JavaScript libraries - Chart.js

Data Visualization for Developers

Developed by:

- *Daniel Henrique Pereira Tolledo - 200257014*
- *Paulo Filipe Martins Fournier - 200257011*

Date: 27th Apr 2021

Setup

1. Download and install Node.js (follow the wizard with the default configuration)
2. Clone this repository
   git clone https://github.com/danielhpt/DV-Tutorial-Begin.git
3. Open the project in the IDE of your preference (we recommend WebStorm)
4. In the terminal (in the root of the project) execute: `npm install`

Content of the project

- A simple implementation of a webserver with Node and Express
- The "style.css" file (inside ./public/stylesheets/) contains the css of the page, you can change the 'width' and ' height' of the '.divTableCell' to resize the table cells
- The "index.html" file (inside ./views/) is where the tutorial will take place. Contains:
  - Chart.js the library that will be used in this tutorial

    ```
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    ```

  - JQuery
  - CSS
  - The space to put your code

    ```
    let onLoad = function () {
        /**
        * Insert your code here
        */
    };
    ```

  - The canvas tags that will contain the charts

    ```
    <canvas id="myChart_1"></canvas>
    ```

Running the project

In the terminal (in the root of the project) execute: `npm run dev`
It will run the project using *nodemon*, that automatically restart the node application when files change. This way, there is no need to restart the server all the time.

In the browser open localhost:3000

Tutorial

Exercises

Open the file ./view/index.html and write your code inside the onload function.

1. Adding the data:
    1. Add the data below, corresponding to the Top 10 councils with the largest population in Portugal, in 2018 (and their population in 2011), as well as their coordinates:

```
// Top 10 councils with the largest population in Portugal, in
2018
const cities = [
    'Lisboa',
    'Sintra',
    'Vila Nova de Gaia',
    'Porto',
    'Cascais',
    'Loures',
    'Braga',
    'Amadora',
    'Oeiras',
    'Matosinhos'
];
const pop2018 = [
    507220,
    388434,
    299938,
    215284,
    212474,
    211359,
    181919,
    181724,
    176218,
    174382
];

// Populations of those cities in 2011
const pop2011 = [
    547733,
    377835,
    302295,
    237591,
    206479,
    205054,
    181494,
    175136,
    172120,
    175478
];
```

```
// Longitude and latitude (x, y)
const citGeo = [
    {x: -9.13333, y: 38.71667},
    {x: -9.37826, y: 38.80097},
    {x: -8.61742, y: 41.13363},
    {x: -8.61099, y: 41.14961},
    {x: -9.42146, y: 38.69790},
    {x: -9.16845, y: 38.83091},
    {x: -8.42005, y: 41.55032},
    {x: -9.23083, y: 38.75382},
    {x: -9.31460, y: 38.69690},
    {x: -8.69630, y: 41.18440}
];
// Longitude, latitude and population in 2018 (x, y, r)
const citGeoPop = [  // population divided by 10000 for scale
    {x: -9.13333, y: 38.71667, r: 50.7220},
    {x: -9.37826, y: 38.80097, r: 38.8434},
    {x: -8.61742, y: 41.13363, r: 29.9938},
    {x: -8.61099, y: 41.14961, r: 21.5284},
    {x: -9.42146, y: 38.69790, r: 21.2474},
    {x: -9.16845, y: 38.83091, r: 21.1359},
    {x: -8.42005, y: 41.55032, r: 18.1919},
    {x: -9.23083, y: 38.75382, r: 18.1724},
    {x: -9.31460, y: 38.69690, r: 17.6218},
    {x: -8.69630, y: 41.18440, r: 17.4382}
];
```

2. Also add some colors, for later:

```
const colors = [
    'rgba(109, 198, 42, 0.7)',
    'rgba(23, 54, 120, 0.7)',
    'rgba(23, 120, 105, 0.7)',
    'rgba(23, 120, 54, 0.7)',
    'rgba(120, 120, 23, 0.7)',
    'rgba(120, 86, 23, 0.7)',
    'rgba(120, 23, 23, 0.7)',
    'rgba(120, 23, 58, 0.7)',
    'rgba(120, 23, 102, 0.7)',
    'rgba(102, 23, 120, 0.7)',
    'rgba(58, 23, 120, 0.7)',
    'rgba(29, 23, 120, 0.7)',
];
```

2. Creating your first chart:
    1. Let's begin with the basic configuration of every chart:

```
new Chart(document.getElementById(id) /* Canvas element */,{
    type: '', // bar, line, doughnut, pie, radar, scatter,
bubble, polar
    data: {
        labels: [],
        datasets: [{
            label: "",
            data: [],
        }]
    },
    options: {}
});
```

    2. For this chart let's use the first canvas `id = 'myChart_1'` and the *type* `'bar'`
    3. For the *data.labels* we will use the `cities` constant, for the *data.datasets.label* lets put `"Population"` and for the *data.datasets.data* we will use the `pop2018` constant
    4. Let's try to run it and open localhost:3000
    5. Better add some color. Inside *data.datasets* add:

```
backgroundColor: 'rgba(109, 198, 42, 0.9)'
```

    6. Refresh the page
    7. At this point you should have something like this:

```
new Chart(document.getElementById("myChart_1"), {
    type: 'bar',
    data: {
        labels: cities,
        datasets: [{
            label: "Population",
            data: pop2018,
            backgroundColor: 'rgba(109, 198, 42, 0.9)'
        }]
    },
    options: {}
});
```

3. Line chart:
    1. Let's begin by coping the code of the bar chart
    2. Change the canvas id to `id = 'myChart_2'` and the *type* to `'line'`
    3. Refresh the page.
    4. Let's make sure that it begins at 0. Inside *options* add:

```
scales: {
    y: {
        beginAtZero: true
    }
}
```

    5. Refresh the page

6. At this point you should have something like this:

```
new Chart(document.getElementById("myChart_2"), {
    type: 'line',
    data: {
        labels: cities,
        datasets: [{
            label: "Population",
            data: pop2018,
            backgroundColor: 'rgba(109, 198, 42, 0.9)'
        }]
    },
    options: {
        scales: {
            y: {
                beginAtZero: true
            }
        }
    }
});
```

4. Pie/Doughnut chart:
    1. Again, let's begin by coping the code of the bar chart
    2. Change the canvas id to `id = 'myChart_3'` and the *type* to `'pie'` or `'doughnut'`
    3. It's better to have different colors for each slice. Change the *data.datasets.backgroundColor* to the `colors` constant
    4. Refresh the page
    5. Try clicking in the legend and see what happens
    6. At this point you should have something like this:

```
new Chart(document.getElementById("myChart_3"), {
    type: 'pie', // doughnut, pie
    data: {
        labels: cities,
        datasets: [{
            label: "Population",
            data: pop2018,
            backgroundColor: colors
        }]
    },
    options: {}
});
```

5. Radar chart:

    1. Let's go back to the basic configuration and add a second dataset:

    ```
    new Chart(document.getElementById(id),{
        type: '',
        data: {
            labels: [],
            datasets: [{
                label: "",
                data: [],
            }, {
                label: "",
                data: [],
            }]
        },
        options: {}
    });
    ```

    2. For this chart let's use `id = 'myChart_4'`, the *type* `'radar'` and for the *data.labels* `cities`
    3. For the first dataset:
        1. *data.datasets.label* = `"Population in 2018"`
        2. *data.datasets.data* = `pop2018`
    4. For the second dataset:
        1. *data.datasets.label* = `"Population in 2011"`
        2. *data.datasets.data* = `pop2011`
    5. Let's add some color to differentiate them

        1. First dataset (red)

        ```
        fill: true,
        backgroundColor: 'rgba(255, 99, 132, 0.2)',
        borderColor: 'rgb(255, 99, 132)'
        ```

        2. Second dataset (blue)

        ```
        fill: true,
        backgroundColor: 'rgba(54, 162, 235, 0.2)',
        borderColor: 'rgb(54, 162, 235)'
        ```

    6. And make sure that it begins at 0. Inside *options* add:

    ```
    scales: {
        r: {
            beginAtZero: true
        }
    }
    ```

    7. Refresh the page.

8. At this point you should have something like this:

```
new Chart(document.getElementById("myChart_4"), {
    type: 'radar',
    data: {
        labels: cities,
        datasets: [{
            label: "Population in 2018",
            data: pop2018,
            fill: true,
            backgroundColor: 'rgba(255, 99, 132, 0.2)',
            borderColor: 'rgb(255, 99, 132)',
        }, {
            label: "Population in 2011",
            data: pop2011,
            fill: true,
            backgroundColor: 'rgba(54, 162, 235, 0.2)',
            borderColor: 'rgb(54, 162, 235)',
        }]
    },
    options: {
        scales: {
            r: {
                beginAtZero: true
            }
        }
    }
});
```

6. Scatter chart
    1. Let's begin by coping the code of the pie chart
    2. Change the canvas id to `id = 'myChart_5'`, the *type* to `'scatter'` and the *data.datasets.data* to the `citGeo` constant
    3. Let's make sure that the x-axis appears. Inside *options* add:

```
scales: {
    x: {
        type: 'linear',
        position: 'bottom'
    }
}
```

    4. Refresh the page.

5. At this point you should have something like this:

```
new Chart(document.getElementById("myChart_5"), {
    type: 'scatter',
    data: {
        labels: cities,
        datasets: [{
            label: "Population",
            data: citGeo,
            backgroundColor: colors
        }]
    },
    options: {
        scales: {
            x: {
                type: 'linear',
                position: 'bottom'
            }
        }
    }
});
```

7. Bubble chart
    1. Let's begin by coping the code of the scatter chart
    2. Change the canvas id to `id = 'myChart_6'`, the *type* to `'bubble'` and the *data.datasets.data* to the `citGeoPop` constant
    3. Refresh the page
    4. At this point you should have something like this:

```
new Chart(document.getElementById("myChart_6"), {
    type: 'bubble',
    data: {
        labels: cities,
        datasets: [{
            label: "Population",
            data: citGeoPop,
            backgroundColor: colors
        }]
    },
    options: {
        scales: {
            x: {
                type: 'linear',
                position: 'bottom'
            }
        }
    }
});
```

Tutorial - Advanced

Setup

1. Make sure to save/commit all changes.
2. Switch to advanced branch.
   git checkout -B advanced origin/advanced

Content of the project

- The "index.html" file (inside ./views/) has now changed.
  Contains:
  - chart.js with the complied code used in part I of this seminar.

    ```
    <script src="javascripts/charts.js"></script>
    ```

    This will create an object called app with the method **getChart(pos)** that will return the chart object based on the grid position.
    Example: getChart(2) will return the pie chart Object.
  - The space to put your code will now be

    ```
    let handsOn = function () {
        /**
     * Insert your code here
     */
    };
    ```

Running the project

In the terminal (in the root of the project) execute: `npm run dev`
It will run the project using *nodemon*, that automatically restart the node application when files change. This way, there is no need to restart the server all the time.

In the browser open localhost:3000

Exercises

Open the file ./view/index.html and write your code inside the `handsOn` function.

1. Add the data dynamically to the bar chart.

   Use function ***addDataToBarChart*** to add data dynamically to the bar chart. Each record should have a label and, at least, one value so this function will accept a value and a label as parameters.

```
/**
 * Add data dynamically
 * @param (string) label The dataset label
 * @param (number) The dataset value
 */
addDataToBarChart: (label, data) => {}
```

   i. All labels can be found on *data.labels* Array, so to add the label just run

```
app.getChart(0).data.labels.push(label);
```

   ii. The data is saved in each datasets. In this case we only have one, so we will add the data to the first dataset (position 0)

```
app.getChart(0).data.datasets[0].data.push(data);
```

   iii. In order to view our changes we need to trigger the update event on the chart

```
app.getChart(0).update();
```

2. Change labels dynamically on the bar chart.

   Use function ***changeDataSetLabel*** to change the label dynamically on the bar chart.

```
/**
 * Will update the first dataset label
 * @param (string) label The dataset label
 */
changeDataSetLabel: (label) => {}
```

   i. All configurations can be changed directly on the chart object.
   After that we just need to trigger.

```
app.getChart(0).data.datasets[0].label = label;
app.getChart(0).update();
```

3. Add the data dynamically to the bar chart, with remove request

   Use function ***getDataFromServerSideForBarChart*** to request and add data dynamically to the bar chart, making an AJAX request to the backend server. In this exercise we will use jquery to perform the remote request.

```
/**
 * Get random data and label from server
 */
getDataFromServerSideForBarChart: () => {}
```

   On the server backend there is a GET route called ***/data/getrandom*** that will return a JSON object with random data and label to be used here. Example:

```
{"label": "dzy1e","value": 176175}
```

   i. So, first we need to make the AJAX request to that route.
   This simple command with perform the GET request and will make the response available on the data object. Once that this head already tells that the return will be a json object, the data will be already an object, in this case, two attributes: label with a string random label and value with a random Number between 0 and 500.000.

```
$.get( "data/getrandom", (data) => {});
```

   ii. No, we just need to call the function that we already implement on exercise one.

```
$.get( "data/getrandom", (data) => {
    handsOn.addDataToBarChart(data.label, data.value);
});
```

4. Add a *delay* animation to the bar chart.

   On chartjs we have different animations available to be used. In this example we will add an animation called delay that will have impact when adding a new value to the dataset.

   Use function ***addDelayAnimationToBarChart*** set the new animation dynamically.

```
/**
 * Add slide delay animation
 */
addDelayAnimationToBarChart: () => {}
```

   i. All animations are set as an option of the chart in *option.animation* object.
   The specification can be found here:
   https://www.chartjs.org/docs/latest/api/#animationspec

Name	Type	Description
`delay` ?	*Scriptable*<number, *ScriptableContext* <TType>>	Delay before starting the animations. `default` 0

So, we will increase the delay time for each animation that will happen.
The delay function has an argument called context that regards the dataset where the animation is taking place.

In this example, we are saying to the chart to delay the next animation 200ms times the data index.

```
app.getChart(0).options.animation = {
    delay: (context) => {
        return context.dataIndex * 200;
    }
};
```

If we want to see the animation for the current data, we just need to trigger the update function.

5. Add a progressive line animation to the line chart

Use function ***addProgressiveLineToLineChart*** to add this animation.

```
/**
 * Add progressive line animation
 */
addProgressiveLineToLineChart: () => {}
```

i. In this case we just want to change the behavior on the x-axis.
The animation object allows us the specify the axis that we intend to work on.

So, we will say that we want the animation to be set on the "x" axis just setting the animation configuration directly on the x attribute.

The easing functions specify the rate of change of a parameter over time. In this case this option allows a huge number of different possibilities

`easing` ?	*Scriptable*<*EasingFunction*, *ScriptableContext*<TType>>	Easing function to use `default` 'easeOutQuart'

T **EasingFunction**: *linear* | *easeInQuad* | *easeOutQuad* | *easeInOutQuad* | *easeInCubic* | *easeOutCubic* | *easeInOutCubic* | *easeInQuart* | *easeOutQuart* | *easeInOutQuart* | *easeInQuint* | *easeOutQuint* | *easeInOutQuint* | *easeInSine* | *easeOutSine* | *easeInOutSine* | *easeInExpo* | *easeOutExpo* | *easeInOutExpo* | *easeInCirc* | *easeOutCirc* | *easeInOutCirc* | *easeInElastic* | *easeOutElastic* | *easeInOutElastic* | *easeInBack* | *easeOutBack* | *easeInOutBack* | *easeInBounce* | *easeOutBounce* | *easeInOutBounce*

These visual examples can make us understand what will be the output.

https://easings.net/

So, we will set the *easing* as "linear", the *duration* of each animation for 500ms, we will set the start of the animation in the beginning "*from:NaN*" and the *delay* behavior will be same of the previous exercise.

```
app.getChart(1).options.animation = {
    x: {
        type: 'number',
        easing: 'linear',
        duration: 500,
        from: NaN, // the point is initially skipped
        delay(context) {
            return context.dataIndex * 200;
        }
    }
};
```

Again, don't forget to trigger the update to see it happen.

6. Add on click listener event

In this exercise we will learn to get the clicked value in a chart, on this case we will work directly on the pie chart.

Use function ***getValueFromPieChartOnClick*** to add this listener.

```
/**
 * Add onclick listener
 */
getValueFromPieChartOnClick: () => {}
```

Chartjs has two default listener that can be set directly in the options.
Specification can be found here:
https://www.chartjs.org/docs/2.6.0/general/interactions/events.html

For this exercise we will use the onClick property:

`onClick`	`Function`	`null`	Called if the event is of type 'mouseup' or 'click'. Called in the context of the chart and passed the event and an array of active elements

i. We will alter the user with the dataset, data index, and data value when clicked. This event will be triggered every time that the user clicks the chart, even outside if the dataset.

The *onClick* function when triggered will give us two parameters, one is the default click event of the browser with the chart object inside. The other one will be an array with the list of datasets clicked.

Why a list? If we have two datasets that are overlapped, if the user clicks it we can get the value of the dataset in front as well of the clicked value of the dataset behind it.

```
app.getChart(2).options.onClick = (event, arrayOfActiveElements) => {

    if(!arrayOfActiveElements.length)
        return;

    alert(
        'Dataset: ' + arrayOfActiveElements[0].datasetIndex +
        '\nIndex: ' + arrayOfActiveElements[0].index +
        '\nValor: ' +
event.chart.data.datasets[arrayOfActiveElements[0].datasetIndex].data
[arrayOfActiveElements[0].index]
    );
};
```

7. Dynamically smooth the radar chart

   We can set the line tension between each point of the chart, and we can set it dynamically.

   Use function ***smoothRadarChart*** to add this animation.

```
/**
 * Will dynamically smooth the radar chart
 */
smoothRadarChart: () => {}
```

   One of the default options are the element options that can be found at options/elements. The elements object has all components of the chart, like the Arc, bar, Line, etc. Specification is here: https://www.chartjs.org/docs/latest/api/interfaces/elementoptionsbytype.html

   i. We want to change the line tension, that can be found here. https://www.chartjs.org/docs/latest/api/interfaces/lineoptions.html

```
app.getChart(3).options.elements.line.tension = 0.4;
```

      Again, don't forget to trigger the update to see it happen.

8. Add a second dataset as a different chart type

   In this example we will add a line dataset to the bar chart.
   This Chartjs allows us to add different datasets and combine them like bar with radar, etc.

   Let's use function ***addLinetoBarChart*** to add this animation.

```
/**
 * Add second chart on top of the first one
 */
addLinetoBarChart: () => {}
```

   i. To add a new dataset we just need to push the new one to the list of datasets in the chart. Here we can also set the dataset type as being line.

```
app.getChart(0).data.datasets.push({
    label: 'Some other dataset',
    data: app.pop2018,
    backgroundColor: app.colors,
    type: 'line',
    order: 0
});
```

9. Change the background color of each data

   As our final exercise we will dynamically set a new color pallet to the chart.
   Use function ***setBarColors*** to change them.

```
/**
 * Update bar color
 */
setBarColors: () => {}
```

   i. We will use the list of colors already set on the app object and set them on the backgroundColor property

```
app.getChart(0).data.datasets[0].backgroundColor = app.colors;
```