

Let's walk through an example of creating a custom ThingsBoard widget that uses a third-party JavaScript framework like **Chart.js** to display sensor data as a beautiful, interactive line chart.

Goal: Create a custom ThingsBoard widget that fetches time-series data (e.g., temperature) from a device and displays it using a Chart.js line chart.

Prerequisites:

- ThingsBoard instance (Community Edition or Professional Edition).
- A device configured in ThingsBoard with some time-series telemetry data (e.g., temperature).

Steps:

1. Create a New Widget

1. **Log in to ThingsBoard** as a Tenant Administrator.
2. Navigate to "**Widgets Library**" in the left-hand menu.
3. Choose an existing **Widget Bundle** (e.g., "Charts") or create a new one.
4. Click the "+" **icon** in the top right corner and select "**Create new widget**".
5. In the "Select widget type" dialog, choose "**Time series**" (since we'll be displaying historical data).
6. Give your widget a **Name** (e.g., "Chart.js Temperature Line Chart") and an optional description.

2. Add Third-Party Library (Chart.js)

Now, in the Widget Editor, you'll see several tabs: "Resources," "HTML," "CSS," "JavaScript," "Settings Schema," etc.

1. Go to the "**Resources**" tab.
2. In the "JavaScript" section, click the "+" **button** to add a new resource.
3. In the "URL" field, paste the CDN link for Chart.js. We'll use a widely available version:
<https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.min.js>
 - **Note:** Always use a specific version for stability. You can find the latest stable versions on CDNs like cdnjs.com.

3. Design the HTML Structure

1. Go to the "**HTML**" tab.
2. You'll typically need a canvas element where Chart.js will render the chart.

HTML

```
<div style="width: 100%; height: 100%;">
  <canvas id="myThingsBoardChart"></canvas>
</div>
```

- The style attribute helps the chart fill the widget's available space.
- The id="myThingsBoardChart" is crucial for referencing this canvas in your JavaScript.

4. Write the JavaScript Logic

Now for the core logic in the **"JavaScript"** tab. ThingsBoard provides a self object and a ctx (Widget Context) object that gives you access to data, APIs, and the widget's DOM.

JavaScript

```
self.onInit = function() {  
    // This function is called when the widget is initialized.  
    // It's the perfect place to set up your Chart.js instance.  
  
    var canvas = document.getElementById('myThingsBoardChart');  
    var chartContext = canvas.getContext('2d');  
    var myChart; // Declare chart variable to update later  
  
    // Function to update the chart with new data  
    self.onDataUpdated = function() {  
        // This function is called whenever the widget's data sources update.  
  
        if (!self.ctx.data || self.ctx.data.length === 0) {  
            // No data or empty data, clear chart if exists  
            if (myChart) {  
                myChart.destroy();  
                myChart = null;  
            }  
            return;  
        }  
  
        // ThingsBoard data structure:  
        // self.ctx.data is an array of data key objects.  
        // Each data key object has:  
        // - dataKey: object (label, color, type, etc.)  
        // - data: array of [timestamp, value] pairs  
  
        var labels = [];  
        var datasets = [];  
  
        // Assuming a single data source for simplicity (e.g., 'temperature')  
        var telemetryData = self.ctx.data[0];  
  
        // Extract timestamps (labels) and values for the dataset  
        telemetryData.data.forEach(function(item) {  
            labels.push(new Date(item[0]).toLocaleTimeString()); // Format timestamp for display
```

```
});
```

```
// Create a Chart.js dataset
```

```
datasets.push({
```

```
  label: telemetryData.dataKey.label, // Use ThingsBoard data key label
```

```
  data: telemetryData.data.map(function(item) { return item[1]; }), // Extract values
```

```
  borderColor: telemetryData.dataKey.color || 'rgba(75, 192, 192, 1)', // Use ThingsBoard color or default
```

```
  backgroundColor: 'rgba(75, 192, 192, 0.2)',
```

```
  fill: false,
```

```
  lineTension: 0.1
```

```
});
```

```
// If chart already exists, update its data
```

```
if (myChart) {
```

```
  myChart.data.labels = labels;
```

```
  myChart.data.datasets = datasets;
```

```
  myChart.update();
```

```
} else {
```

```
  // Otherwise, create a new Chart.js instance
```

```
  myChart = new Chart(chartContext, {
```

```
    type: 'line',
```

```
    data: {
```

```
      labels: labels,
```

```
      datasets: datasets
```

```
    },
```

```
    options: {
```

```
      responsive: true,
```

```
      maintainAspectRatio: false,
```

```
      scales: {
```

```
        xAxes: [{
```

```
          type: 'category', // For time series, but Chart.js handles this well
```

```
          labels: labels
```

```
        }],
```

```
        yAxes: [{
```

```
          ticks: {
```

```
            beginAtZero: true
```

```
          }
```

```
        }]
```

```
      },
```

```
      tooltips: {
```

```
        mode: 'index',
```

```
        intersect: false,
```

```
      },
```

```
      hover: {
```

```
        mode: 'nearest',
```

```

        intersect: true
    }
}
});
}
};

// Cleanup when widget is destroyed (important to prevent memory leaks)
self.onDestroy = function() {
    if (myChart) {
        myChart.destroy();
        myChart = null;
    }
};
};

```

5. Configure Data Sources

1. **Save** your custom widget definition.
2. Go to a **Dashboard** where you want to add this widget.
3. Enter **Edit mode**.
4. Click "+ **Add widget**".
5. Select your newly created widget from the bundle.
6. In the "Add Widget" dialog, go to the "**Data Source**" tab.
7. Click "**Add**" to add a data source.
8. **Create an Entity Alias** (e.g., "MyDevice") pointing to the device that sends temperature data.
9. Under "Data Keys," click "**Add**" and select "**Time-series**".
10. For the "Key" field, type temperature (or whatever your telemetry key is). You can also set a custom label and color here if you wish.
11. Click "**Add**" to add the data key.
12. Click "**Add**" on the "Add Widget" dialog.
13. **Save** the dashboard.

6. Adjust CSS (Optional)

You might want to add some CSS in the "**CSS**" tab of the widget editor to control the canvas size or other elements within your widget.

CSS

```

#myThingsBoardChart {
    /* Example: ensure canvas fills container */

```

```
width: 100% !important;
height: 100% !important;
}
```

Explanation of ThingsBoard Widget API used:

- **self.onInit():** Called once when the widget is initialized. Good for setting up non-data-dependent elements or external library instances.
- **self.onDataUpdated():** The most important callback for time-series and latest values widgets. This function is triggered every time new data arrives or the time window changes. You'll process `self.ctx.data` here.
- **self.onDestroy():** Called when the widget is removed from the dashboard or the dashboard is destroyed. Essential for cleaning up resources (like destroying Chart.js instances) to prevent memory leaks.
- **self.ctx.\$container:** A jQuery object representing the root HTML element of your widget. You can manipulate this to add/remove elements. (Note: The example above uses `document.getElementById` which is simpler for a single canvas, but `$container` is powerful for more complex DOM manipulation).
- **self.ctx.data:** An array containing the data fetched from your configured data sources. For time-series widgets, each item in `self.ctx.data` will have a data array of [timestamp, value] pairs.
- **self.ctx.data[i].dataKey.label:** Accesses the label defined for your data key in the widget configuration.
- **self.ctx.data[i].dataKey.color:** Accesses the color defined for your data key.

Result:

You should now see a line chart rendered by Chart.js on your ThingsBoard dashboard, displaying the temperature data from your device. When new temperature data arrives, the `onDataUpdated` function will automatically trigger, and your Chart.js instance will update to reflect the latest values.

This example demonstrates the core concept. You can extend this further by:

- Adding more data keys and datasets to your chart.
- Implementing different chart types (bar, pie, etc.) from Chart.js.
- Adding user interaction (e.g., custom buttons, filtering) by leveraging ThingsBoard's `self.ctx.controlApi` or `self.ctx.actionsApi`.
- Using settings schema to allow users to configure chart options (colors, labels, scales) directly from the widget settings.