MMISP Frontend Documentation

Design Phase

MMISP Frontend Team

Table of contents

1. Intro 5
2. Design Overview 6
2.1 Components 6
2.2 API 6
2.3 Authentification 6
2.4 Stores 6
2.5 Design 6
2.6 Framework 7
3. Dynamic Layout generation 8
3.1 createTableHeadGenerator 8
4. Sequence Diagrams 10
4.1 Add Event 10
4.2 Edit Event 10
4.3 Filter Events 11
4.4 Publish Event 12
4.5 Add Reference 13
4.6 Freetext Import 14
5. Components 16
5.1 Info 16
5.2 Pill 16
5.3 DatePill 17
5.4 RelativeDatePill 18
5.5 HrefPill 18
5.6 PillCollection 19
5.7 Breadcrumbs 19
5.8 CallbackEntry 20
5.9 ActionBar 20
5.10 ActionBarEntryTemplate 21
5.11 ActionBarEntry 21
5.12 HrefEntry 22
5.13 Boolean 22
5.14 LookupPill 23
5.15 Checkbox 23
5.16 Input 24
5.17 InputCollection 25

5.18 Select 26
5.19 Card 27
5.20 CardRow 27
5.21 Button 28
5.22 AddTagForm 28
5.23 Table 28
5.24 Td 29
5.25 Th 29
5.26 DynTable 30
5.27 DynCard 31
5.28 IconCardRow 31
5.29 FrameNode 32
5.30 BaseNode 33
5.31 ModuleParam 35
5.32 ModuleNode 36
5.33 Flow 37
5.34 EventGraph 38
5.35 FilterCard 38
5.36 Filter 39
5.37 SideMenuDivider 39
5.38 SideMenuEntry 40
5.39 SideMenu 41
5.40 ToggleModeEntry 42
5.41 TopMenu 42
5.42 Layout 43
5.43 ActiveEntry 43
5.44 Pagination 44
5.45 SettingsEntry 44
5.46 ActionModuleNode 45
5.47 TriggerModuleNode 46
5.48 ActionCard 47
5.49 DynActionCard 48
5.50 SelectionCard 48
6. Pages 50
6.1 /admin/keys 50
6.2 /admin/keys/[id] 50
6.3 /admin/keys/new 50
6.4 /admin/servers 51

6.5 /admin/servers/[id] 51
6.6 /admin/servers/new 51
6.7 /admin/users 51
6.8 /admin/users/[id] 52
6.9 /admin/users/new 52
6.10 /attributes 52
6.11 /attributes/[id] 53
6.12 /events 53
6.13 /events/[id] 54
6.14 /events/new 54
6.15 /galaxies 54
6.16 /galaxies/[id] 55
6.17 /galaxies/clusters/[id] 55
6.18 /settings 56
6.19 /tags 56
6.20 /tags/[id] 56
6.21 /workflows 57
6.22 /workflows/[id] 57
6.23 /workflows/modules 57
6.24 /workflows/modules/[id] 58
6.25 /workflows/triggers 58
6.26 /workflows/triggers/[id] 58
6.27 /login 59
7. Layouts 60
7.1 / 60
7.2 / 60
7.3 /workflows/[id] 60
8. Error Pages 61
8.1 / 61

1. Intro

The project "Modern MISP Frontend" is implemented as a client-side rendered SvelteKit application.

SvelteKit applications are constructed out of Components. They are `.svelte` files found in the `src` directory of this project. See here for general documentation about Svelte Components.

All routes of the application are represented by specific Components called Pages. They are the files called `+page.svelte` in the `src/routes` directory of this project. See here for general documentation about SvelteKit Pages.

Pages "inherit" from specific Components called Layouts placed higher up in the route tree by automatically being placed into the Layout's default slot. They are the files called `+layout.svelte` in the `src/routes` directory of this project. See here for general documentation about SvelteKit Layouts.

Note that the inter-component dependencies and the specifics of exposed props, slots and events may not fully represent the final state of the application, as agreed upon with our supervisors.

2. Design Overview

The frontend represents the "view" part of the Model-View-Controller architecture in regard to the overall MISP project. Therefore we do not have a very complex architecture.

2.1 Components

All components placed inside the `src/lib/components` directory (`Components` box in the uml diagram) are atomic, meaning they do not create any side effects and are usually (close to) stateless. As a way to achieve a similar effect to inheritance between components, dependency injection is used throughout the application. As an example, a `RelativeDatePill` is a kind of `DatePill` wich is a `Pill`. This kind of "inheritance" is achieved by injecting the more generalized components into the more specific components.

2.2 API

Communication with the API is achieved via the browser Fetch API. The library openapi-fetch is used to automatically ensure type safety for all API calls according to the agreed upon OpenAPI spec. We build an adapter around that function to be able to enable and disable specific api methods. There is a custom wrapper around that function to provide advanced capabilities like the option to enable and disable specific HTTP methods. HTTP methods are configurable with the `PUBLIC_REST_DISABLED` environment variable. The format is `{[httpMethod]: boolean}`

The API auth key (`PUBLIC_MISP_KEY`) and the MISP API endpoint (`PUBLIC_MISP_API_ENDPOINT`) are provided via environment variables, and are required to exist.

2.3 Authentification

When the app loads, the `+layout.page` file checks if a token is set in the `localStorage`. If it is set, the user gets redirected to the default page. If it isn't set, the user gets redirected to the `/login` page instead.

2.4 Stores

All the application state is centralized in the `stores.ts` file via Svelte Stores. All stores are global singletons that can be accessed throughout the application.

The exported stores are:

- settings: All settings of the application. The possible values are user-configurable. The settings page will generate the correct layout from this object.
- mode: "view" or "edit"
- currentRoute: The route currently displayed.
- actionBarEntries: All entries that are supposed to be shown inside the `ActionBar`.
- user: The current user with permissions. The type of this value is extracted from the OpenAPI specification.

2.5 Design

The initial design work was done with Figma. A prototype is visible here.

2.6 Framework

The application is built as a SvelteKit application, but without using any of SvelteKit's server functionality. Please refer to the docs.

3. Dynamic Layout generation

Most layouts (especially lists and cards) in the appliction are generated using "dynamic" components like `DynTable` and `DynCard`.

These components allow generating layouts by mapping the properties of the provided data onto `TableHead`s, which are objects describing how to display that data.

`TableHead` objects contain a `value` function and an optional `display` property which allow mapping the data to either a string, in which case the data is diplayed as that string, or to a `Component` constructor and its props, which displays the data using that component and essentially allows data to be displayed in every possible way desired.

This approach allows easily programmatically setting up `Table`s and `Card`s without manually having to specify the whole page layout as HTML, greatly increasing maintainability and reusability.

```
type TableHead<
  Value,
  DisplayComp extends SvelteComponent | undefined = SvelteComponent | undefined,
  DefaultValueReturn = string
> = {
  label: string;
  display?: DisplayComp extends SvelteComponent ? ComponentType<DisplayComp> : undefined;
  value: (
    value: Value
  ) => DisplayComp extends SvelteComponent ? ComponentProps<DisplayComp> : DefaultValueReturn;
};
```

3.1 createTableHeadGenerator

The `createTableHeadGenerator` function is an essential utility function that allows generating/validating type safe `TableHead`s.

It is generic over the type of data that is supposed to be displayed, which is usually the autogenerated type returned by the OpenAPI fetch client, and ensures that all generated `TableHead`s can only attempt to display properties of the data that are actually available on the provided type.

Signature:

```
function createTableHeadGenerator<
  T,
  E extends Record<string, unknown> = Record<string, unknown>
>(): <K extends SvelteComponent | undefined>(tableHead: TableHead<T, K> & E) => TableHead<T, K> & E;
```

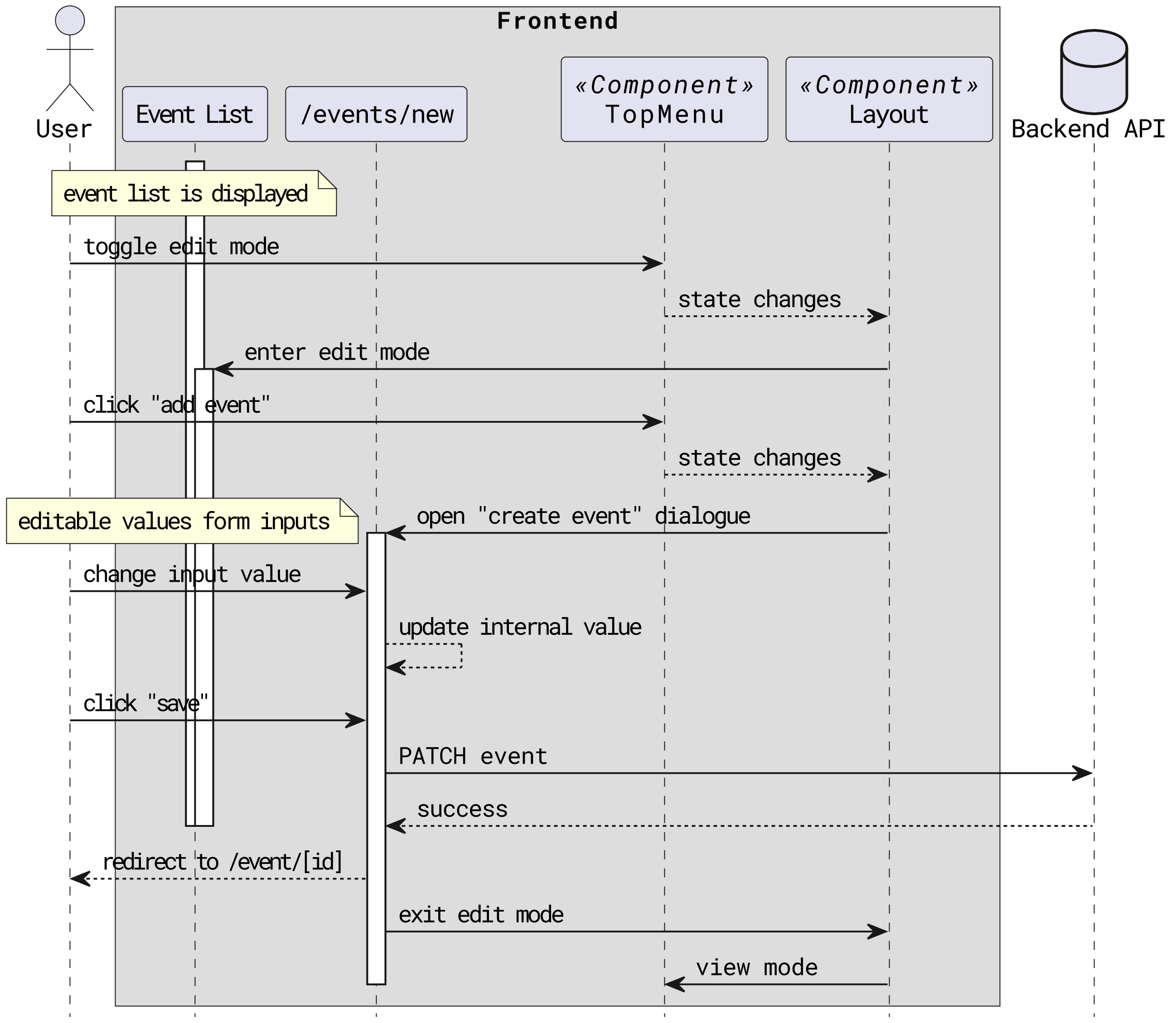
Note that due to several constraints in TypeScript, `createTableHeadGenerator` is a wrapper function returning the actual function responsible for validating the `TableHead`. In practice though, this is mostly a convencience instead of an inconvenience, because it allows binding the type of the data to the outer function once and then receiving the reusable inner function that is already bound to the correct type, resulting in ergonomic code like this:

```
const data = /* data from some kind of source, usually the MISP API */;

/* Binds the `col` function to `typeof data` */
const col = createTableHeadGenerator<typeof data>();

/* Type checks correctly and only allows using properties available on `data` */
const header = [
  col({
    icon: 'mdi:id-card',
    key: 'name',
    label: 'Name',
    value: (x) => ({ text: x.name ?? 'unknown' }),
    display: Info
  }),
  col({
    icon: 'mdi:telescope',
    key: 'scope',
    label: 'Scope',
    value: (x) => ({ icon: 'mdi:telescope', text: x.scope ?? 'unknown' }),
    display: Pill
  }),
  col({
    icon: 'mdi:head-alert',
    key: 'overhead',
    label: 'Overhead',
    value: (x) => ({ value: x.trigger_overhead, options: THREAT_LEVEL_LOOKUP }),
    display: LookupPill
  })
];
```

4. Sequence Diagrams

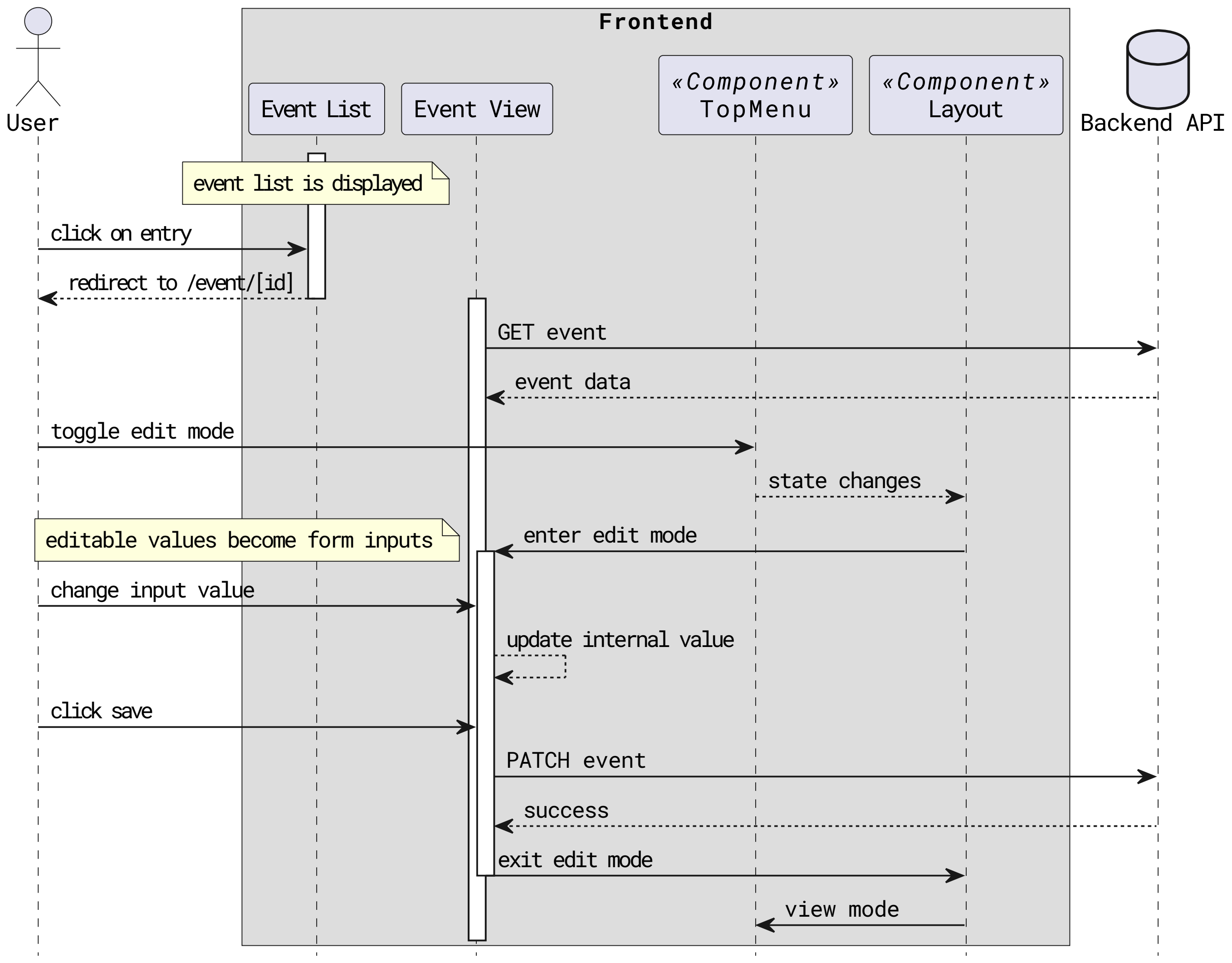
4.1 Add Event

Add a new event to the Events-List. Only Allowed after switching to Edit-Mode.



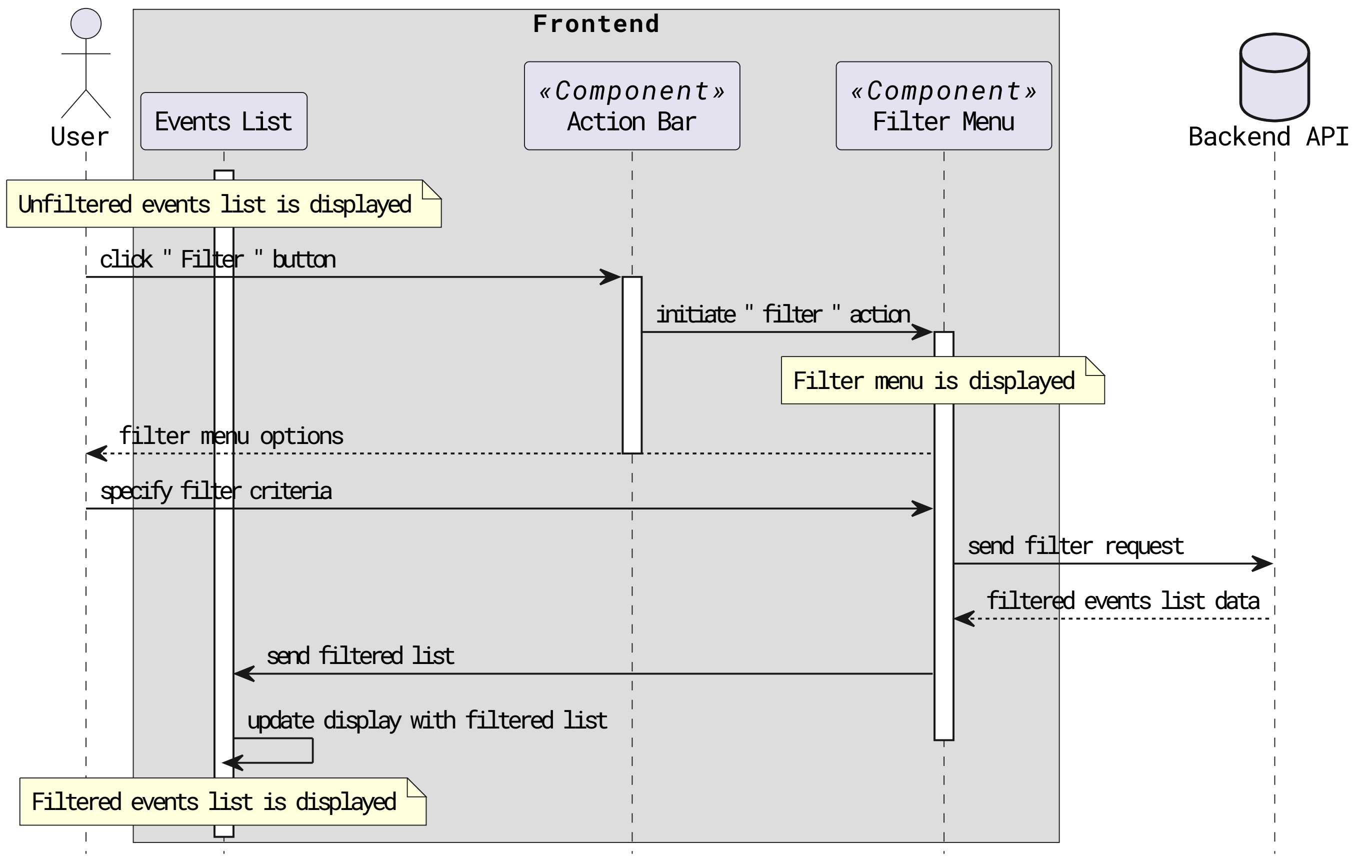
4.2 Edit Event

Edit an already existing event. Only Allowed after switching to Edit-Mode.



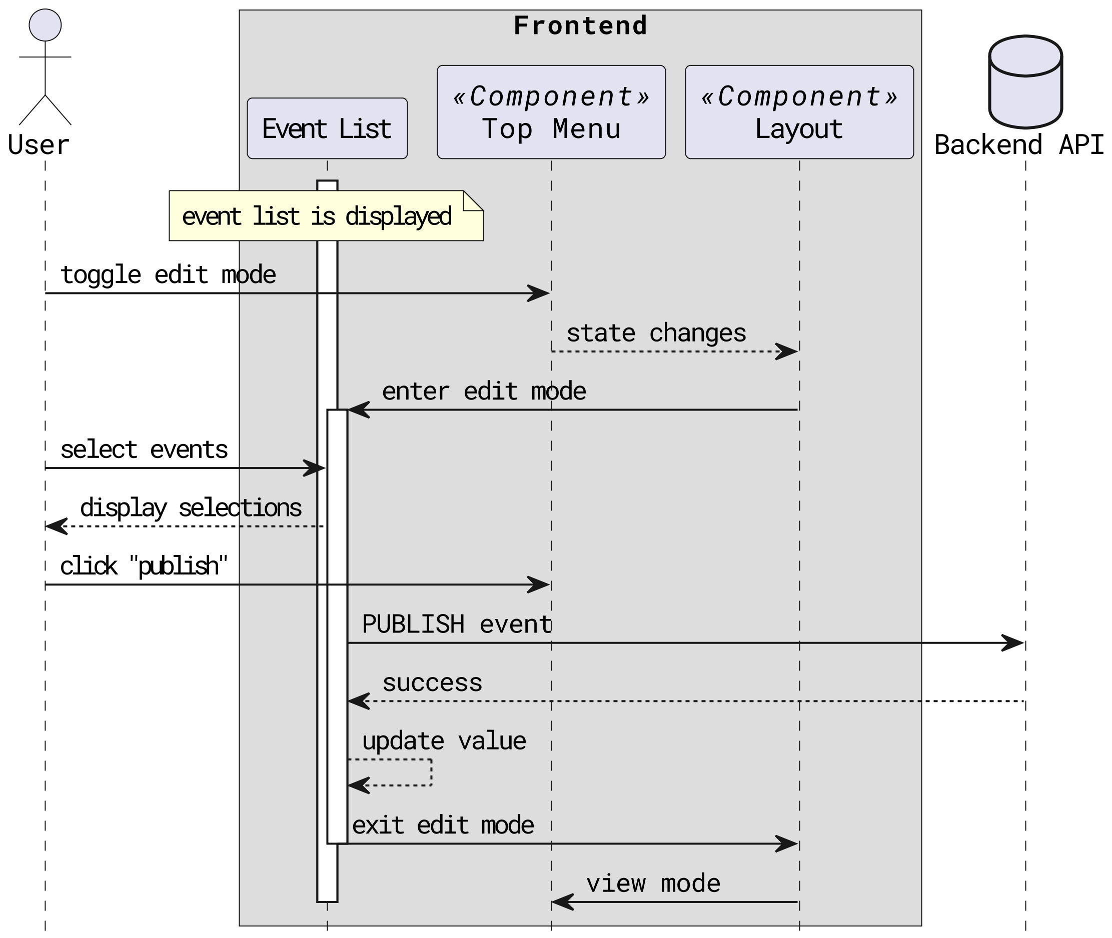
4.3 Filter Events

Filter events of the Events-List according to some given criteria.



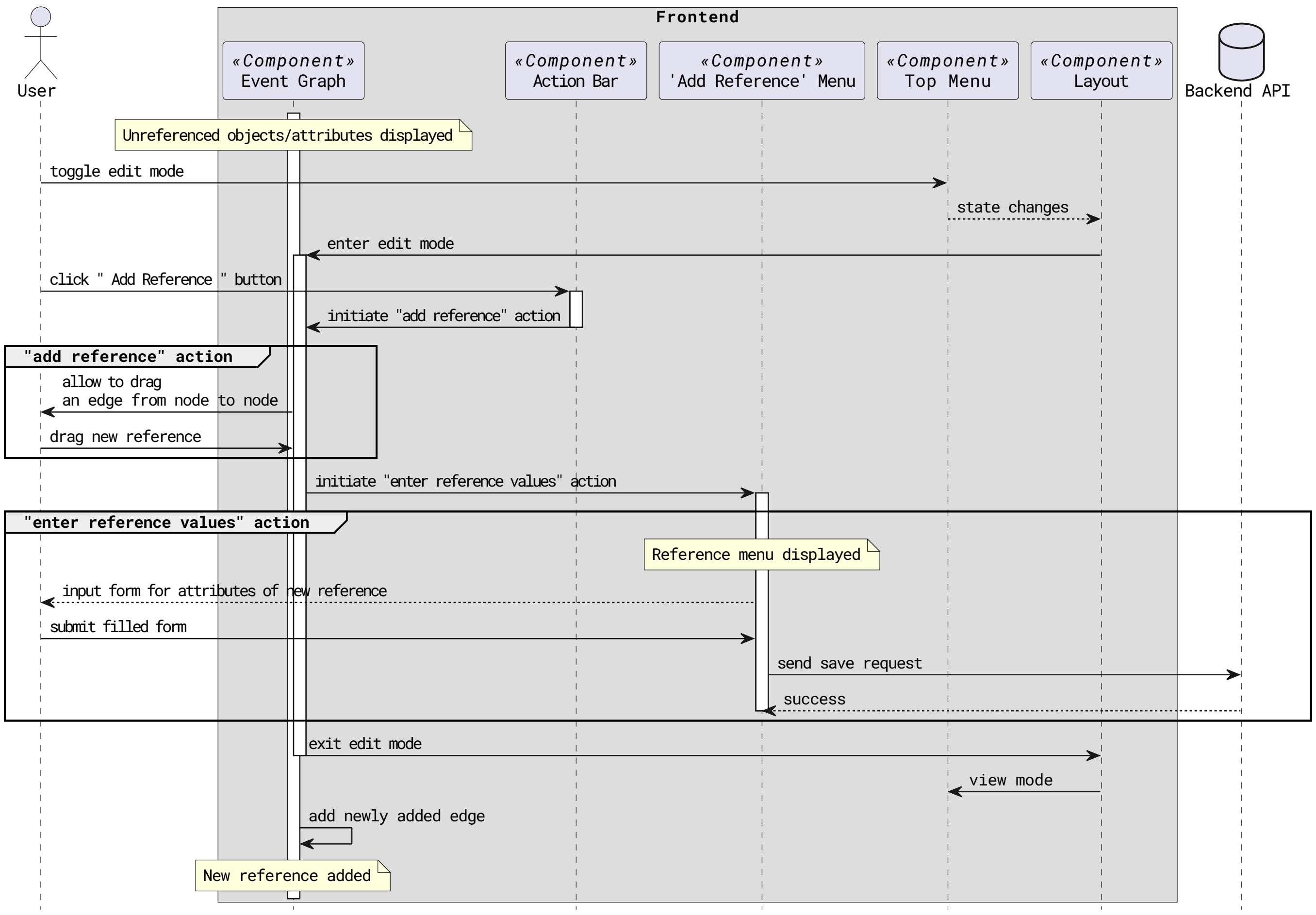
4.4 Publish Event

Publish an already existing event.



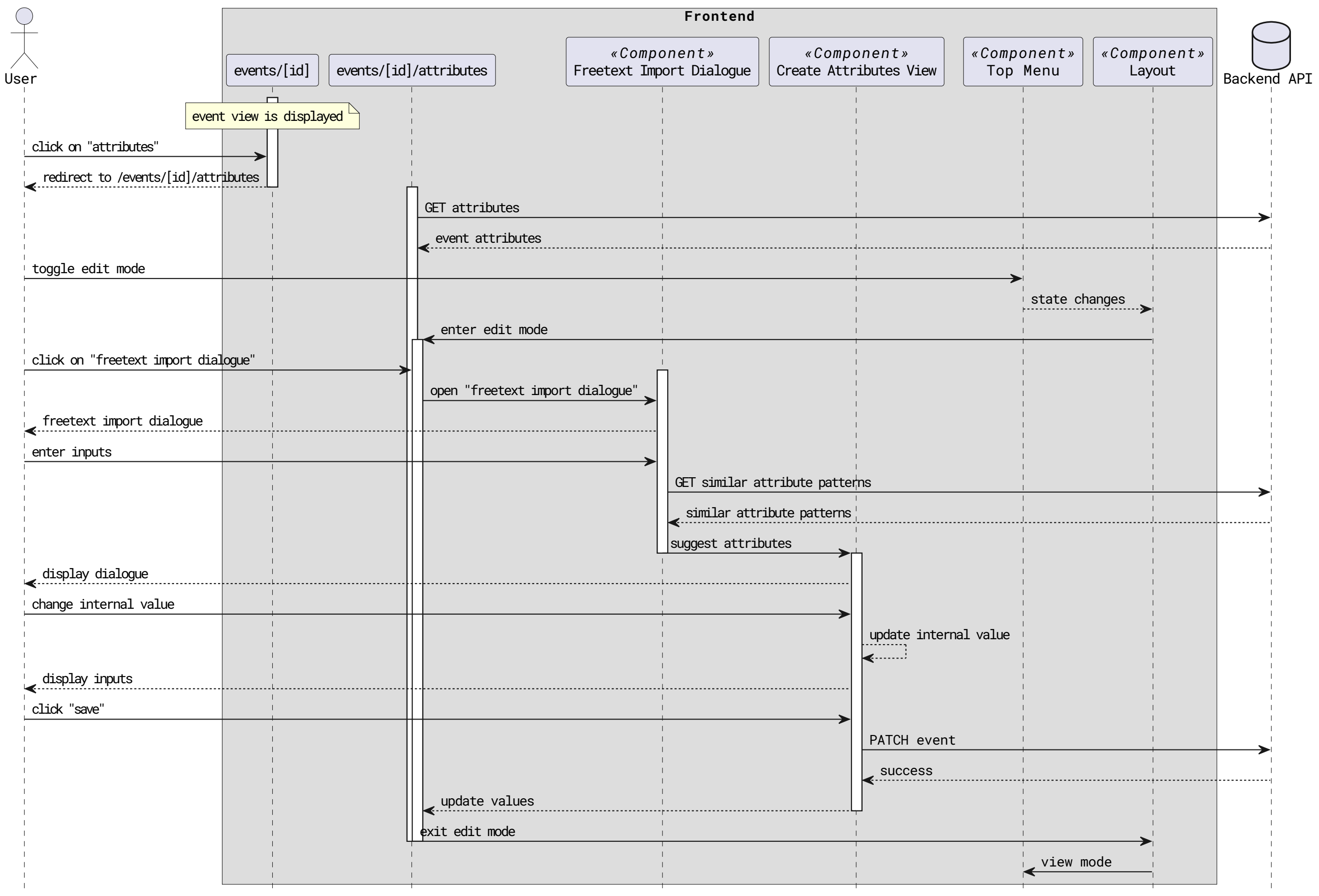
4.5 Add Reference

Add a new reference between an object and attribute in an Event-Graph. Only Allowed after switching to Edit-Mode.



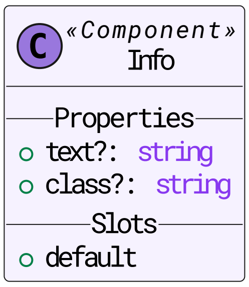
4.6 Freetext Import

Suggest attribute patterns of an event according to IoCs ("Indicator of Compromise"). The attributes can be edited and added to the event. Only Allowed after switching to Edit-Mode.


Frontend
User
events/[id]
events/[id]/attributes
«Component» Freetext Import Dialogue
«Component» Create Attributes View
«Component» Top Menu
«Component» Layout
Backend API
event view is displayed
click on "attributes"
redirect to /events/[id]/attributes
GET attributes
event attributes
toggle edit mode
state changes
enter edit mode
click on "freetext import dialogue"
open "freetext import dialogue"
freetext import dialogue
enter inputs
GET similar attribute patterns
similar attribute patterns
suggest attributes
display dialogue
change internal value
update internal value
display inputs
click "save"
PATCH event
success
update values
exit edit mode
view mode

5. Components

5.1 Info



Displays a text with a default background color of `surface1`. Also sets the default padding and border radius. You can override this by passing your own classes.

5.1.1 Props

text: `string | undefined`

The text to be displayed.

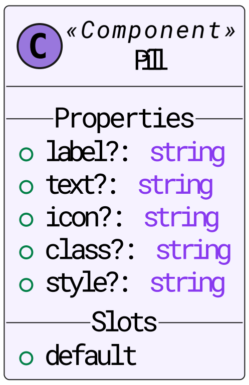
class: `string | undefined`

Additional classes to be applied.

5.1.2 Slots

default

5.2 Pill



A pill component. A pill is a small rounded rectangle with a label and/or text and/or icon.

Slot:

The content of the pill. If no slot is provided, the text prop will be used.

5.2.1 Props

label: `string | undefined`

The label of the pill. Will be placed on the left side of the pill. The background of the label is `bg-crust`.

text: `string | undefined`

The text of the pill. Will be placed in the middle of the pill.

icon: `string | undefined`

The icon of the pill. Will be placed on the left side of the pill. If a label is present, the icon will be placed on the left side of the label.

class: `string | undefined`

Class that should be applied to the pill.

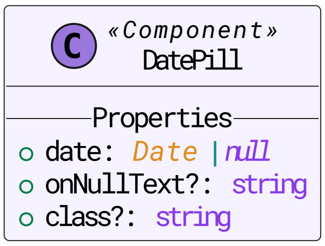
style: `string | undefined`

Some style overrides. When possible, the `class` prop should be used instead.

5.2.2 Slots

default

5.3 DatePill



Uses Pill.

Displays a date in a pill with the default format. The date format can be configured in the `config.ts` file.

5.3.1 Props

date: `Date | null`

The date of the to be displayed.

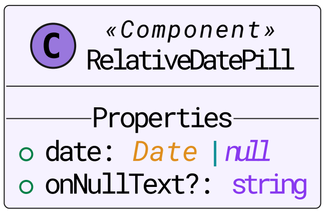
onNullText: `string | undefined`

The text that should be displayed if the date is null.

class: `string | undefined`

Class that should be applied to the pill.

5.4 RelativeDatePill



Uses DatePill.

Displays a relative date in a pill. The color of the pill is based on the date.

- If the date is in the past, the pill will be red.
- If the date is over one week in the future, the pill will be green.
- If the date is less then one week in the future, the pill will be orange.

5.4.1 Props

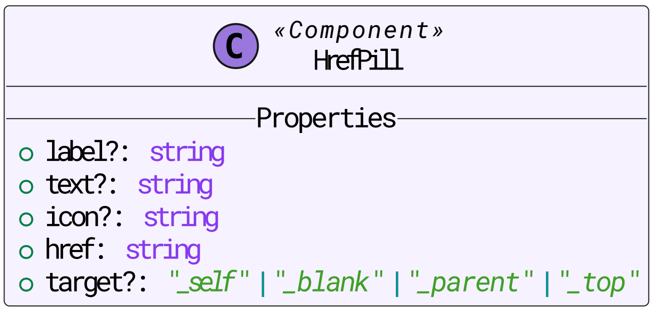
date: `Date | null`

The date of the to be displayed.

onNullText: `string | undefined`

The text that should be displayed if the date is null.

5.5 HrefPill



Uses Pill.

A pill component that acts as a link. This pills text will be blue:

5.5.1 Props

label: `string | undefined`

The label of the pill. Will be placed on the left side of the pill. The background of the label is `bg-crust`.

text: `string | undefined`

The text of the pill. Will be placed in the middle of the pill.

icon: `string | undefined`

The icon of the pill. Will be placed on the left side of the pill. If a label is present, the icon will be placed on the left side of the label.

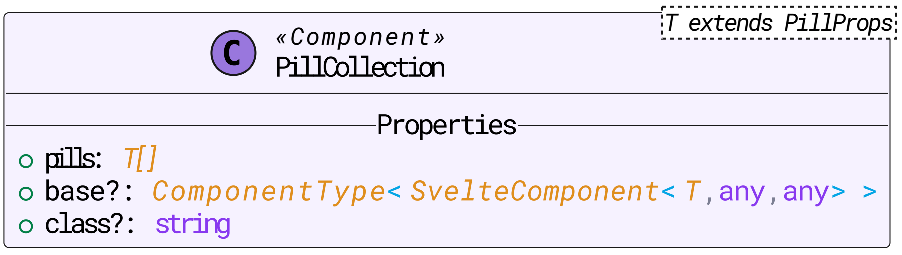
href: `string`

The target URL of the pill, which will be navigated to when the pill is clicked.

target: `"_self" | "_blank" | "_parent" | "_top" | undefined`

The target browsing context i.e. where to open the URL.

5.6 PillCollection



Uses Pill.

Displays a collection of pills. The pill component that should be used for each pill can be specified by setting the `base` prop.

5.6.1 Props

Generics: `T extends PillProps`

pills: `T[]`

The Pills that should be displayed.

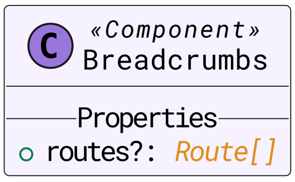
base: `ComponentType<SvelteComponent<T, any, any>> | undefined`

The pill component to be used for each pill.

class: `string | undefined`

The class of the pill wrapper.

5.7 Breadcrumbs



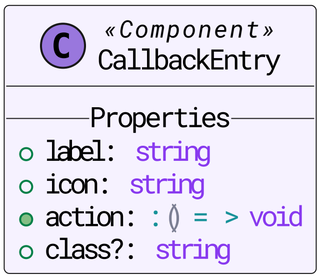
Displays a breadcrumb trail with the given routes.

5.7.1 Props

routes: `Route[] | undefined`

The route that will be displayed in the breadcrumbs.

5.8 CallbackEntry



Uses ActionBarEntry.

An ActionBarEntry with an `on:click` callback action associated with it.

5.8.1 Props

label: `string`

The label of this ActionBar entry.

icon: `string`

The icon of this ActionBar entry.

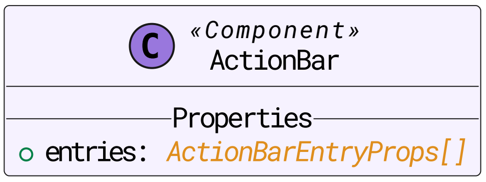
action: `() => void`

Callback function that is executed on click.

class: `string | undefined`

The class of this ActionBar entry.

5.9 ActionBar



Uses CallbackEntry, HrefEntry.

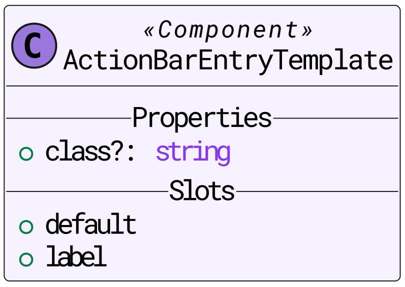
The action bar contains actions that can be performed when in edit mode.

5.9.1 Props

entries: `ActionBarEntryProps[]`

Actions that are displayed.

5.10 ActionBarEntryTemplate



5.10.1 Props

class: `string | undefined`

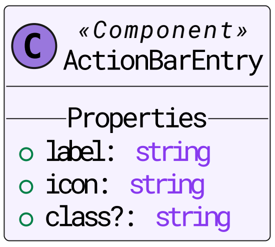
The class of this ActionBar entry.

5.10.2 Slots

default

label

5.11 ActionBarEntry



Uses ActionBarEntryTemplate.

Represents one of the entries of the ActionBar.

5.11.1 Props

label: `string`

The label of this ActionBar entry.

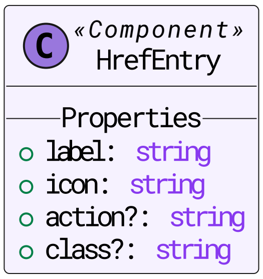
icon: `string`

The icon of this ActionBar entry.

class: `string | undefined`

The class of this ActionBar entry.

5.12 HrefEntry



Uses ActionBarEntry.

An ActionBarEntry that acts as a link to the specified URL.

5.12.1 Props

label: `string`

The label of this ActionBar entry.

icon: `string`

The icon of this ActionBar entry.

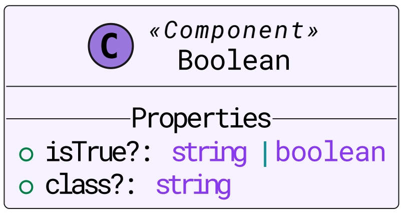
action: `string | undefined`

URL for hyperlink

class: `string | undefined`

The class of this ActionBar entry.

5.13 Boolean



Uses Pill.

Displays a boolean value as a text using the Pill component. The background is green if the value is true and red if the value is false.

5.13.1 Props

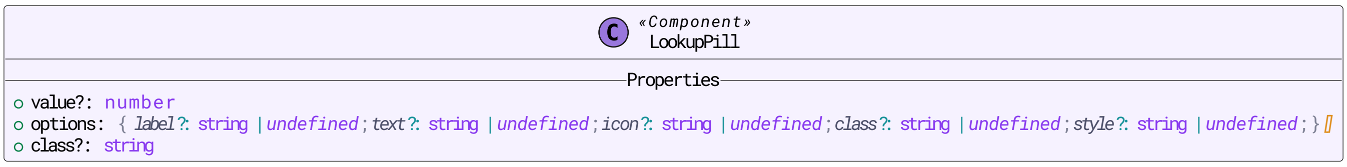
isTrue: `string | boolean | undefined`

Displays a boolean value as a text using the Pill component. Also parses strings to booleans. String must be either 'true' or 'false'.

class: `string | undefined`

Additional classes to be applied to the Pill component.

5.14 LookupPill


«Component»
LookupPill
Properties
value?: number
options: { label?: string | undefined; text?: string | undefined; icon?: string | undefined; class?: string | undefined; style?: string | undefined; }[]
class?: string

Uses Pill.

Converts the value given by the `value` prop to an entry from the `options` lookup array and displays the result as a pill.

5.14.1 Props

value: `number | undefined`

The index corresponding to the entry in the `options` array.

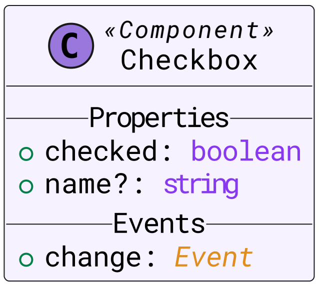
options: `{ label?: string | undefined; text?: string | undefined; icon?: string | undefined; class?: string | undefined; style?: string | undefined; }[]`

Array with props of Pills, indexed by `value`.

class: `string | undefined`

The class of the pill.

5.15 Checkbox



A checkbox component. In order to receive changes, the `checked` prop can be reactively bound or the `on:change` event can be listened to for changes.

Internal:

Uses some tailwind css trickery to make the checkbox value to look like a switch. Basically hides the input and sets the focus state via the label. The div is the actual switch and is moved via the peer-checked class where the peer class is set in the input.

5.15.1 Props

checked: `boolean`

Whether the checkbox is checked or not.

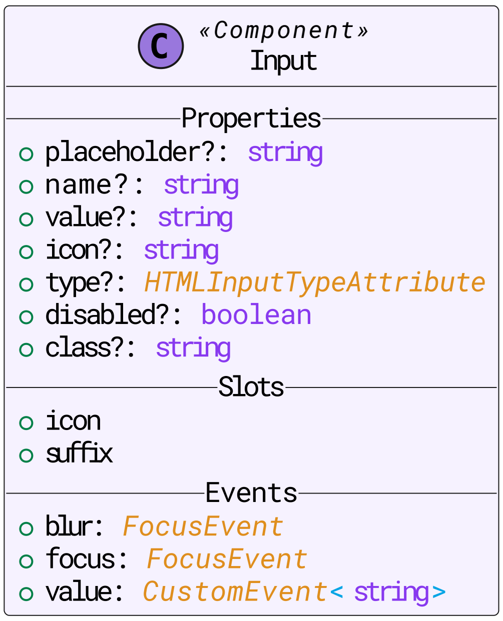
name: `string | undefined`

The form name of this checkbox.

5.15.2 Events

change: `Event`

5.16 Input



The default input component. A prefix icon can be added inside of the `icon` slot, and/or a suffix icon in the `suffix` slot.

In order to use this component in forms, the `name` prop should be set.

You can also set the value prop, if you want to set an initial value. Or bind to it if you want to use this outside of a form. You can also set the placeholder prop, if you want to set an placeholder.

5.16.1 Props

placeholder: `string | undefined`

Placeholder of the input.

name: `string | undefined`

The name of the input. Used for the label and for form submission.

value: `string | undefined`

The current value of the input.

icon: `string | undefined`

The icon to be displayed inside of the input.

type: `HTMLInputTypeAttribute | undefined`

The type of the input.

disabled: `boolean | undefined`

When true, the input is disabled an cannot be changed.

class: `string | undefined`

Additional classes to be applied.

5.16.2 Slots

icon

suffix

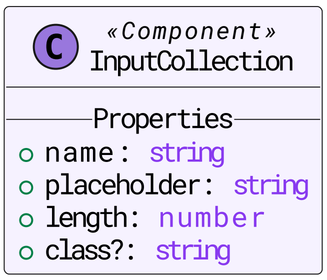
5.16.3 Events

blur: `FocusEvent`

focus: `FocusEvent`

value: `CustomEvent<string>`

5.17 InputCollection



Uses Input.

5.17.1 Props

name: `string`

The name of this input array

placeholder: `string`

The placeholder of the inputs

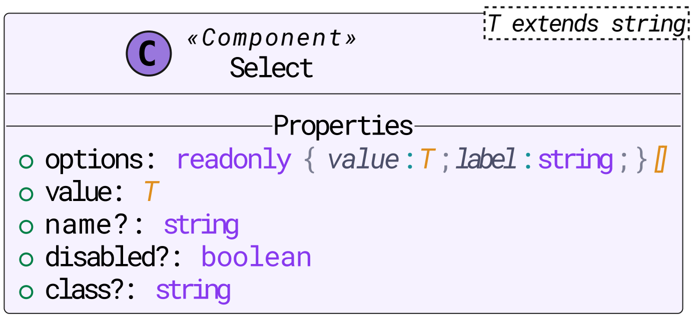
length: `number`

The amount of inputs

class: `string | undefined`

The class overload for each input

5.18 Select



A select component that uses the native select element. The options are passed as a prop and the value is bound to the `value` prop. The options prop should be an `as const` array of objects with a value and a label property to allow full type safety.

5.18.1 Props

Generics: `T extends string`

options: `readonly { value: T; label: string; }[]`

The options of the select. The value is the value of the option and the label is the label of the option.

value: `T`

The value that is currently selected. Because of the template variable, full type safety should be enforced if using `const`s as options.

name: `string | undefined`

Name of this `select` element. Used for forms.

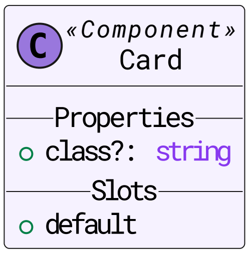
disabled: `boolean | undefined`

When true, selection is disabled.

class: `string | undefined`

The class of the select element.

5.19 Card



A card with a slot for content. Sets the default padding and border radius. You can override this by passing your own classes.

5.19.1 Props

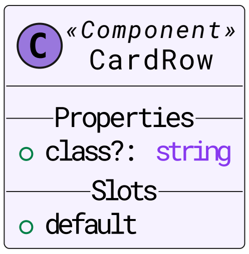
class: `string | undefined`

Additional classes to be applied to this component.

5.19.2 Slots

default

5.20 CardRow



This component should be used to display rows inside of a Card.

It's recommended to only use up to two children in the slot, which will be displayed at both ends of the row.

5.20.1 Props

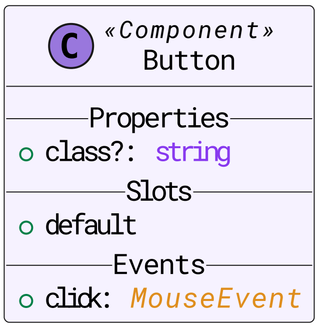
class: `string | undefined`

Additional classes to be applied to this component.

5.20.2 Slots

default

5.21 Button



A button with a slot for content.

5.21.1 Props

class: `string | undefined`

Additional classes to be applied to this component.

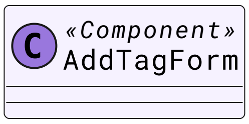
5.21.2 Slots

default

5.21.3 Events

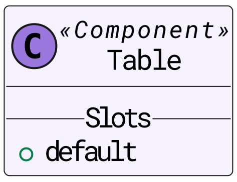
click: `MouseEvent`

5.22 AddTagForm



Uses Card, CardRow, Input, Select, Checkbox, Button, Pill.

5.23 Table



Creates an HTML `table` element with specific styling.

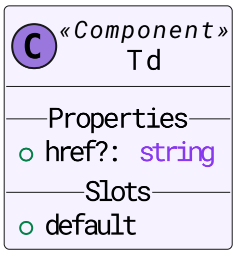
Slot:

The full table.

5.23.1 Slots

default

5.24 Td



Creates an HTML `td` element for the table with specific styling.

Slot:

- The content of the td.

5.24.1 Props

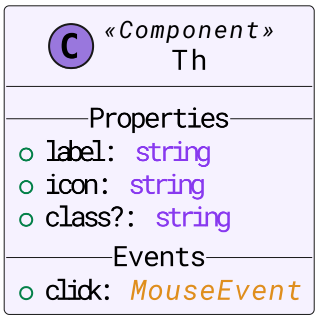
href: `string | undefined`

The url to navigate to when clicking on the `td`.

5.24.2 Slots

default

5.25 Th



Creates an HTML `th` element for the table with specific styling.

Slot:

- The content of the th.

5.25.1 Props

label: `string`

The label of the column.

icon: `string`

The icon of the column.

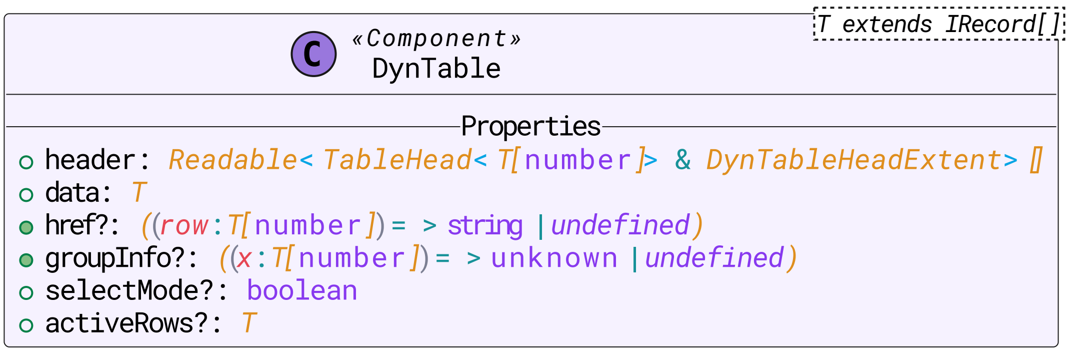
class: `string | undefined`

The class to be applied to the table head.

5.25.2 Events

click: `MouseEvent`

5.26 DynTable


T extends IRecord[]
C
«Component»
DynTable
Properties
header: Readable< TableHead< T[number]> & DynTableHeadExtent> []
data: T
href?: ((row:T[number])= > string |undefined)
groupInfo?: ((x:T[number])= > unknown |undefined)
selectMode?: boolean
activeRows?: T

Uses Table, Td, Th.

Creates a dynamic Table using the `header` and `data` props.

The `header` props specifies the columns of the table, while the `data` prop provides rows of data that conform to the structure of the header.

Type safety of this is enforced at compile time using Typescript.

Can enable selectMode. Where you can navigate to href by double click, and it adds the row to activeRows on click. Removes it, if it is present.

5.26.1 Props

Generics: `T extends IRecord[]`

header: `Readable<TableHead<T[number]> & DynTableHeadExtent>[]`

The header of the table. Also includes the icon and the href. When setting this, it's recommended to use the `createTableHeadGenerator` util function.

data: `T`

The data that will be displayed in the table.

href: `((row: T[number]) => string | undefined) | undefined`

The callback that will be called when the user clicks on the row.

groupInfo: `((x: T[number]) => unknown | undefined) | undefined`

The callback that will be called to determine if the row should be grouped with other rows, and what info to show

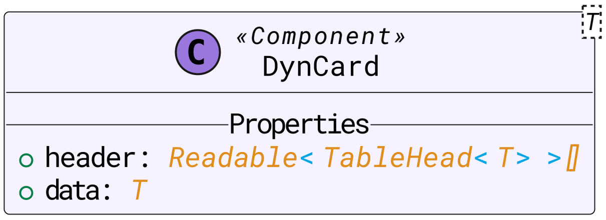
selectMode: `boolean | undefined`

Is the table in select mode. Aka. Select rows by single click. Navigate to href on double click

activeRows: `T | undefined`

currentlyActive rows. Should probably bind to this.

5.27 DynCard



Uses Card, CardRow.

A card that displays the data of the given header.

This works dynamically similar to the DynTable component. So you should probably use the `createTableHeadGenerator` util function to create the header.

5.27.1 Props

Generics: `T`

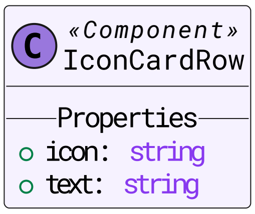
header: `Readable<TableHead<T>>[]`

The header of the table. Also includes the icon and the href.

data: `T`

The data that will be displayed in the table.

5.28 IconCardRow



Uses CardRow.

5.28.1 Props

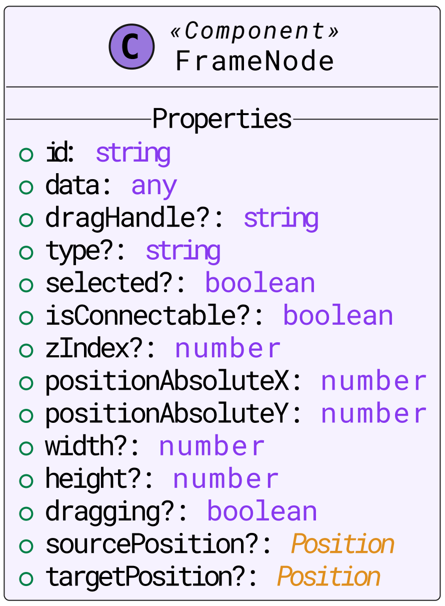
icon: `string`

The icon to be displayed in this row.

text: `string`

The text to be displayed in this row.

5.29 FrameNode



A node representing a workflow frame.

5.29.1 Props

id: `string`

Id of the node

data: `any`

dragHandle: `string | undefined`

type: `string | undefined`

selected: `boolean | undefined`

isConnectable: `boolean | undefined`

zIndex: `number | undefined`

positionAbsoluteX: `number`

positionAbsoluteY: `number`

width: `number | undefined`

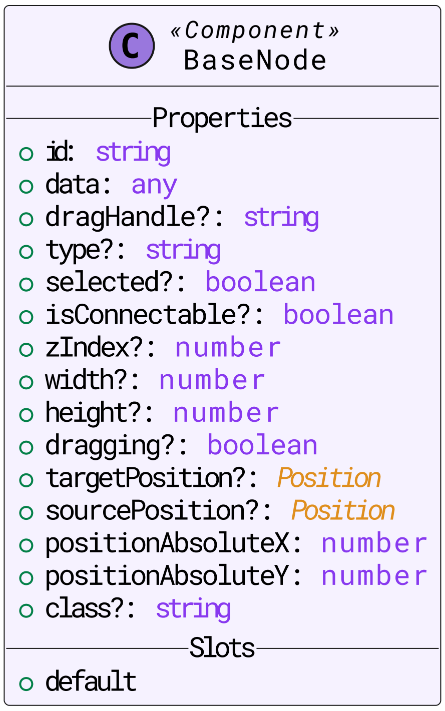
height: `number | undefined`

dragging: `boolean | undefined`

sourcePosition: `Position | undefined`

targetPosition: `Position | undefined`

5.30 BaseNode



The base component for all custom diagram nodes. Other custom node types should use this as their container.

5.30.1 Props

id: `string`

Node id

data: `any`

Node data

dragHandle: `string | undefined`

Node drag handle

type: `string | undefined`

Node type

selected: `boolean | undefined`

Node selected

isConnectable: `boolean | undefined`

Node is connectable

zIndex: `number | undefined`

Node z index

width: `number | undefined`

Node width

height: `number | undefined`

Node height

dragging: `boolean | undefined`

Node dragging

targetPosition: `Position | undefined`

Node target position

sourcePosition: `Position | undefined`

Node source position

positionAbsoluteX: `number`

Node absolute x position

positionAbsoluteY: `number`

Node absolute y position

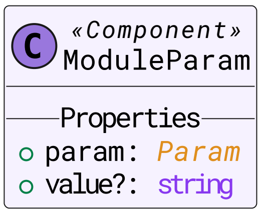
class: `string | undefined`

Additional classes applied to outer div.

5.30.2 Slots

default

5.31 ModuleParam



Uses Select, Input.

Parameter of a workflow module.

5.31.1 Props

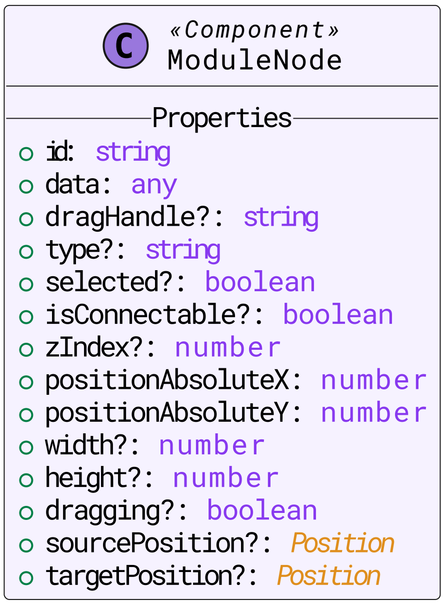
param: `Param`

The parameter data from the workflow module.

value: `string | undefined`

The already supplied value for an indexed parameter.

5.32 ModuleNode



Uses BaseNode, ModuleParam.

A node representing a generic workflow module.

5.32.1 Props

id: `string`

Id of the node

data: `any`

dragHandle: `string | undefined`

type: `string | undefined`

selected: `boolean | undefined`

isConnectable: `boolean | undefined`

zIndex: `number | undefined`

positionAbsoluteX: `number`

positionAbsoluteY: `number`

width: `number | undefined`

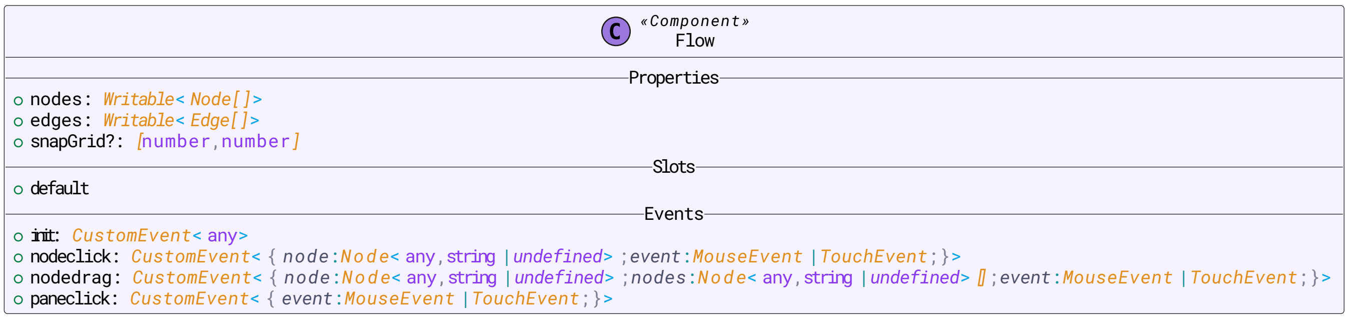
height: `number | undefined`

dragging: `boolean | undefined`

sourcePosition: `Position | undefined`

targetPosition: `Position | undefined`

5.33 Flow


«Component»
Flow
Properties
nodes: Writable< Node[ ]>
edges: Writable< Edge[ ]>
snapGrid?: [number,number]
Slots
default
Events
init: CustomEvent< any>
nodeclick: CustomEvent< { node:Node< any,string |undefined> ;event:MouseEvent |TouchEvent;}>
nodedrag: CustomEvent< { node:Node< any,string |undefined> ;nodes:Node< any,string |undefined> [] ;event:MouseEvent |TouchEvent;}>
paneclick: CustomEvent< { event:MouseEvent |TouchEvent;}>

Uses FrameNode, ModuleNode.

This component contains a node-based editor or interactive diagram provided by SvelteFlow.

It acts like a canvas. All elements, such as nodes, edges and controls, are rendered inside.

5.33.1 Props

nodes: `Writable<Node[]>`

Nodes that are rendered on the flow

edges: `Writable<Edge[]>`

Edges that are rendered on the flow

snapGrid: `[number, number] | undefined`

Dimensions of the grid that nodes will snap onto

5.33.2 Slots

default

5.33.3 Events

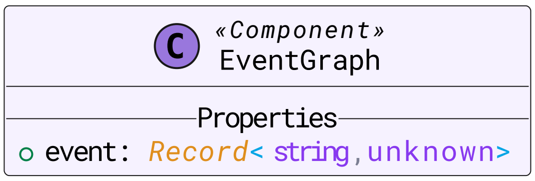
init: `CustomEvent<any>`

nodeclick: `CustomEvent<{ node: Node<any, string | undefined>; event: MouseEvent | TouchEvent; }>`

nodedrag:
`CustomEvent<{ node: Node<any, string | undefined>; nodes: Node<any, string | undefined>[]; event: MouseEvent | TouchEvent; }>`

paneclick: `CustomEvent<{ event: MouseEvent | TouchEvent; }>`

5.34 EventGraph



Uses Card, IconCardRow, Flow.

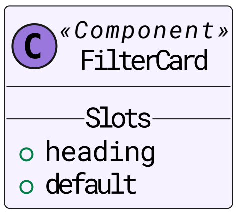
The event Graph component. Uses the Flow component to render the graph. Uses the Card component to render the action Bar and a table, where the unreferenced objects and attributes are displayed.

5.34.1 Props

event: `Record<string, unknown>`

The Event to be displayed on this page.

5.35 FilterCard



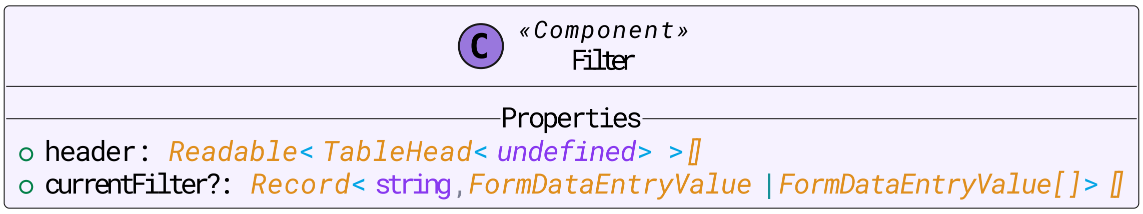
Uses Card.

5.35.1 Slots

heading

default

5.36 Filter


«Component»
Filter
Properties
header: Readable< TableHead< undefined> >[]
currentFilter?: Record< string, FormDataEntryValue |FormDataEntryValue[ ]> []

Uses Button, FilterCard, Select, Input.

5.36.1 Props

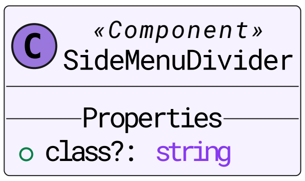
header: `Readable<TableHead<undefined>>[]`

All possible filter

currentFilter: `Record<string, FormDataEntryValue | FormDataEntryValue[]>[] | undefined`

The current filter values. You should probably bind this.

5.37 SideMenuDivider

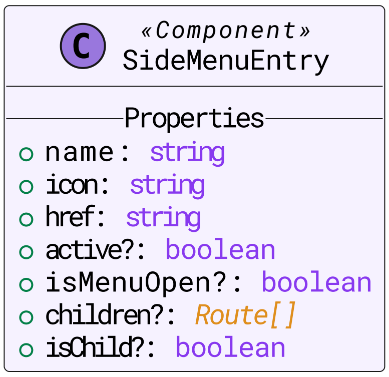


A divider for the side menu.

5.37.1 Props

class: `string | undefined`

5.38 SideMenuEntry



Uses SideMenuDivider.

The side menu entry component.

It can be opened by clicking on it when the parent side menu is open.

When open, all the children will be displayed as subentries using this component.

5.38.1 Props

name: `string`

The name to be displayed in this side menu entry.

icon: `string`

The icon to be displayed in this side menu entry.

href: `string`

The href to be used in this side menu entry.

This is the URL of the page this entry links to.

active: `boolean | undefined`

Whether this side menu entry is active or not.

Active entries are highlighted visually.

isMenuOpen: `boolean | undefined`

Whether the parent side menu is open or not.

children: `Route[] | undefined`

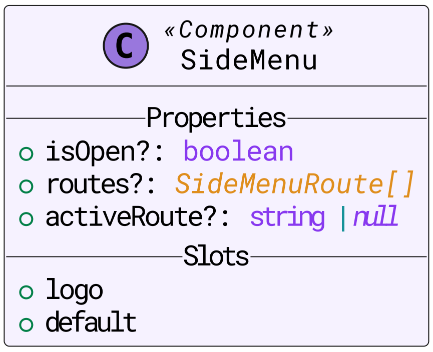
The children of this side menu entry.

Will be displayed as subentries.

isChild: `boolean | undefined`

Whether this side menu entry is a child of another SideMenuEntry, meaning it is a subentry.

5.39 SideMenu



Uses SideMenuDivider, SideMenuEntry.

The side menu component. It contains the SideMenuEntry and SideMenuDivider components.

You can override the default SideMenuEntry list display by using the default slot.

You can also override the logo by using the `logo` slot.

Internal:

When setting a logo, do not forget to set the global `FADE_OPTIONS` constant, otherwise it may look weird.

5.39.1 Props

isOpen: `boolean | undefined`

The current state of the side menu.

Can be bound in order to change the state from other components.

routes: `SideMenuRoute[] | undefined`

The routes to be displayed in the side menu.

activeRoute: `string | null | undefined`

The current route that is active.

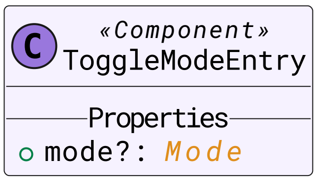
Should usually be the current URL provided by SvelteKit (`$page.url.href`).

5.39.2 Slots

logo

default

5.40 ToggleModeEntry



Uses Checkbox, ActionBarEntryTemplate.

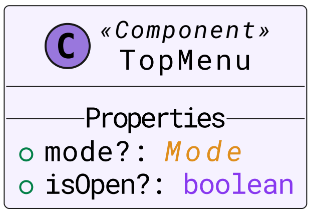
The ActionBar entry responsible for toggling modes.

5.40.1 Props

mode: `Mode | undefined`

The current mode of this Entry.

5.41 TopMenu



Uses Input, ActionBar, ToggleModeEntry.

The top menu component.

The search bar and the ActionBar are located here.

5.41.1 Props

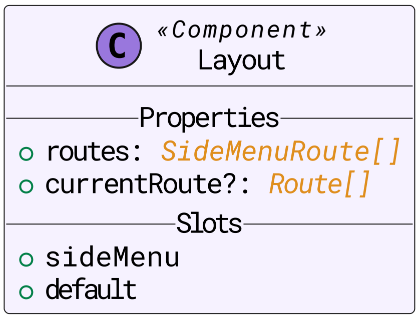
mode: `Mode | undefined`

The mode of the current page. Possible modes are currently "view" and "edit": TODO: maybe extract this to a store?

isOpen: `boolean | undefined`

Whether the side menu is open or not. TODO: probably should search for a better solution for this.

5.42 Layout



Uses SideMenu, TopMenu, Breadcrumbs.

The basic component for the layout of the application.

This Component is intended to be used in Layouts, where the page body will automatically be inserted into the default slot.

You can also override the SideMenu by using the `sideMenu` slot.

5.42.1 Props

routes: `SideMenuRoute[]`

The routes to be displayed in the side menu.

currentRoute: `Route[] | undefined`

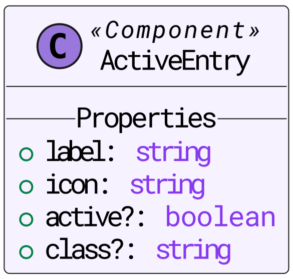
The current route to be displayed in the Breadcrumbs.

5.42.2 Slots

sideMenu

default

5.43 ActiveEntry



Uses ActionBarEntry.

An ActionBarEntry with an `on:click` callback action associated with it.

5.43.1 Props

label: `string`

The label of this ActionBar entry.

icon: `string`

The icon of this ActionBar entry.

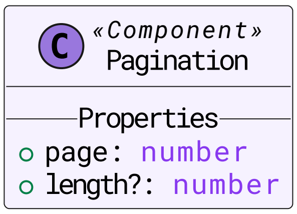
active: `boolean | undefined`

Is this entry active. You should bind to this

class: `string | undefined`

The class of this ActionBar entry.

5.44 Pagination



A pagination component that allows the user to navigate through pages of a list.

5.44.1 Props

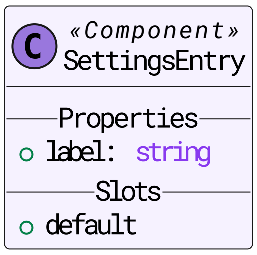
page: `number`

The current page.

length: `number | undefined`

The total number of pages.

5.45 SettingsEntry



Represents a single settings entry on `/settings` .

Slot:

The component/html (e.g. a Checkbox or Select to display in the entry.

5.45.1 Props

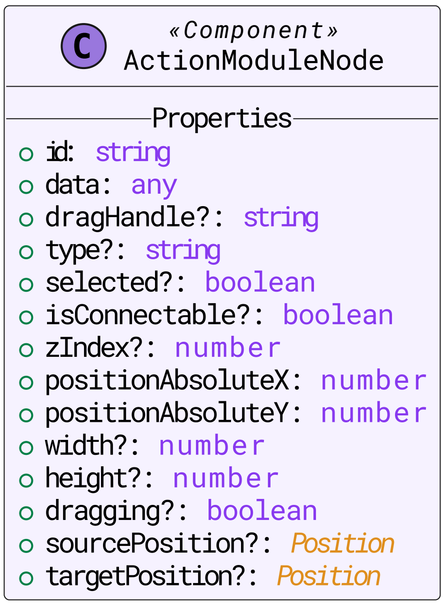
label: `string`

The label of the settings entry.

5.45.2 Slots

default

5.46 ActionModuleNode



Uses BaseNode.

A node representing a workflow module.

5.46.1 Props

id: `string`

Id of the node

data: `any`

dragHandle: `string | undefined`

type: `string | undefined`

selected: `boolean | undefined`

isConnectable: `boolean | undefined`

zIndex: `number | undefined`

positionAbsoluteX: `number`

positionAbsoluteY: `number`

width: `number | undefined`

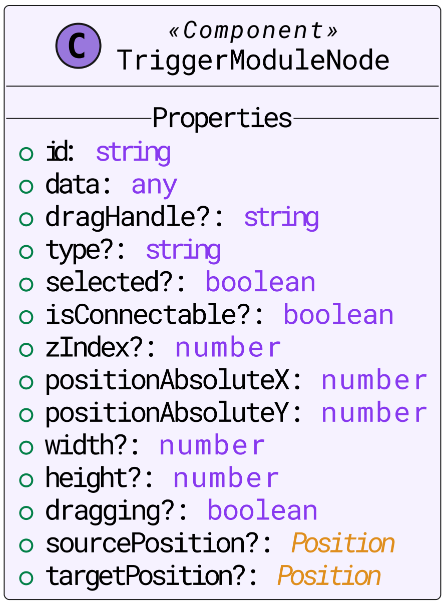
height: `number | undefined`

dragging: `boolean | undefined`

sourcePosition: `Position | undefined`

targetPosition: `Position | undefined`

5.47 TriggerModuleNode



Uses BaseNode.

A node representing a workflow trigger.

5.47.1 Props

id: `string`

```
Id of the node
```

data: `any`

dragHandle: `string | undefined`

type: `string | undefined`

selected: `boolean | undefined`

isConnectable: `boolean | undefined`

zIndex: `number | undefined`

positionAbsoluteX: `number`

positionAbsoluteY: `number`

width: `number | undefined`

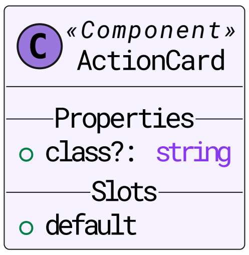
height: `number | undefined`

dragging: `boolean | undefined`

sourcePosition: `Position | undefined`

targetPosition: `Position | undefined`

5.48 ActionCard



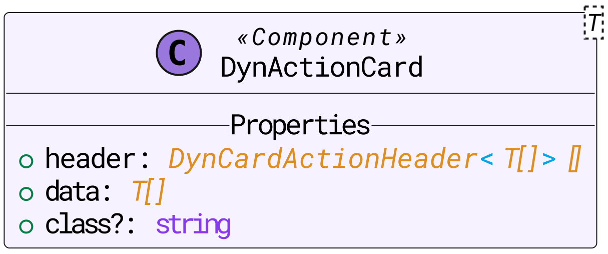
5.48.1 Props

class: `string | undefined`

5.48.2 Slots

default

5.49 DynActionCard



Uses CallbackEntry, ActionCard.

5.49.1 Props

Generics: `T`

header: `DynCardActionHeader<T[]>[]`

The header of the table. Also includes the icon and the href.

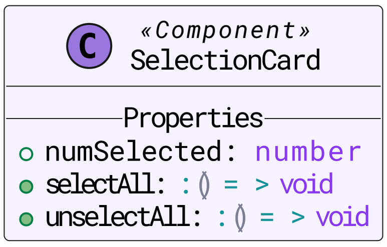
data: `T[]`

The data that will be displayed in the table.

class: `string | undefined`

Class overload

5.50 SelectionCard



Uses CallbackEntry, ActionCard.

Display number of selected rows alongside actions for selecting/unselecting all rows.

5.50.1 Props

numSelected: `number`

```
Number of rows that are currently selected.
```

selectAll: `() => void`

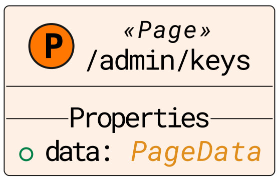
```
Callback for selecting all rows.
```

unselectAll: `() => void`

```
Callback for unselecting all rows.
```

6. Pages

6.1 /admin/keys

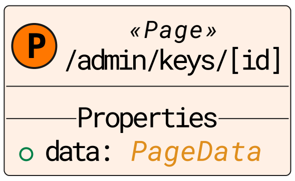


Uses DynTable, CallbackEntry, ActionCard.

6.1.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; header: (Readable<TableHead<{ AuthKey?: { id?: string | undefined; uuid?: string | undefined; authkey_start?: string | undefined; authkey_end?: string | undefined; ... 6 more ...; last_used?: string | ... 1 more ... | undefined; } | undefined; User?: { ...; } |...`

6.2 /admin/keys/[id]



Uses DynCard.

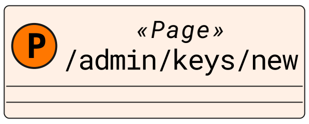
Displays information about a specific auth key, specified by `id`.

6.2.1 Props

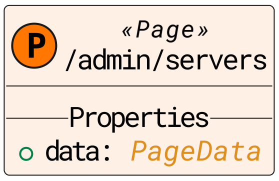
data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; key: { AuthKey?: { id?: string | undefined; uuid?: string | undefined; authkey_start?: string | undefined; authkey_end?: string | undefined; ... 6 more ...; last_used?: string | ... 1 more ... | undefined; } | undefined; User?: { ...; } | undefined; }; left: (R...`

Data that is displayed on this page.

6.3 /admin/keys/new



6.4 /admin/servers



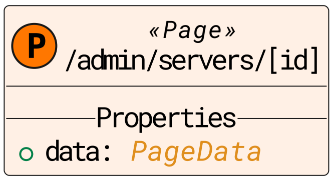
Uses DynTable.

Displays a list of all remote servers of the instance.

6.4.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; data: { Server?: ({ id?: string | undefined; } & { name?: string | undefined; url?: string | undefined; authkey?: string | undefined; org_id?: string | undefined; ... 20 more ...; cache_timestamp?: boolean | undefined; }) | undefined; Organisation?: ({ ...; } &...`

6.5 /admin/servers/[id]



Uses DynCard.

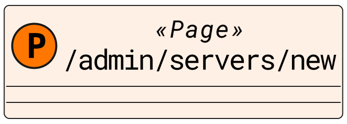
Displays information about a specific remote server, specified by `id`.

6.5.1 Props

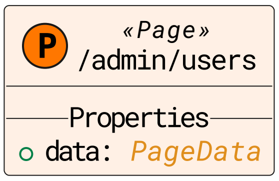
data: `{ [x: string]: any; }`

Data that is displayed on this page.

6.6 /admin/servers/new



6.7 /admin/users



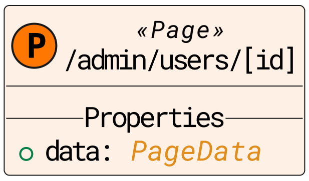
Uses DynActionCard, SelectionCard, DynTable.

Displays a list of all users of the instance.

6.7.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; data: { User?: ({ id?: string | undefined; } & { org_id?: string | undefined; server_id?: string | undefined; email?: string | undefined; ... 17 more ...; date_modified?: string | undefined; }) | undefined; Role?: { ...; } | undefined; Organisation?: { ...; } |...`

6.8 /admin/users/[id]



Uses DynCard.

Displays information about a specific user, specified by `id`.

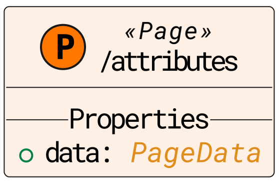
6.8.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; user: { id?: string | undefined; } & { org_id?: string | undefined; server_id?: string | undefined; email?: string | undefined; autoalert?: boolean | undefined; ... 16 more ...; date_modified?: string | undefined; } & { ...; }; left: (Readable<...> | Readable<....`

Data that is displayed on this page.

6.9 /admin/users/new

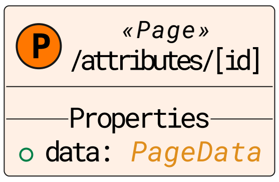
6.10 /attributes



6.10.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; tableData: ({ id?: string | undefined; } & { event_id?: string | undefined; object_id?: string | undefined; object_relation?: string | null | undefined; ... 12 more ...; last_seen?: string | ... 1 more ... | undefined; })[]; }`

6.11 /attributes/[id]

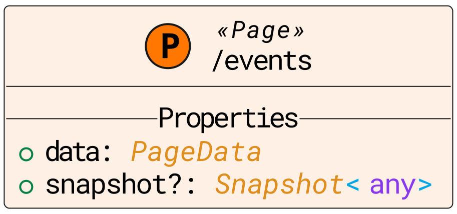


Uses DynCard.

6.11.1 Props

data:
`{ [x: string]: any; [x: number]: any; [x: symbol]: any; header: (Readable<TableHead<{ id?: string | undefined; } & { event_id?: string | undefined; object_id?: string | undefined; object_relation?: string | null | undefined; ... 12 more ...; last_seen?: string | ... 1 more ... | undefined; }, undefined> & Record<......`

6.12 /events



Uses Pagination, DynTable, Filter, ActiveEntry, Pill, ActionCard.

Displays a list of all events.

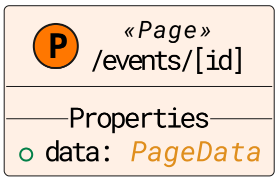
6.12.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; header: (Readable<TableHead<{ id?: string | undefined; } & { org_id?: string | undefined; distribution?: "0" | "1" | "2" | "3" | "4" | "5" | undefined; ... 16 more ...; event_creator_email?: string | undefined; } & { ...; } & { ...; }, undefined> & DynTableHead...`

Page data

snapshot: `Snapshot<any> | undefined`

6.13 /events/[id]



Uses DynCard, AddTagForm, EventGraph, Card, PillCollection.

This Page will display the event with a DynCard component. The header will be generated by the formHeaders depending on the mode.

The Galaxies and Tags will be displayed with a PillCollection. The EventGraph will be displayed with the EventGraph component. The Attributes are basically a DynTable

The update of the event will be handled by a form inside of this page, that is a wrapper around some DynCards. Therefore the "name" from from any inputted component will be used to calculate the final object we will send to the server.

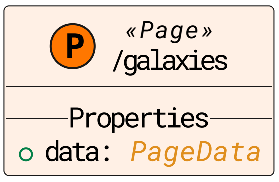
6.13.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; event: { id?: string | undefined; } & { org_id?: string | undefined; distribution?: "0" | "1" | "2" | "3" | "4" | "5" | undefined; info?: string | undefined; ... 15 more ...; event_creator_email?: string | undefined; } & { ...; } & { ...; }; }`

Page data containing the data of the event with the id in the url

6.14 /events/new

6.15 /galaxies



Uses DynTable.

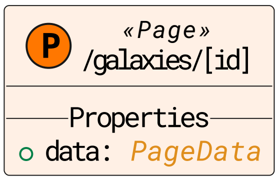
A list of all galaxies.

6.15.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; galaxies: { Galaxy?: { id?: string | undefined; uuid?: string | undefined; name?: string | undefined; type?: string | undefined; description?: string | undefined; version?: string | undefined; icon?: string | undefined; namespace?: string | undefined; kill_chai...`

The data that will be displayed on this page

6.16 /galaxies/[id]



Uses DynTable, DynCard.

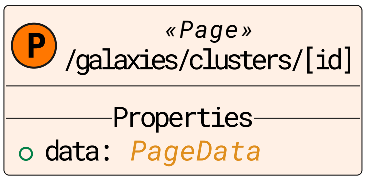
Information about a single galaxy, including a list of its clusters.

6.16.1 Props

data:
`{ [x: string]: any; [x: number]: any; [x: symbol]: any; galaxy: { Galaxy?: { id?: string | undefined; uuid?: string | undefined; name?: string | undefined; type?: string | undefined; description?: string | undefined; version?: string | undefined; icon?: string | undefined; namespace?: string | undefined; kill_chain_...`

Data that is displayed on this page.

6.17 /galaxies/clusters/[id]



Uses DynCard, DynTable.

Show all information about a single galaxy cluster, including its elements.

6.17.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; cardData: ({ id?: string | undefined; } & { uuid?: string | undefined; collection_uuid?: string | undefined; type?: string | undefined; ... 17 more ...; GalaxyElement?: { ...; }[] | undefined; } & { ...; }) | undefined; leftCardHeader: (Readable<...> | ... 1 mo...`

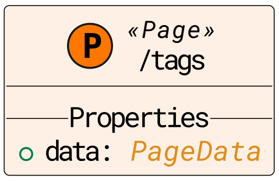
Data that is provided +page.ts on page load.

6.18 /settings

Uses Checkbox, Select, SettingsEntry.

Exposes various global settings of the application.

6.19 /tags



Uses Pagination, DynTable.

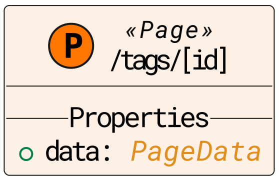
Displays a combined list of the tags of all events.

6.19.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; data: { Tag?: ({ id?: string | undefined; } & { name?: string | undefined; colour?: string | undefined; exportable?: boolean | undefined; ... 6 more ...; inherited?: number | undefined; })[] | undefined; }; tableData: ({ id?: string | undefined; } & { name?: st...`

Page data

6.20 /tags/[id]



Uses DynCard.

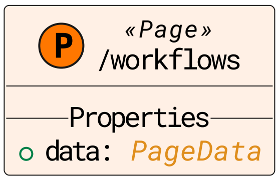
Shows information about a specific tag, specified by `id`.

6.20.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; user: { id?: string | undefined; } & { name?: string | undefined; colour?: string | undefined; exportable?: boolean | undefined; org_id?: string | undefined; ... 5 more ...; inherited?: number | undefined; }; header: (Readable<...> | ... 2 more ... | Readable<....`

Data that is displayed on this page.

6.21 /workflows



Uses DynTable.

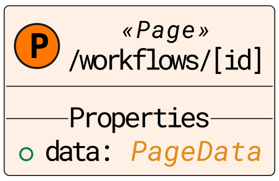
A list of all workflows.

6.21.1 Props

data: `{ [x: string]: any; }`

The data that will be displayed on this page.

6.22 /workflows/[id]



Uses DynCard, Flow.

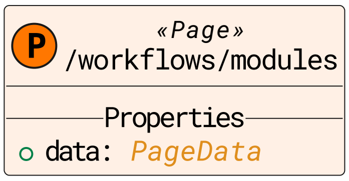
All the information about a specific workflow, including an interactive node-based diagram for visualization.

6.22.1 Props

data: `{ [x: string]: any; }`

The data that will be displayed on this page.

6.23 /workflows/modules

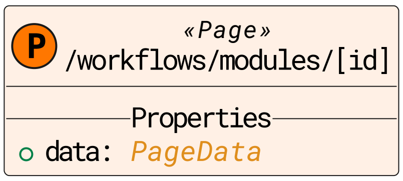


Uses DynTable.

6.23.1 Props

data:
`{ [x: string]: any; [x: number]: any; [x: symbol]: any; data: Module[]; tableData: Module[]; header: (Readable<TableHead<Module, Info__SvelteComponent_> & DynTableHeadExtent> | Readable<...>)[]; }`

6.24 /workflows/modules/[id]



Uses DynCard.

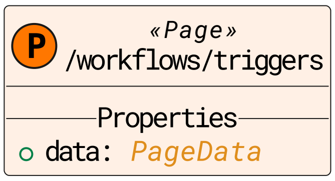
Displays information about a specific user, specified by `id`.

6.24.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; module: Module; infoHeader: (Readable<TableHead<Module, Info__SvelteComponent_> & Record<...>> | Readable<...>)[]; }`

Data that is displayed on this page.

6.25 /workflows/triggers



Uses DynTable.

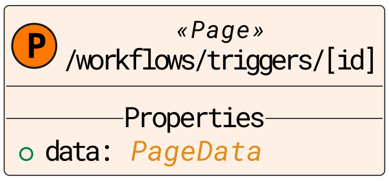
A list of all workflow triggers.

6.25.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; tableData: Trigger[]; header: (Readable<TableHead<Trigger, Info__SvelteComponent_> & DynTableHeadExtent> | Readable<...> | Readable<...> | Readable<...> | Readable<...>)[]; }`

The data that will be displayed on this page.

6.26 /workflows/triggers/[id]



Uses DynCard.

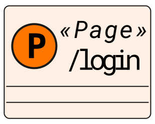
All the information about a specific workflow trigger, including an interactive node-based diagram for visualizing the workflow.

6.26.1 Props

data: `{ [x: string]: any; [x: number]: any; [x: symbol]: any; trigger: Trigger; infoHeader: (Readable<TableHead<Trigger, Info__SvelteComponent_> & Record<...>> | Readable<...> | Readable<...> | Readable<...>)[]; workflowHeader: (Readable<...> | ... 2 more ... | Readable<...>)[] | Readable<...>[]; }`

The data that will be displayed on this page.

6.27 /login

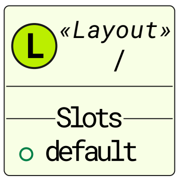


Uses Button, Input.

Provides a login flow via username and password. Stores the generated authentication token in `localStorage`, allowing the user to stay logged in after closing the page.

7. Layouts

7.1 /

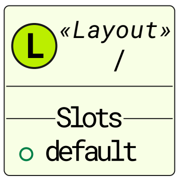


Root layout. Used to apply the theme and render the full application. The theme is based on css variables and tailwindcss. Elements using the proper tailwind classes will be themed automatically according to the current theme when placed in this layout.

7.1.1 Slots

default

7.2 /



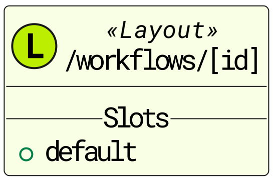
Uses Layout.

App Layout. Used for all routes besides `/login`. Contains the Layout component, in which each page's content is inserted into via the component's default slot.

7.2.1 Slots

default

7.3 /workflows/[id]

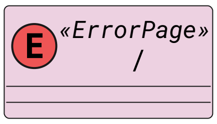


7.3.1 Slots

default

8. Error Pages

8.1 /



If any error occurs inside of a +page.ts load function, this page will be rendered. Handles 403 errors, by showing a link to the login page.

See: https://kit.svelte.dev/docs/errors