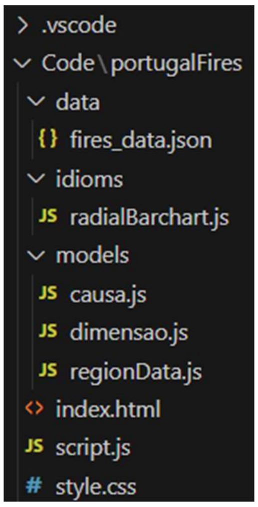
Checkpoint III: First Prototype

Group: 43
Date: 2025/09/26

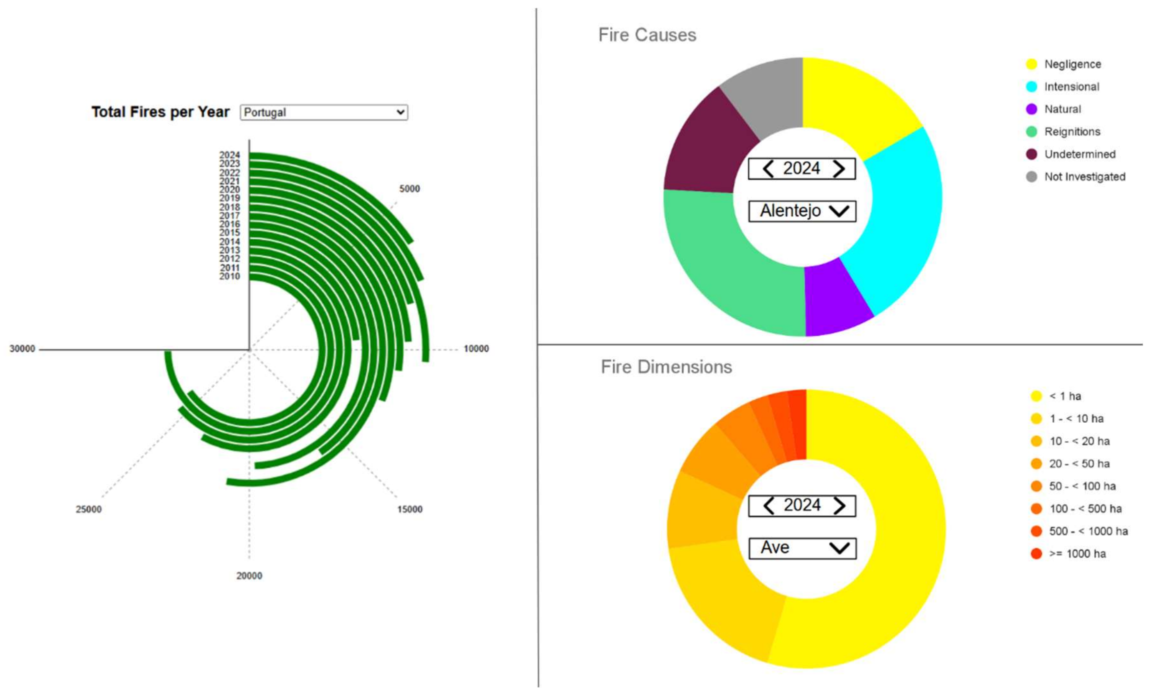
Prototype Architecture



```
> .vscode
v Code \ portugalFires
  v data
    {} fires_data.json
  v idioms
    JS radialBarchart.js
  v models
    JS causa.js
    JS dimensao.js
    JS regionData.js
  <> index.html
  JS script.js
  # style.css
```

- ***fires_data.json*** - contains all our data.
- ***radialBarchart.js*** - implements the *createRadialBarchart()* function. Contains the logic for data aggregation, scales, rendering arcs and labels, and user interactivity (hover highlights, click pop-ups).
- ***causa.js*** - data model for a fire cause (cause name + number of occurrences).
- ***dimensao.js*** - data model for a fire dimension (dimension label + number of occurrences).
- regionData.js - data model for a single region in a given year, storing attributes like burned area, percentage, number of sapadores, efficiency/prevention indexes, total fires, causes, and dimensions.
- ***index.html*** - provides the basic structure of the web application. Loads D3 and all project scripts, defines the container for the charts, and runs init() when the page loads.
- ***script.js*** - acts as the entry point. Fetches and restructures the dataset (*fires_data.json*), creates *RegionData* objects, and initializes the visualizations (currently only the **radial bar chart**).
- ***style.css*** - defines the styling and layout of the application. Uses CSS Grid to organize the chart container, applies a clean font, and styles the chart controls (centered dropdown and label).

Dashboard Layout



Our dashboard includes a **radial bar chart** (the one that is implemented) that displays the total number of fires per year, either for a selected Portuguese sub-region or for the entire country. The user can choose the desired sub-region, or Portugal as a whole, through the dropdown menu located above the chart.

It includes also two **donut charts** (not implemented yet). One of them displays the Fire Causes with bright different colors, because they represent nominal data, the other one displays the dimension of the fires with a continuous color gradient transitioning smoothly from yellow to red, because we are dealing with quantitative data (we will change Fire Dimensions donut chart on CPII improvement submission). The user can choose the desired sub-region and year for each chart, through the dropdown menu located in the middle of the charts.

We also included two lines intentionally to separate the three charts, so that users understand they are independent — changing the filters on one chart only affects that chart.

Data Processing

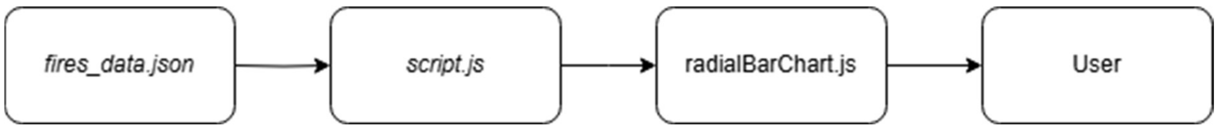
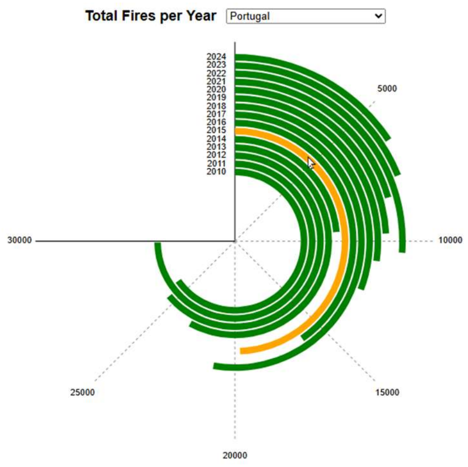
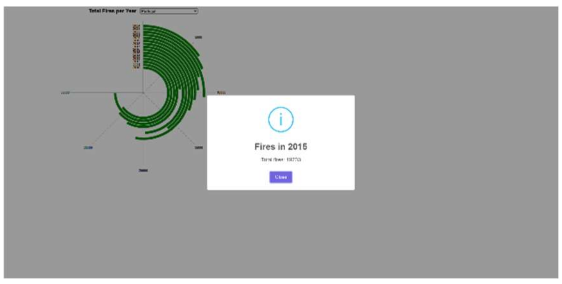
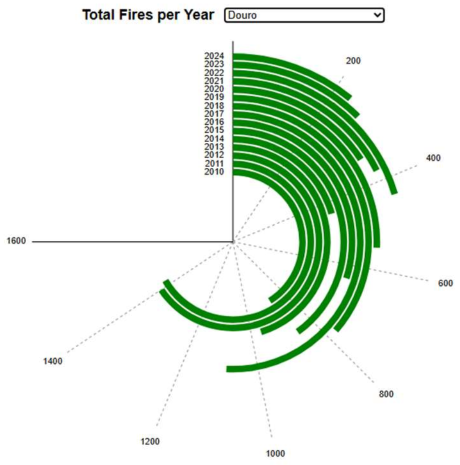


Chart Interaction

The **radial chart** is also **interactive**: when the user hovers over a bar, its color changes to highlight the selection. By clicking on a bar, a pop-up layer appears, showing the exact number of fires represented and providing a button to close the pop-up. Changing the sub-region automatically updates the scale according to the number of fires, keeping the data clear and visible for the user.







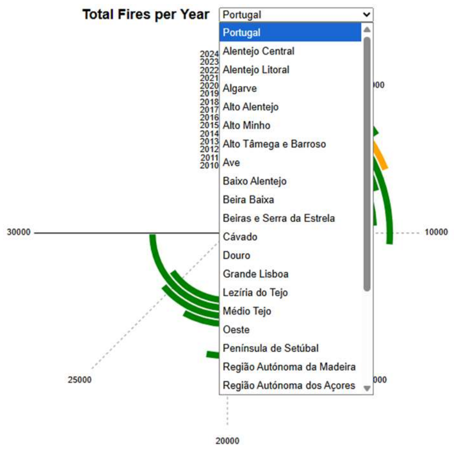


Chart Integration

Techniques used:

- Each chart will be implemented as a **modular function** (createRadialBarchart, donutChart, etc.) that receives structured data and a container selector.
- The **dashboard** separates the views visually and functionally — each chart has its **own container <div>** and **independent update logic**.
- Interactivity is **encapsulated inside each chart module**, so events only affect that specific chart.

Why adding more charts is easy:

- The architecture is **modular**, adding a new chart only requires:
    1. Creating a new chart function (createNewChart(data, container)).
    2. Adding a container <div> in the HTML.
    3. Calling the chart function in script.js with the structured data.
- No changes are needed in the existing chart modules, so each chart **remains independent**.

How the views are linked:

- Currently, in the prototype, the views are **not linked**: changing filters in one chart **does not affect the others**.
- The separation will be explicit, both visually (lines between charts) and functionally (each chart will have its own dropdown and update function).
- If in the future you want to link charts (e.g., filtering one chart updates others), it can be done by creating a **shared state object** in script.js that holds the currently selected filters. Each chart's updateChart function could then **subscribe to changes** in this shared state.

Mechanism for future linking:

- All chart modules already **accept structured data and container as parameters**, which makes it easy to propagate **filter changes** across multiple views.
- When a filter changes, the **shared state** is updated, and all charts that are subscribed to that state can **re-render automatically**.
- This approach works even when more idioms are added, because each chart is **modular and listens to the same shared state**, without breaking existing charts.