data field in datasets array should have x and y coordinates to determine the value of “bubbles” along the x and y axis, while remaining other data values and setting will be same as area chart.

Sample Input:



```
// Previous code goes here
const scatterChartData ={
  datasets:[

    { label:"Youtube", data:[ {x:100,y:200}, {x:150,y:100} ] },
    { label:"InstaGram", data:[ {x:150,y:250}, {x:450,y:150} ] },

    //additonal objects goes here
  ] //datasets array ends here
} // scatterCharData object ends here
```

Sample Example:

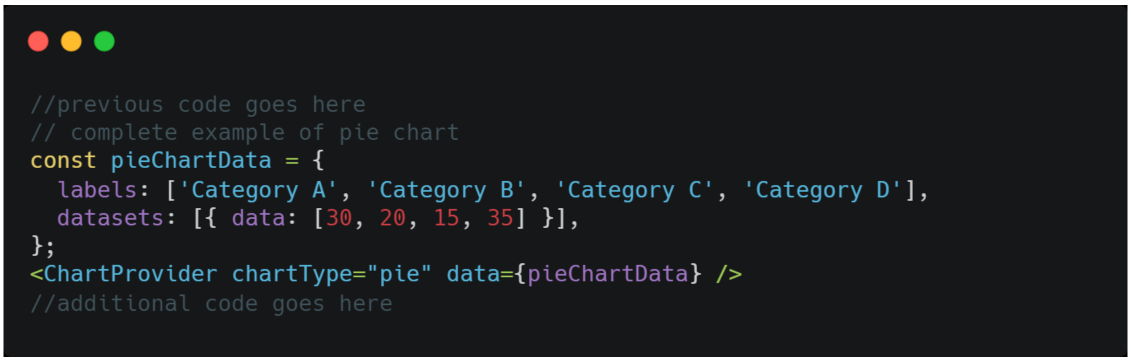


```
//previous code goes here
const scatterChartData = {
    datasets: [
      {
        label: "YouTube",
        data: [
          { x: 100, y: 200 },{ x: 150, y: 100 },
          { x: 250, y: 350 },{ x: 300, y: 250 },
        ],
      },
      {
        label: "Bing",
        data: [
          { x: 150, y: 250 },{ x: 450, y: 150 },
          { x: 650, y: 350 },{ x: 340, y: 250 },
        ],
      }
};
const scatterChartOptions = {
    radius: 8,
    xAxisLabel: "Vists (K)",
    yAxisLabel: "Revenue Generated (Millions)"
    //additional options here
};
// additional code goes here
<ChartProvider
  chartType="scatter"
  data={scatterChartData}          /* Chart Provider data object */
  options={scatterChartOptions}     /* Chart Provider options object */
/>
// additional code goes here
```

Pie Chart:

Pie chart uses the same dataset model as area chart, while **label** field within the 'datasets' array is optional.

Example:

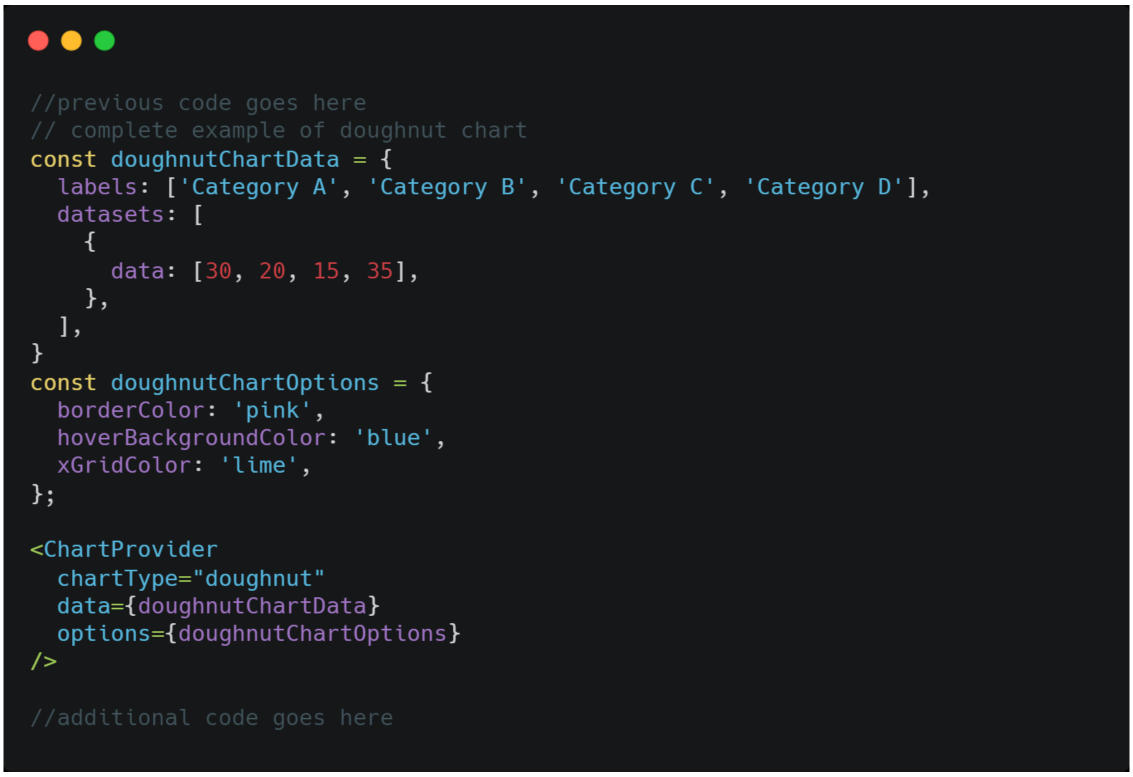


```
//previous code goes here
// complete example of pie chart
const pieChartData = {
  labels: ['Category A', 'Category B', 'Category C', 'Category D'],
  datasets: [{ data: [30, 20, 15, 35] }],
};
<ChartProvider chartType="pie" data={pieChartData} />
//additional code goes here
```

Doughnut Chart:

The structure of the data object for Doughnut Charts closely resembles that of Pie Charts and hence same dataset object model can be used for rendering it.

Example:



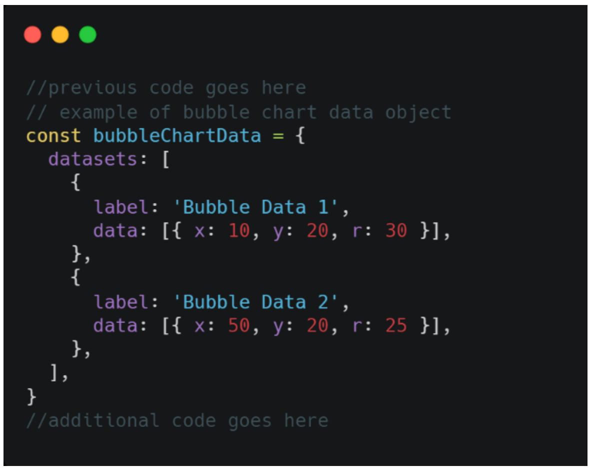
```
//previous code goes here
// complete example of doughnut chart
const doughnutChartData = {
  labels: ['Category A', 'Category B', 'Category C', 'Category D'],
  datasets: [
    {
      data: [30, 20, 15, 35],
    },
  ],
}
const doughnutChartOptions = {
  borderColor: 'pink',
  hoverBackgroundColor: 'blue',
  xGridColor: 'lime',
};

<ChartProvider
  chartType="doughnut"
  data={doughnutChartData}
  options={doughnutChartOptions}
/>

//additional code goes here
```

Bubble Chart:

Bubble Charts require additional parameters for each data point to draw the bubble on chart area. Following are the required object properties for a bubble chart:

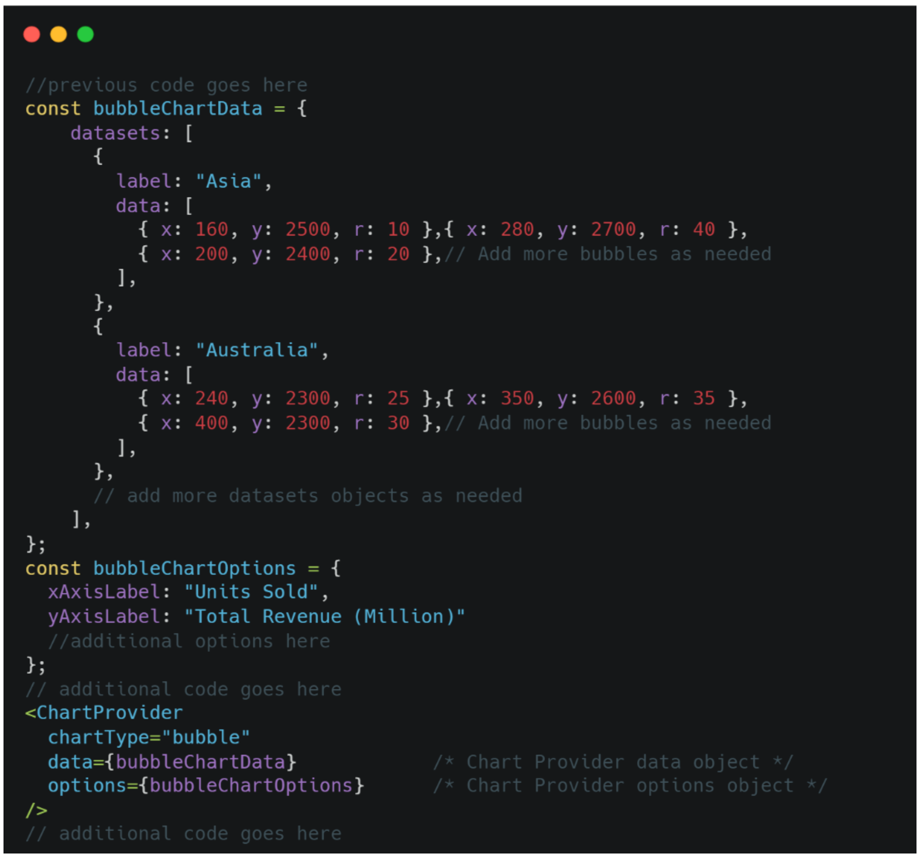
- For a bubble to be rendered on a specific data point, a **x** & **y** Coordinates are required which represents X-axis and Y-axis respectively.
- For size of bubble, the "radius" is necessary, which can be passed along as '**r**' within the data array objects.

Sample Input:



```
//previous code goes here
// example of bubble chart data object
const bubbleChartData = {
  datasets: [
    {
      label: 'Bubble Data 1',
      data: [{ x: 10, y: 20, r: 30 }],
    },
    {
      label: 'Bubble Data 2',
      data: [{ x: 50, y: 20, r: 25 }],
    },
  ],
}
//additional code goes here
```

Sample Example:

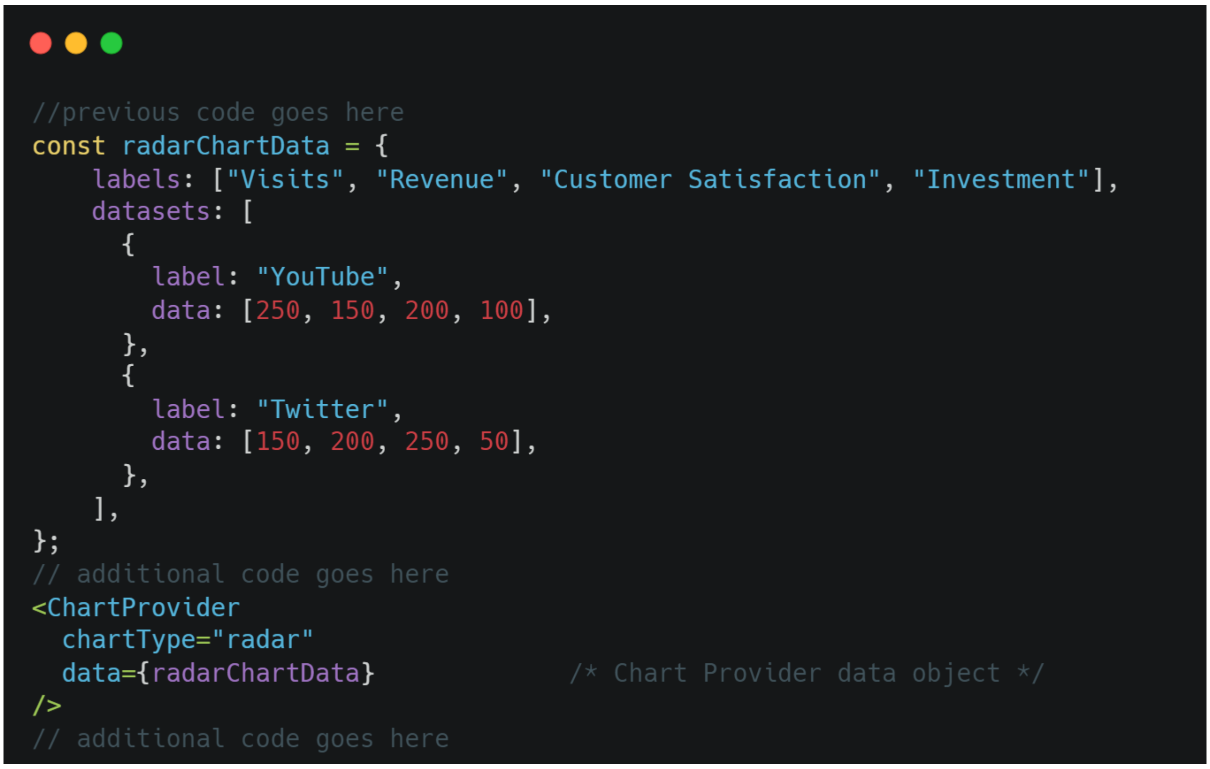


```
//previous code goes here
const bubbleChartData = {
    datasets: [
      {
        label: "Asia",
        data: [
          { x: 160, y: 2500, r: 10 },{ x: 280, y: 2700, r: 40 },
          { x: 200, y: 2400, r: 20 },// Add more bubbles as needed
        ],
      },
      {
        label: "Australia",
        data: [
          { x: 240, y: 2300, r: 25 },{ x: 350, y: 2600, r: 35 },
          { x: 400, y: 2300, r: 30 },// Add more bubbles as needed
        ],
      },
      // add more datasets objects as needed
    ],
};
const bubbleChartOptions = {
  xAxisLabel: "Units Sold",
  yAxisLabel: "Total Revenue (Million)"
  //additional options here
};
// additional code goes here
<ChartProvider
  chartType="bubble"
  data={bubbleChartData}              /* Chart Provider data object */
  options={bubbleChartOptions}        /* Chart Provider options object */
/>
// additional code goes here
```

Radar chart:

Radar chart can be utilized by passing 'radar' as the char type along with data.

Sample Example:



```
//previous code goes here
const radarChartData = {
    labels: ["Visits", "Revenue", "Customer Satisfaction", "Investment"],
    datasets: [
      {
        label: "YouTube",
        data: [250, 150, 200, 100],
      },
      {
        label: "Twitter",
        data: [150, 200, 250, 50],
      },
    ],
};
// additional code goes here
<ChartProvider
  chartType="radar"
  data={radarChartData}              /* Chart Provider data object */
/>
// additional code goes here
```