# Witches Documentation

* **<map-component>**

All the leaflet map pages that I have re-factored (index, detentions, death, CaseCharacteristics, Case Information) makes use of the main component called <map-component>, which is a customizable component that will take care of the filters box and filtering, the leaflet maps and the timeline if it’s turned on.

* **Props:**

<pageInfo>: Object with the page information. Title for the filters box goes under <title>, text for the info box goes under <html>. The rest is constant across all pages. Example:

Text

Description automatically generated

<originalMarkers>: List of markers generated by loadAccused from the APIDataHandler class. Any given marker has this form:

Graphical user interface, application

Description automatically generated

<location> is the location that leaflet will plot, <longLat> its coordinates, witches an array of its witches, <markerIcon> its icon that will show on the map (might vary as we filter if the filters icons are not constant – will detail this further later), and <active> whether it will show on the map (depends on whether it has witches that are on). Each witch has all of its attributes (sex, socialClass, residence etc), and an object <witchState> that has an entry <activeFilters>, which is array that will include all the filterProperties (will explain what filterProperties are below) that are acting on the witch filtering it off, and <on> entry which determines whether the witch is on.

<filterProperties>: This is the most important prop as it determines the filters that will appear on the filters box, and it is crucial to understand the flow of the code. It is an object of this form (example from index page):

Graphical user interface, text

Description automatically generated

Here, sex, socialClass, occupation and hasWikiPage are filter properties. They will be the titles in the filters box. Each has its own sub-object. Within the sub-object, <label> is the title that will appear. <filters> is an object of filterType objects. To understand what a filterType is, think of this example:

If <filterProperty> = sex, <filterType> = female, then

witch[filterProperty] = <filterType> **equivalent to** witch[sex] = female

So filterProperty is the “property” and filterType is the witch attribute for that property. Each filterType object has its label that will appear in the filters box, whether the filter is active, and its icon URL (might be null depending on the nature of the page). Note that the <filters> object for most filter properties in most pages are built dynamically using APIDataHandler, so whenever that’s the case it must be defined as an empty object in data. This is the case for instance in socialClass and occupation in the above example. Will explain further how to generate the filters using APIDataHandler in the next section.

<filtersGeneralnfo>: Very simple prop where you specify the title that will encapsulate all the filterProperty titles, and whether the filters dropdown is on or off.

<iconBehaviour>: String that determines whether the filters for the page have icons (so that each filterProperty has its own legend when it is set to current in the filters box like in index), in which case the marker icons on the map will change accordingly, or whether the filters for the page don’t have icons, in which case the marker icons are always constant. Has to possible values:

* “constant”: filterTypes don’t have icons and marker icons remain constant.
* “changing”: filterTypes have icons and marker icons change depending on which filter property the user is filtering by.

Note that a given page needs to have all the filterTypes either with or without icons, as of now we don’t allow for a mix, although this is a possible improvement.

Whenever <iconBehaviour> is set to constant, if a given marker has witches that have different filterType attributes for a given filterProperty, the icon for that marker will be purple. Then, whenever we filter on or off and <iconBehaviour> is not constant, the code will automatically detect if a marker has become/stopped becoming mixed (based on <currentProperty>, which is the filterProperty for which icons are currently showing, and change the marker icon accordingly. It does so by calling getMarkerStateIconDependant from FilteringMethods, which returns the marker icon after filtering and whether the marker is on or off based on the witches that are on or off. If <iconBehaviour> is constant instead, we call getMarkerStateNonIconDependant, which just returns whether the marker is on or off based on the witches.

<includeTimeline>: Boolean that determines whether we show the timeline in the filters box. It is set to true by default. We now have an investigation date attribute on each witch, (“N/A”) if the witch doesn’t have an investigation date, so we could in principle add a timeline to every page, and when the user starts filtering with the timeline all the ones that don’t have date would be filtered off (they would be filtered back on again if the timeline is turned off).

The timeline is very responsive for around 1000 markers and under, but if we are dealing with the full amount of markers it struggles slightly. A possible, but ugly, solution would be to remove the reset dates button, and hope that the user focuses on the panic ranges/custom ranges rather than using the timeline on the whole range of dates.

Whenever the user turns the timeline on by selecting a range for it, we automatically turn the leaflet map clusters off.

<clustersOnInitial>: Determines whether the map clusters start on initially. True by default.

* **APIDataHandler**

This class deals with extracting the originalMarkers array and the filterTypes for those filterProperties that we wish to build dynamically. We use it in every page. It deals with all the data processing to get the data from the query into the witches and the witches into the markers.

* **Inputs:**

<queryOutput>: This is the output from the query. At the moment, we are pulling it from a json I produced with python, but eventually we will pull it from the backend PHP with axios. So we need to replace the variable definition of <queryOutput: json> in data for the PHP output in all of the pages (queryOutput can also be defined in mounted before calling loadData). See big-query-output.json in the adding-pages branch for an indication of what the PHP output should look like.

<wikiPages>: An array with witch wiki pages to be able to tell whether the witch has a wiki page or not. The array is built in each page calling loadWikiEntries, which fills up this.wikiPages, which we then pass to the class. Therefore, we need to call loadWikiEntries before creating the class instance. This could be changed to calling loadWikiEntries on the class constructor instead for nicer code.

<icons>: List of icons if the filterTypes are icon dependent. At the moment this is a list with 7 icons from which we pick in filterTypes that are icon dependent.

Text

Description automatically generated

And if a given filterProperty has more than 7 filterTypes, we go back to the first for the 8th and so on. If we wanted to have case-specific icons for the case filters like woodcut cats, we could add another input to the class like <filterSpecificIcons>, which would be usually set to null, but if it weren’t we would read the icons off of it. It would be of the form {filterType: icon}. This would require a bit of implementation, but it would not be hard to do. We would also then need to specify a case-specific mixed icon as well, and enter the mixed icon as a prop to <map-component> (maybe inside filtersGeneralInfo) instead of having the hard-coded purple icon for mixed.

<constantIcon>: If this is not null, we set every marker icon to it. We use this when we build markers for the PrimarySecondary and CaseInformation pages as they have constant icon behaviour.

* **loadAccused method:**

This is the method we call to get the markers and the filterTypes.

* Inputs:

<plotByField>: What we want to plot by. This can be either “residence”, “detention” or “placeOfDeath”. For most pages other than detentions and death we use “residence”. The function takes care of automatically building the markers depending on this input.

<filtersToProduce>: The filterProperties for which we want to build the filterTypes dynamically. It is of this form (example from index page):

Text

Description automatically generated

As we can see, it is a list with sub-lists, each sub-list being a filterProperty and whether it has changing or constant icons. We usually define this list as filtersToFind in data() for every page. Note that the index page has all of them because we save all of them to local storage, so that if you then go to any of the other pages, and local storage has not cleared yet, we can just pull them from local storage whichever filterTypes objects we need. However, for example, in dentention we have:

[ ["socialClass", "changing"],

["occupation", "changing"]]

only, so if you were to land on the page with local storage being cleared, loadAccused would

only fill up the dynamic filterTypes for socialClass and occupation.

* Outputs:

<originalMarkers>: The markers array to plot. Note that the marker icon will only be set if the instance variable <constantIcon> is not null. Therefore, for we usually have a function setMarkersIcons in each page to get the marker icons with marker state functions from FilteringMethods.

<filterProperties>: The filterProperties instance variable of the class, for which the filter properties from <filtersToProduce will have been filled up with filterType objects.

In most pages we then manually select the properties that we want, as it is not too many,

for instance in index we only need socialClass and occupations (although they have all been filled up to save to local storage:

Graphical user interface

Description automatically generated with medium confidence

In pages with more filters like CaseInformation we have the following function to automatically fill in filterProperties:

Graphical user interface, text

Description automatically generated

And then loadData would look like this:

Graphical user interface, text, application, website

Description automatically generated

We could just add the setFilters function to every page and it would make things nicer.

**Splitting CaseInformation into two and adding pages:**

If we wanted to add another page, we would just define the filterProperties and filtersToFind in data() as explained above, and call loadAccused for whichever markers we want and filtersToFind to build the filterTypes to fill up filterProperties. And then just input into <map-component>.

For instance, Ewan and Maggie might like to split CaseInformation into two pages with fewer filters, so then we would just copy and paste the whole of CaseInformation into a new page, and edit filterProperties and filtersToFind in each of them to only include the ones we want on each page. So for instance in one page they might look like this:

Text

Description automatically generated

And in the other like this:

Graphical user interface, text

Description automatically generated

And everything should then adjust automatically.

* **General Flow of Components**

<leaflet-map-main>

Takes care of plotting into the map. Has two sub-components, <clusters-map> and <normal-map>, which are rendered depending on whether clusters are on or off.

<timeline>

Timeline component. Emits filtering on timeline events to <map-component>. Only on if <includeTimeline> is on.

<timeline-range-selector>

Range selector for timeline. Emits events upwards which will eventually reach <timeline> via <map-component>

<normal-filters-list> or <icon-dependent-filters-list>

One or the other depending on icons being constant or not.

<map-filters>

Filters box that displays the filters and keeps track of which ones are active. Emits filtering events to <map-component>. Also includes <timeline-range-selector> and emits selected range/timeline off events.

<map-component>

Takes care of all filtering events that come from both <map-filters> and <timeline>, and inputs filtered markers into <Leaflet-Map-Main>

* **To-do and improvements**
* Fixing nav bar to be made of dropdowns – Ewan and Maggie have ideas as to which menu items we should have on the nav bar.
* Make info icon more visible.
* Include “witch of the day” pop up when user lands. (Same principle as cookie banner, to be added in the <default> layout). Ewan can explain this idea further.
* UI improvements like Stewart’s woodcut icons.
* Try making leaflet map more responsive on the full range. (The loading time comes from having to plot all the markers on each timeline move, not from the filtering. i.e. the more markers the less responsive the timeline. So would involve trying to get leaflet to leave the markers that are not effected by a filtering move on, and just add/remove new ones).