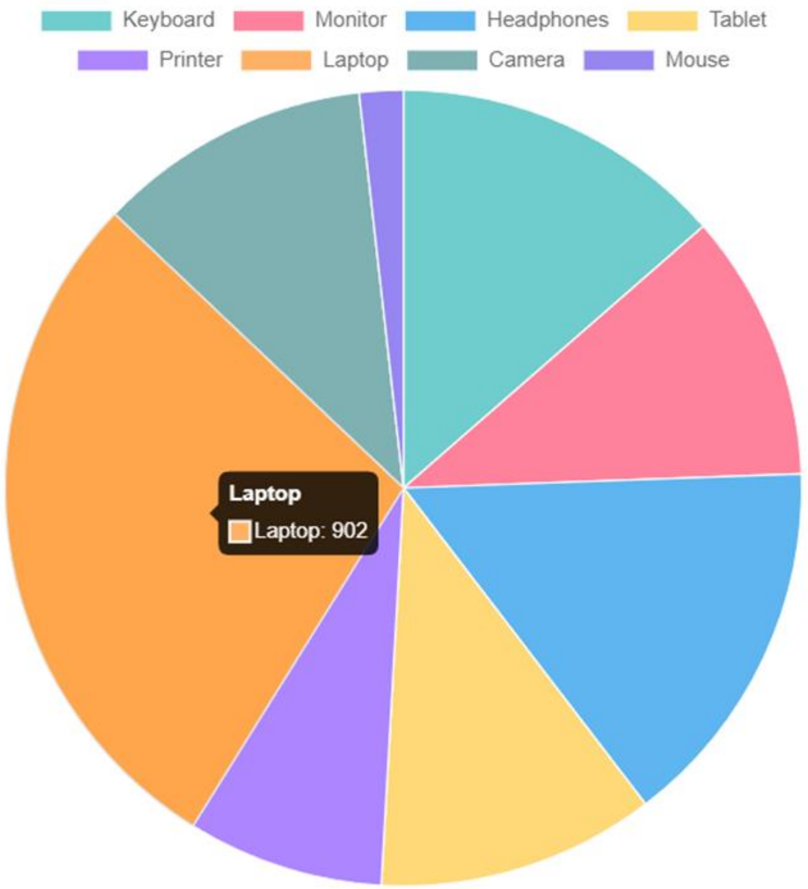
Introduction to Chart Generation using JavaScript (Chart.js)

This week, You will learn how to build different types of charts using **HTML** and **JavaScript**.

In your coursework, you will work with **data stored in MongoDB**. But why do we need charts?

- Data is only valuable if it can be interpreted.
- Visuals help identify patterns, trends, and insight
- Examples of visualisation types:
  - Bar → Compare quantities
  - Pie → Show proportions
  - Line → Show trends over time
  - Scatter → Show correlations



Introduction to Chart Generation using JavaScript (Chart.js)

Understanding the basic concepts of a chart:

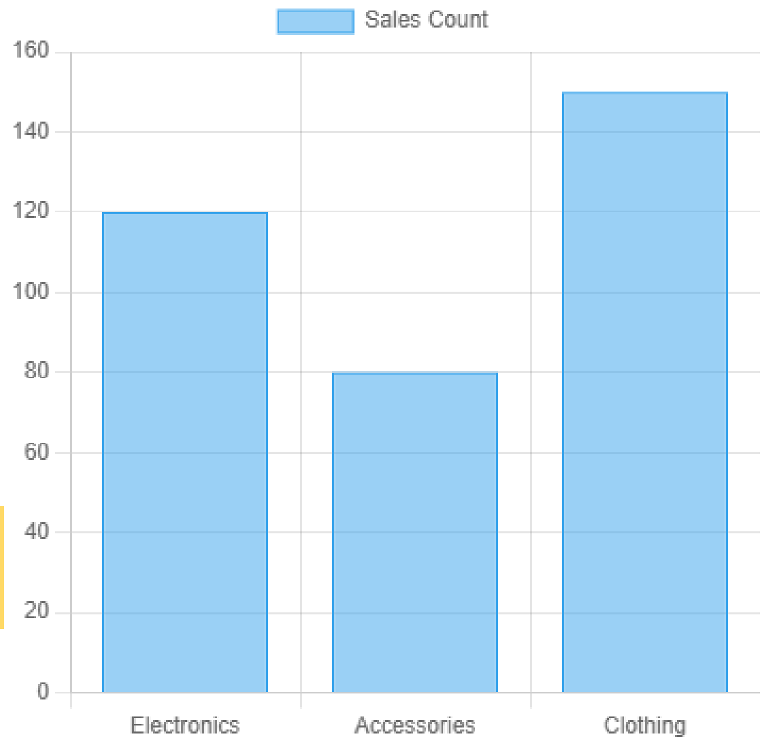
- Every chart needs at least two things:

    1. Labels (X-axis / categories)
    2. Data values (Y-axis / numbers)

- Example:

```
Labels: ["Electronics", "Accessories", "Clothing"]
Data: [120, 80, 150]
```

- These are mapped to visuals by a chart library.



Introduction to Chart Generation using JavaScript (Chart.js)

You will use **Chart.js** library to generate different types of charts:

- **Chart.js**:

  1. A lightweight and powerful JavaScript charting library.
  2. Supports bar, pie, line, radar, doughnut, polar, bubble, etc.
  3. Easy to use with simple HTML + JS.

- Minimal Example:

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

Telling the script where to find the **`chart.js`** library

Creating a **canvas** to draw a chart

Introduction to Chart Generation using JavaScript (Chart.js)

Steps of creating a basic chart:

- Step 1: Create a **`<canvas>`** element in HTML.
- Step 2: Reference it in JS with **`getElementById`**.
- Step 3: Define chart type, labels, and datasets.
- Step 4: Initialise with **`new Chart(ctx, {...})`**.

Example code:

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

<script>
const ctx = document.getElementById('myChart');
new Chart(ctx, {
  type: 'bar',
  data: { labels: ['A', 'B', 'C'], datasets: [{ data: [10, 20, 30] }] }
});
</script>
```

Introduction to Chart Generation using JavaScript (Chart.js)

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

<script>
const ctx = document.getElementById('myChart');
new Chart(ctx, {
  type: 'bar',
  data: { labels: ['A', 'B', 'C'], datasets: [{ data: [10, 20, 30] }] }
});
</script>
```

Key components of a chart

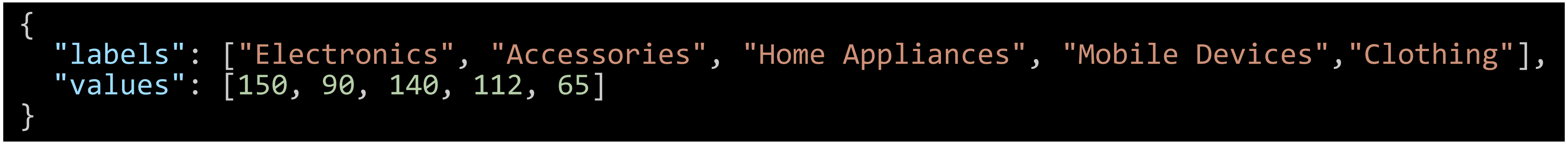
- **type:** defines chart type (bar, pie, doughnut, line).
- **labels**: defines x-axis categories.
- **datasets**: array of objects with:
  - **label**: legend name
  - **data**: values
  - backgroundColor, borderColor
- **options**: controls axes, titles, legends, scales.

Introduction to Chart Generation using JavaScript (Chart.js)

- Download `BasicChart.html` from Canvas and open it in a browser.
  - It should display a **bar chart**.
- **Study the code** to identify the chart components (canvas, labels, datasets, options) and how they map to the rendered chart.

- **Experiment:** change **`type`** to **`pie`**, then **`doughnut,`** respectively, and observe the change in the output chart.
- **Extend:** add extra **`labels`** and corresponding **`data`** points to see how the chart adapts — use this to cement your understanding of labels vs. datasets.

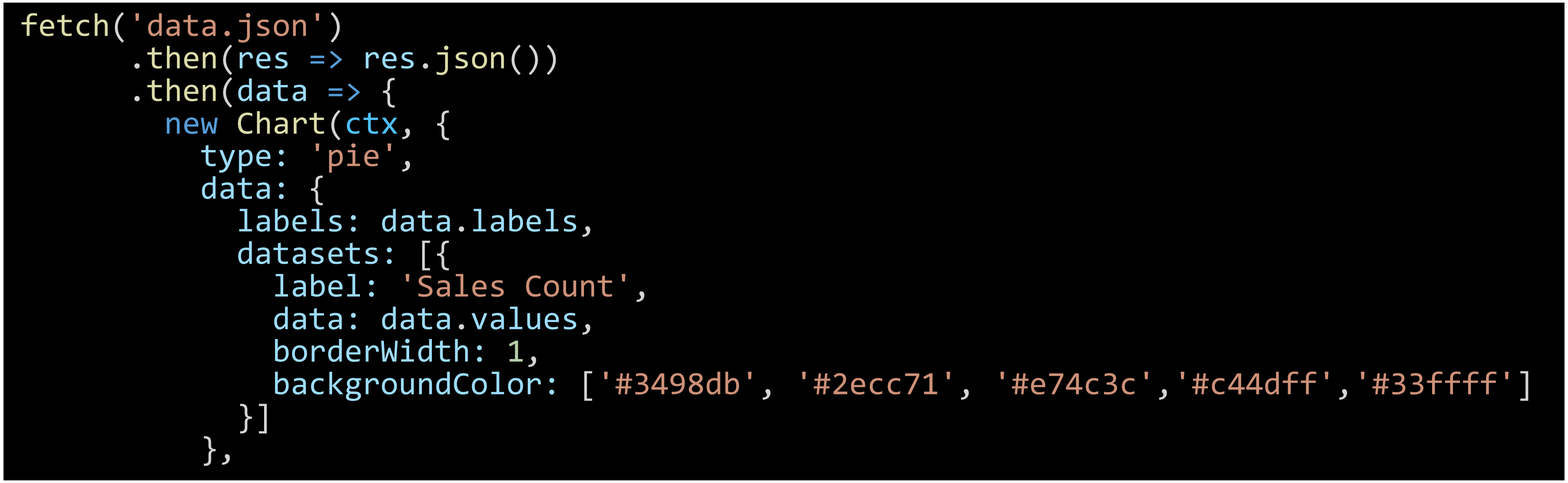
- Until now, the charts were created using **hard-coded data**.
- In practice, charts often use data fetched dynamically from a **JSON** file or an **API**.
- This allows the chart to **update automatically** when the underlying data changes.
- Suppose you have a file named '**`data.json`**' containing the following data: (*If it doesn't exist, create this file with the data shown.*)



```
{
  "labels": ["Electronics", "Accessories", "Home Appliances", "Mobile Devices","Clothing"],
  "values": [150, 90, 140, 112, 65]
}
```

Introduction to Chart Generation using JavaScript (Chart.js)

- Use the code below to **read data from** ‘**`data.json`**’ and render the chart dynamically.
- **Study the code:** compare it with the corresponding section in **`BasicChart.html`** to spot similarities and differences (how data is supplied, parsed, and passed to Chart.js).
- Action: replace the code in lines 12–21 of **`BasicChart.html`** with the code snippet shown below.



```
fetch('data.json')
      .then(res => res.json())
      .then(data => {
        new Chart(ctx, {
          type: 'pie',
          data: {
            labels: data.labels,
            datasets: [{
              label: 'Sales Count',
              data: data.values,
              borderWidth: 1,
              backgroundColor: ['#3498db', '#2ecc71', '#e74c3c','#c44dff','#33ffff']
            }]
          },
```

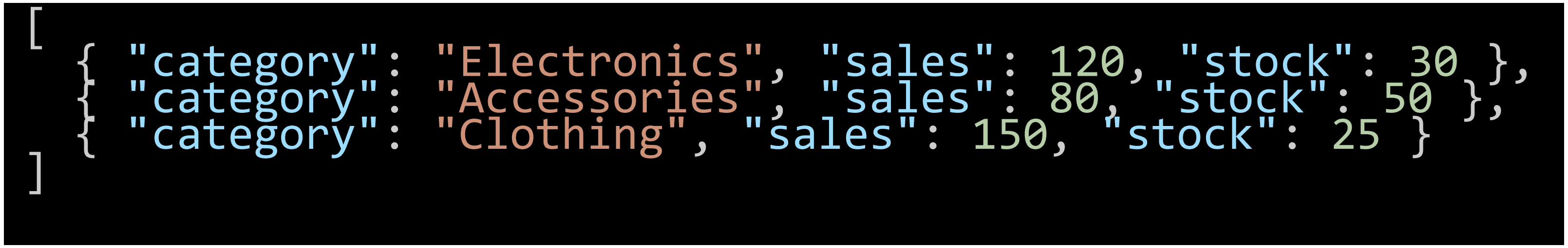
ℹ **`data.json`** must be in the same directory as your HTML file.

⚠ If you’re opening the HTML file directly in a browser (file:///) by double clicking, the browser will block the **`fetch()`** request for local files.
Solution: Run the file from a local server instead of opening it directly. You can:

- Place the files in the **`htdocs`** folder of **XAMPP** and access via `http://localhost/`, or
- Run it from a **Node.js server** (refer to last week’s lab for setup steps).

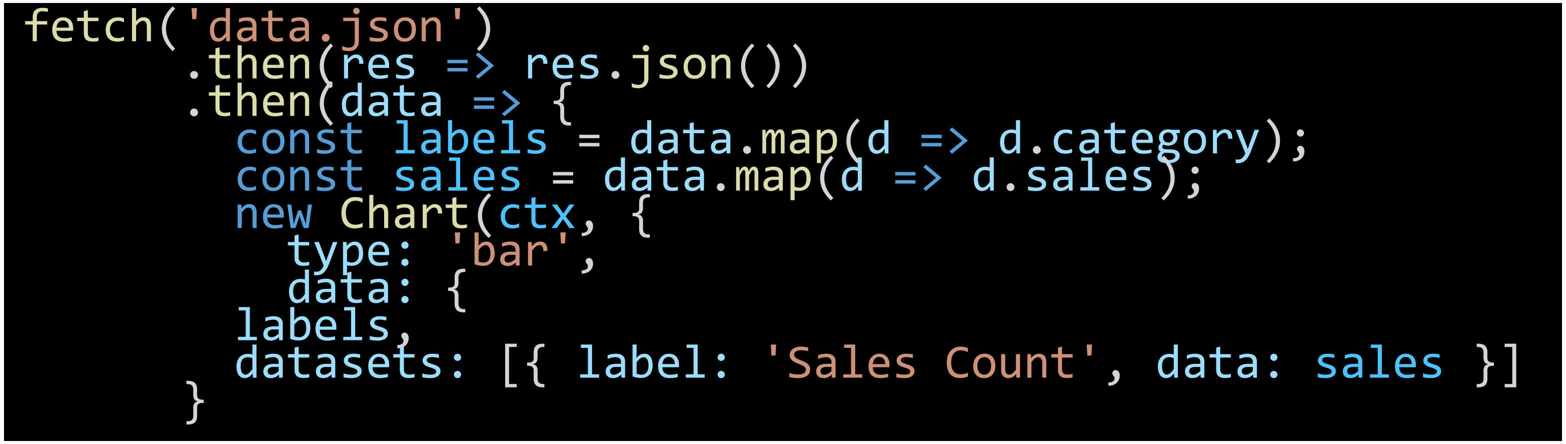
Introduction to Chart Generation using JavaScript (Chart.js)

- If the data in the ‘**data.json**’ file is updated to the following structure, it will resemble a MongoDB-style collection, where each document represents a record within a dataset.



```
[
  { "category": "Electronics", "sales": 120, "stock": 30 },
  { "category": "Accessories", "sales": 80, "stock": 50 },
  { "category": "Clothing", "sales": 150, "stock": 25 }
]
```

Then you can extract arrays dynamically:



```
fetch('data.json')
      .then(res => res.json())
      .then(data => {
        const labels = data.map(d => d.category);
        const sales = data.map(d => d.sales);
        new Chart(ctx, {
          type: 'bar',
          data: {
        labels,
        datasets: [{ label: 'Sales Count', data: sales }]
      }
```

Now that you've learned how to create various charts using **static data** and data from a **JSON file**, it's time to extend that knowledge to generate **dynamic charts** by fetching live data from **MongoDB**.

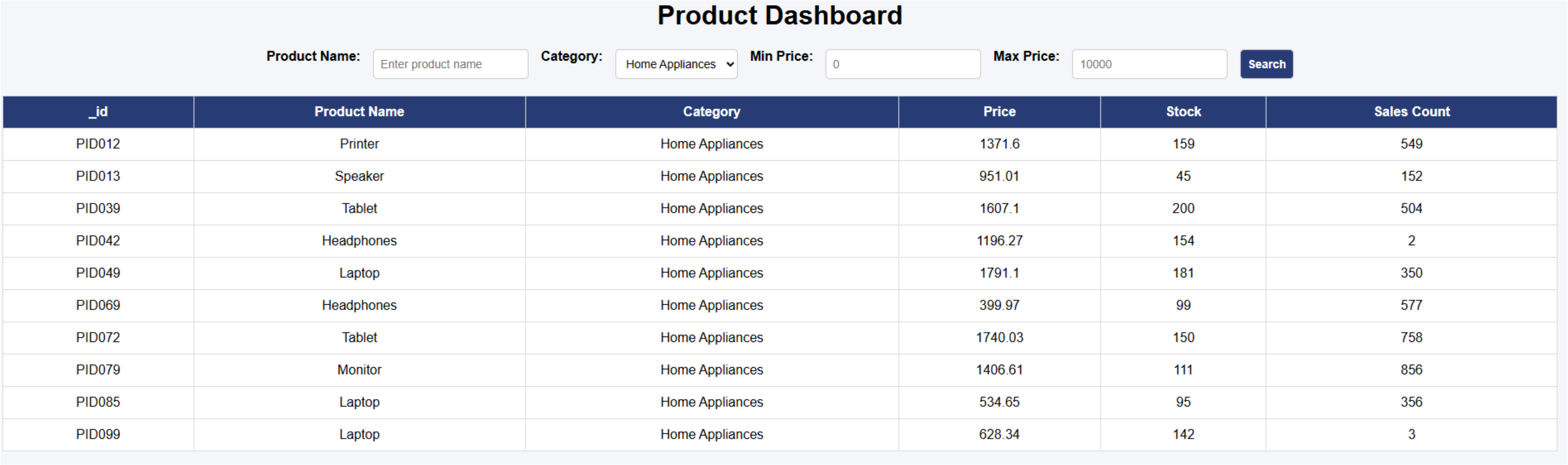
Continue from **Week 7's lab**, where you:

- Imported **100 documents** into the **`products`** collection within the **Shop** database.

To get started:

1. Create a **copy** of your **Week 7 Node.js project**.
2. Rename the new project folder to **LAB 8.**

The dashboard you built in Week 7 looked like the one shown below. In this lab, we'll simplify and enhance it — follow the steps on the next pages carefully.

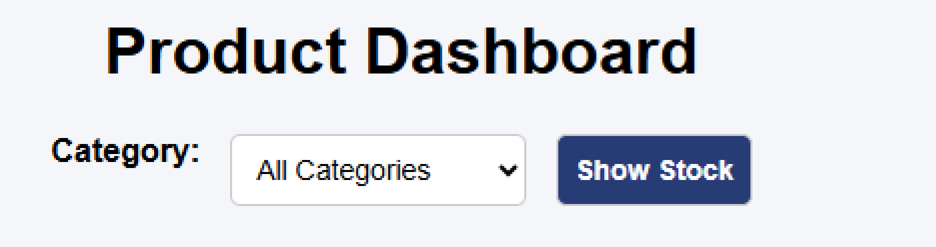


Product Dashboard

Product Name: Enter product name Category: Home Appliances Min Price: 0 Max Price: 10000

_id	Product Name	Category	Price	Stock	Sales Count
PID012	Printer	Home Appliances	1371.6	159	549
PID013	Speaker	Home Appliances	951.01	45	152
PID039	Tablet	Home Appliances	1607.1	200	504
PID042	Headphones	Home Appliances	1196.27	154	2
PID049	Laptop	Home Appliances	1791.1	181	350
PID069	Headphones	Home Appliances	399.97	99	577
PID072	Tablet	Home Appliances	1740.03	150	758
PID079	Monitor	Home Appliances	1406.61	111	856
PID085	Laptop	Home Appliances	534.65	95	356
PID099	Laptop	Home Appliances	628.34	142	3

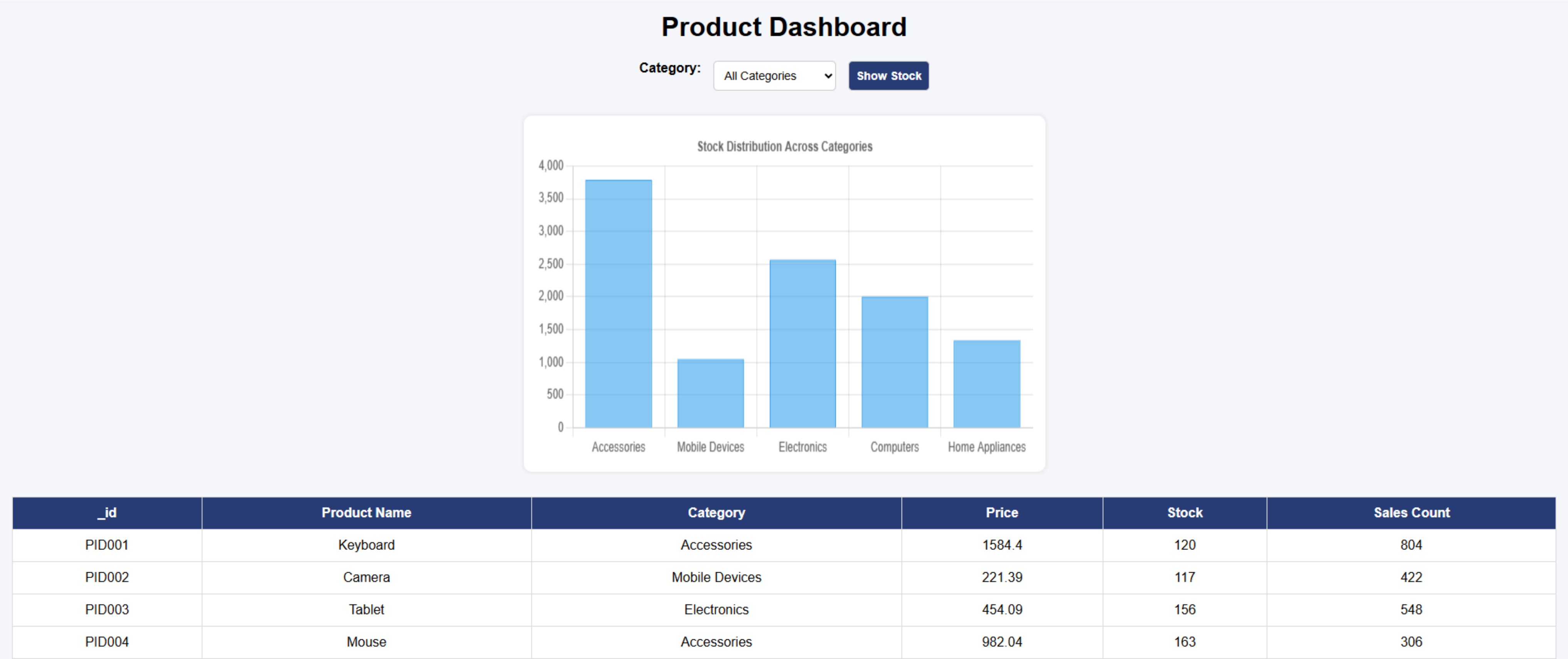
Modify the input panel so that it contains **only one dropdown menu** for selecting a **product category** and include a single **“Show Stock”** button to display the results.



Product Dashboard

When the page loads, the output should look like the following.

- A bar chart shows total stock per category.
- A table lists all products (which is same as the week 7’s lab’s output)

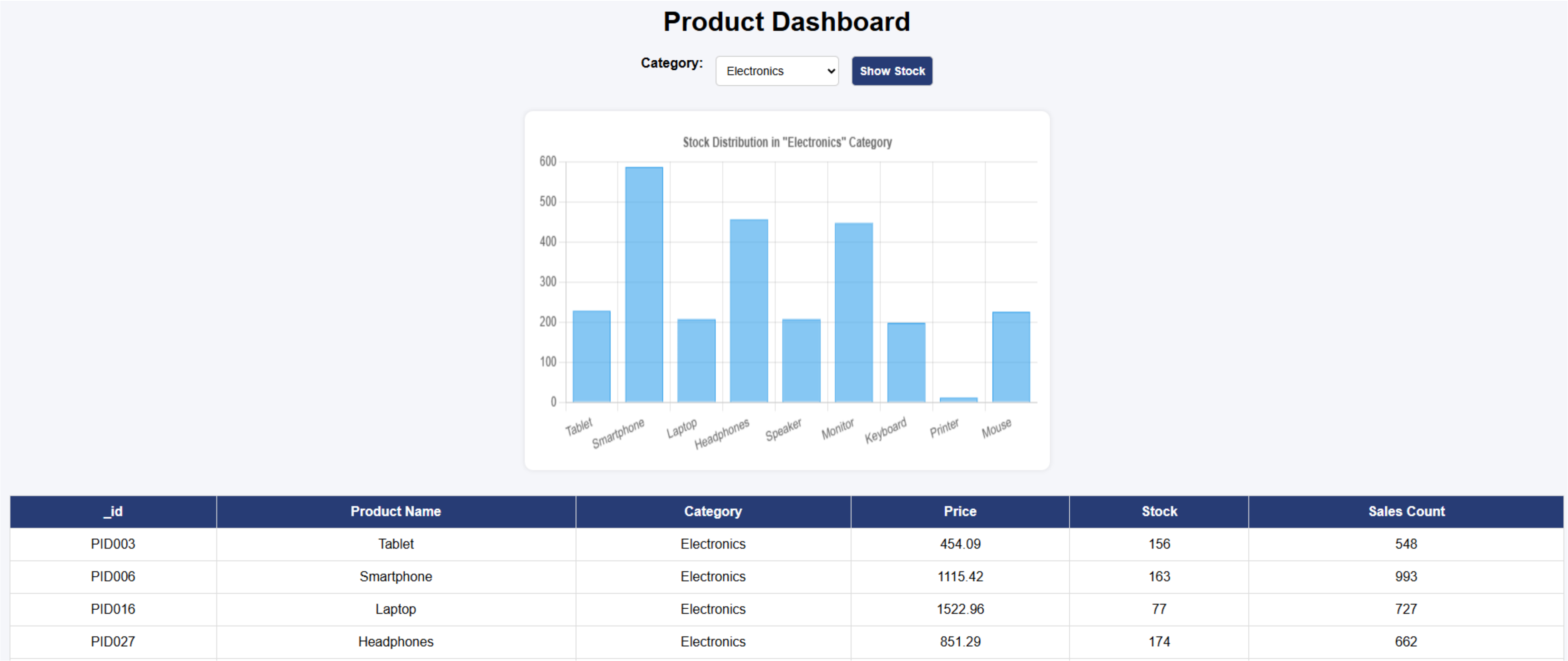


Product Dashboard

_id	Product Name	Category	Price	Stock	Sales Count
PID001	Keyboard	Accessories	1584.4	120	804
PID002	Camera	Mobile Devices	221.39	117	422
PID003	Tablet	Electronics	454.09	156	548
PID004	Mouse	Accessories	982.04	163	306

When the user selects **a specific category** (e.g. Electronics) and clicks **Show Stock** button.

- the chart updates to show stock per product name.
- and the table shows only that category’s products



Product Dashboard

Category: Electronics

_id	Product Name	Category	Price	Stock	Sales Count
PID003	Tablet	Electronics	454.09	156	548
PID006	Smartphone	Electronics	1115.42	163	993
PID016	Laptop	Electronics	1522.96	77	727
PID027	Headphones	Electronics	851.29	174	662

The updated **`server.js`** and **`index.html`** files for the simplified dashboard are available on **Canvas**.

- After updating the *`connectionString`* in `server.js` with your own database connection:
  - The scripts will successfully **read data from your MongoDB database**
  - The data will be **displayed in the table** on your dashboard

⚠ However, at this stage:

  - **The chart is not yet functional — only a placeholder is displayed**

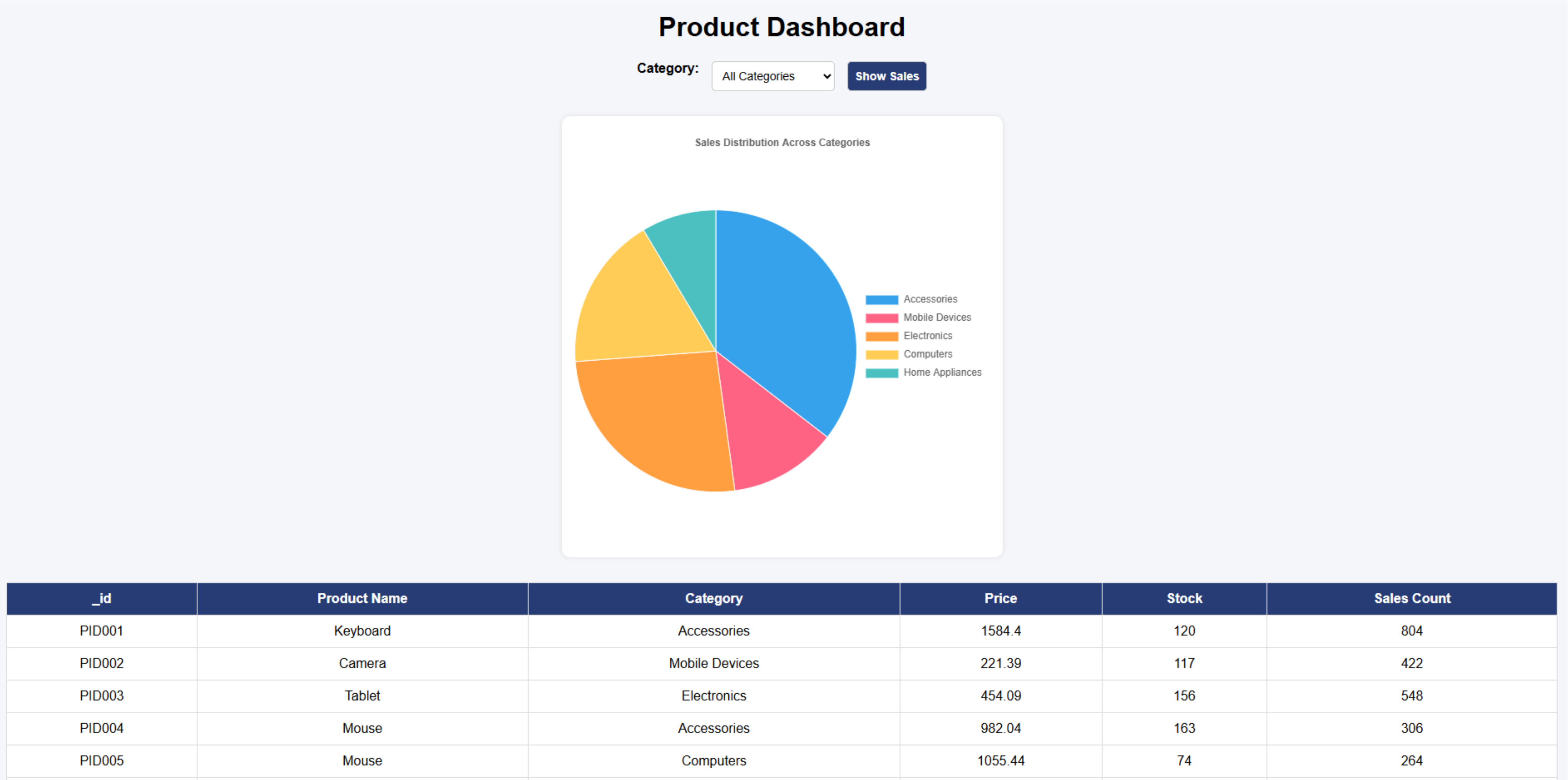
⚠ `Although a partially completed code is provided, you should carefully review the code structure to understand how each part works.`

Now, complete the **`updateDashboard(products, category)`** function in the HTML file as instructed within the comments of the file.

This step will enable the dashboard to dynamically generate and display the charts.

Pie Chart for Sales Distribution by Category

Could you modify the existing code, originally designed for a bar chart, to generate a pie chart that displays the sales distribution across different categories? The expected output should resemble the example shown below.



_id	Product Name	Category	Price	Stock	Sales Count
PID001	Keyboard	Accessories	1584.4	120	804
PID002	Camera	Mobile Devices	221.39	117	422
PID003	Tablet	Electronics	454.09	156	548
PID004	Mouse	Accessories	982.04	163	306
PID005	Mouse	Computers	1055.44	74	264