

MMISP Frontend Documentation

Design Phase

Table of contents

1. Intro	5
2. Design Overview	6
2.1 Components	6
2.2 API	6
2.3 Authentication	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 Boolean	19
5.8 LookupPill	20
5.9 Table	20
5.10 Td	21
5.11 Th	21
5.12 DynTable	22
5.13 Breadcrumbs	23
5.14 Checkbox	23
5.15 Select	24
5.16 Input	25
5.17 Card	26

5.18	CardRow	26
5.19	Button	27
5.20	AddTagForm	27
5.21	DynCard	28
5.22	IconCardRow	28
5.23	BaseNode	29
5.24	TriggerNode	30
5.25	ModuleNode	31
5.26	Flow	32
5.27	EventGraph	33
5.28	SideMenuDivider	33
5.29	SideMenuEntry	34
5.30	SideMenu	35
5.31	ActionBarEntry	36
5.32	CallbackEntry	36
5.33	HrefEntry	37
5.34	ActionBar	37
5.35	ToggleModeEntry	38
5.36	TopMenu	38
5.37	Layout	39
5.38	Pagination	39
5.39	SettingsEntry	40
6.	Pages	41
6.1	/admin/keys	41
6.2	/admin/keys/[id]	41
6.3	/admin/servers	41
6.4	/admin/servers/[id]	42
6.5	/admin/users	42
6.6	/admin/users/[id]	42
6.7	/attributes	42
6.8	/attributes/[id]	43
6.9	/events	43
6.10	/events/[id]	43
6.11	/events/new	44
6.12	/galaxies	44
6.13	/galaxies/[id]	44
6.14	/galaxies/clusters/[id]	45
6.15	/settings	45

6.16	/tags	45
6.17	/tags/[id]	46
6.18	/workflows/modules	46
6.19	/workflows/modules/[id]	46
6.20	/workflows/triggers	46
6.21	/workflows/triggers/[id]	47
6.22	/login	47
7.	Layouts	48
7.1	/	48
7.2	/	48
8.	Error Pages	49
8.1	/	49

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` which 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 Authentication

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 application 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 displayed 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 convenience 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({
```

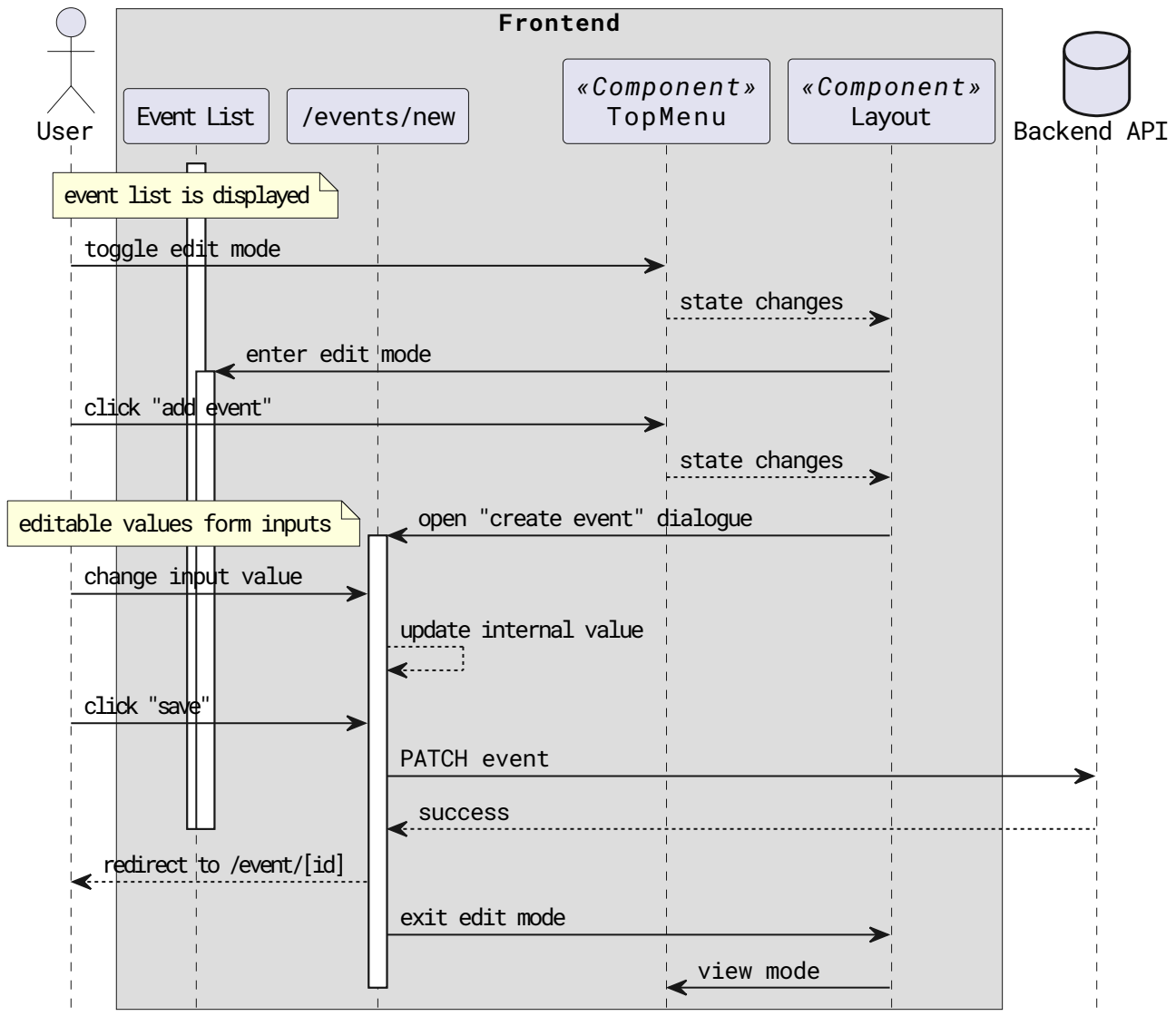


```
    icon: 'mdi:telescope',
    key: 'scope',
    label: 'Scope',
    value: (x) => ({ icon: 'mdi:telescope', text: x.scope ?? 'unknown' }),
    display: Pill
  }),
  col({
    icon: 'mdi:head-alert',
    key: 'overhead',
    label: 'Overhead',
    value: (x) => ({ value: x.trigger_overhead, options: THREAT_LEVEL_LOOKUP }),
    display: LookupPill
  })
];
```

4. Sequence Diagrams

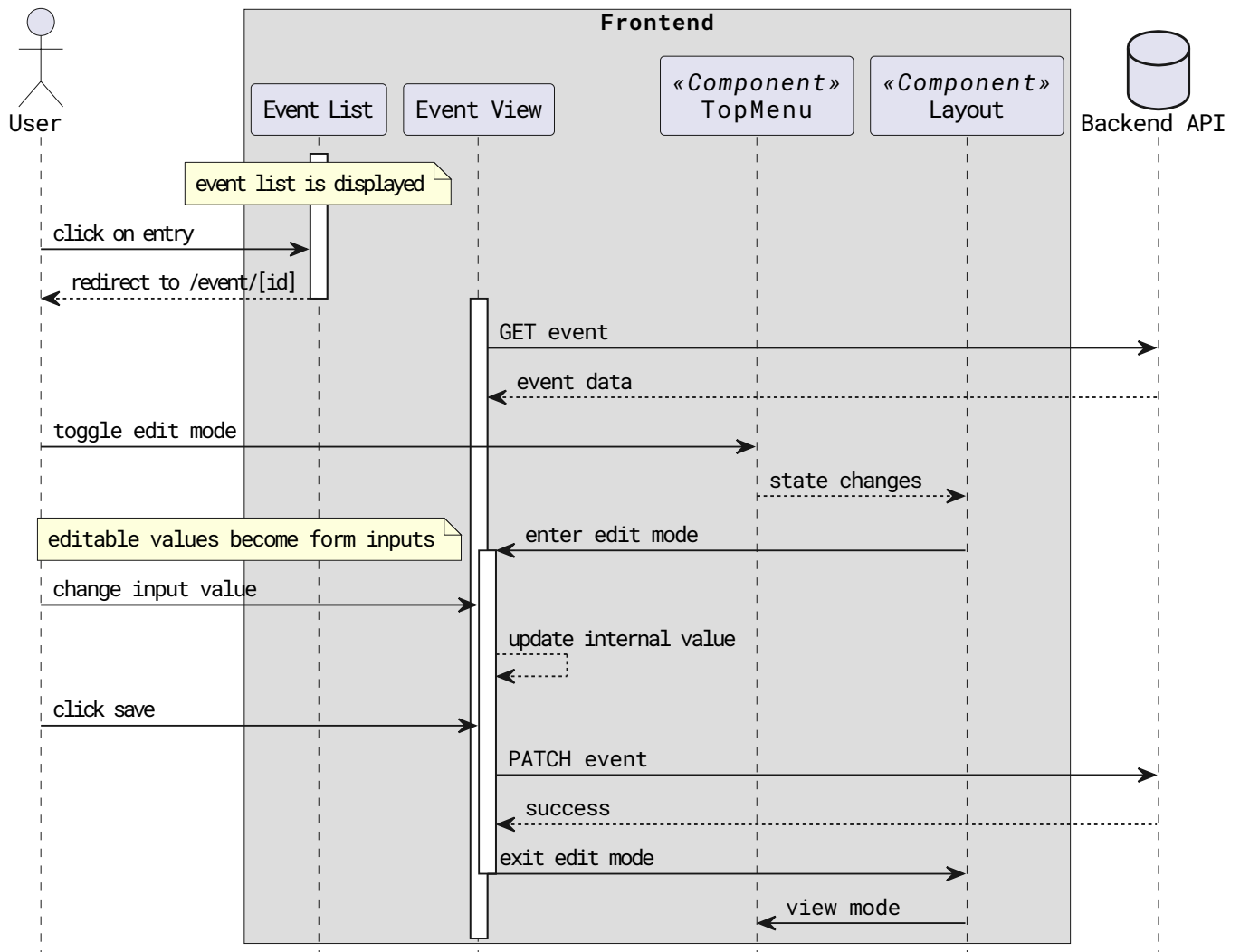
4.1 Add Event

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



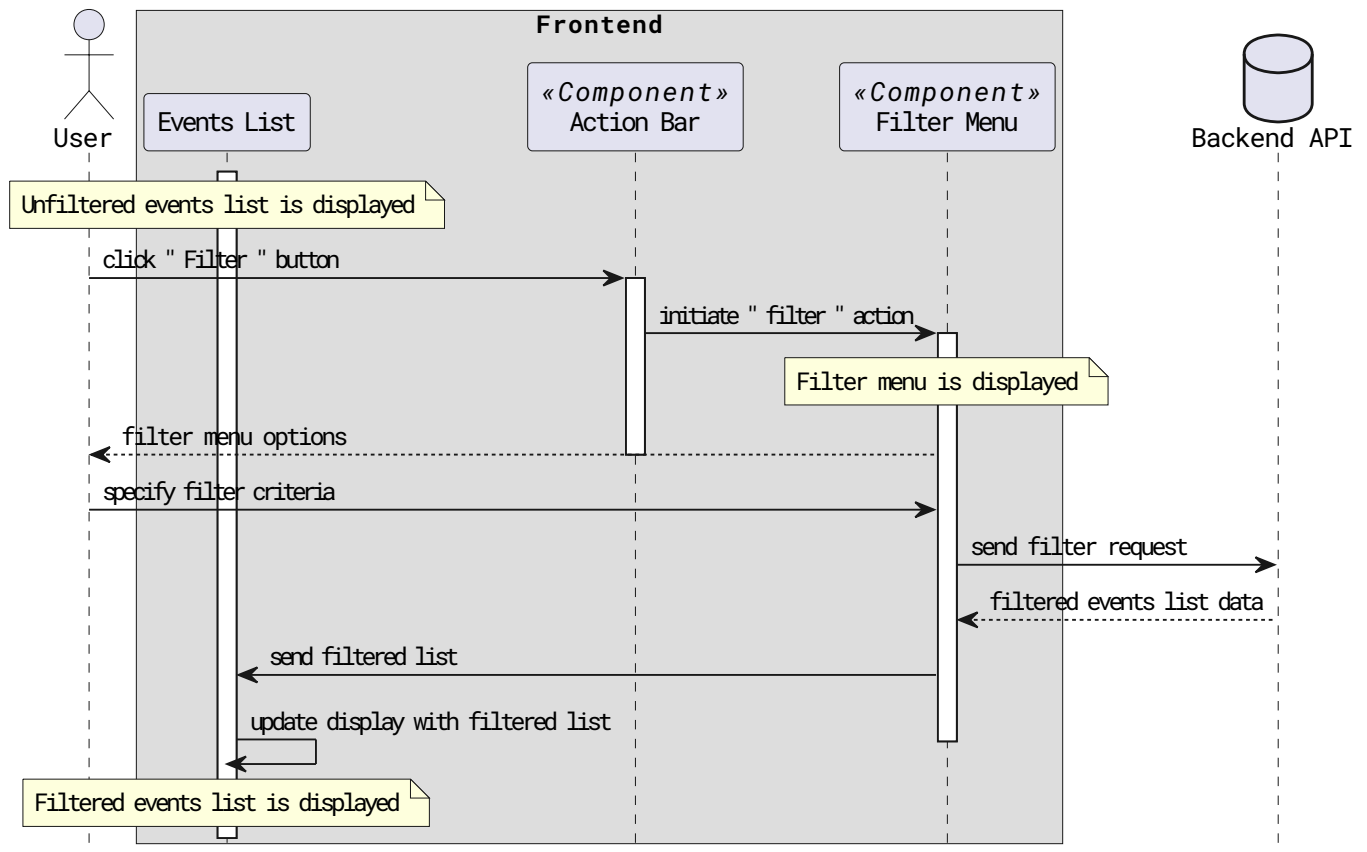
4.2 Edit Event

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



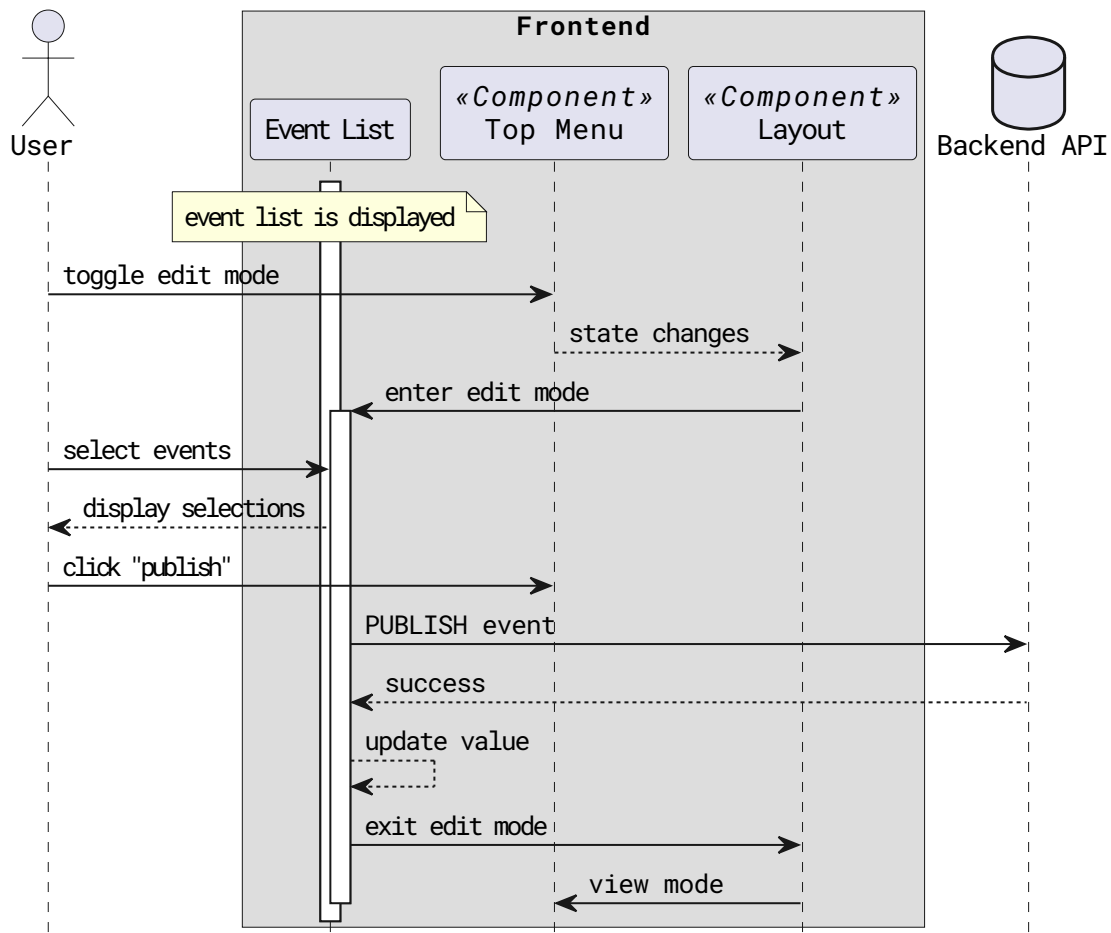
4.3 Filter Events

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



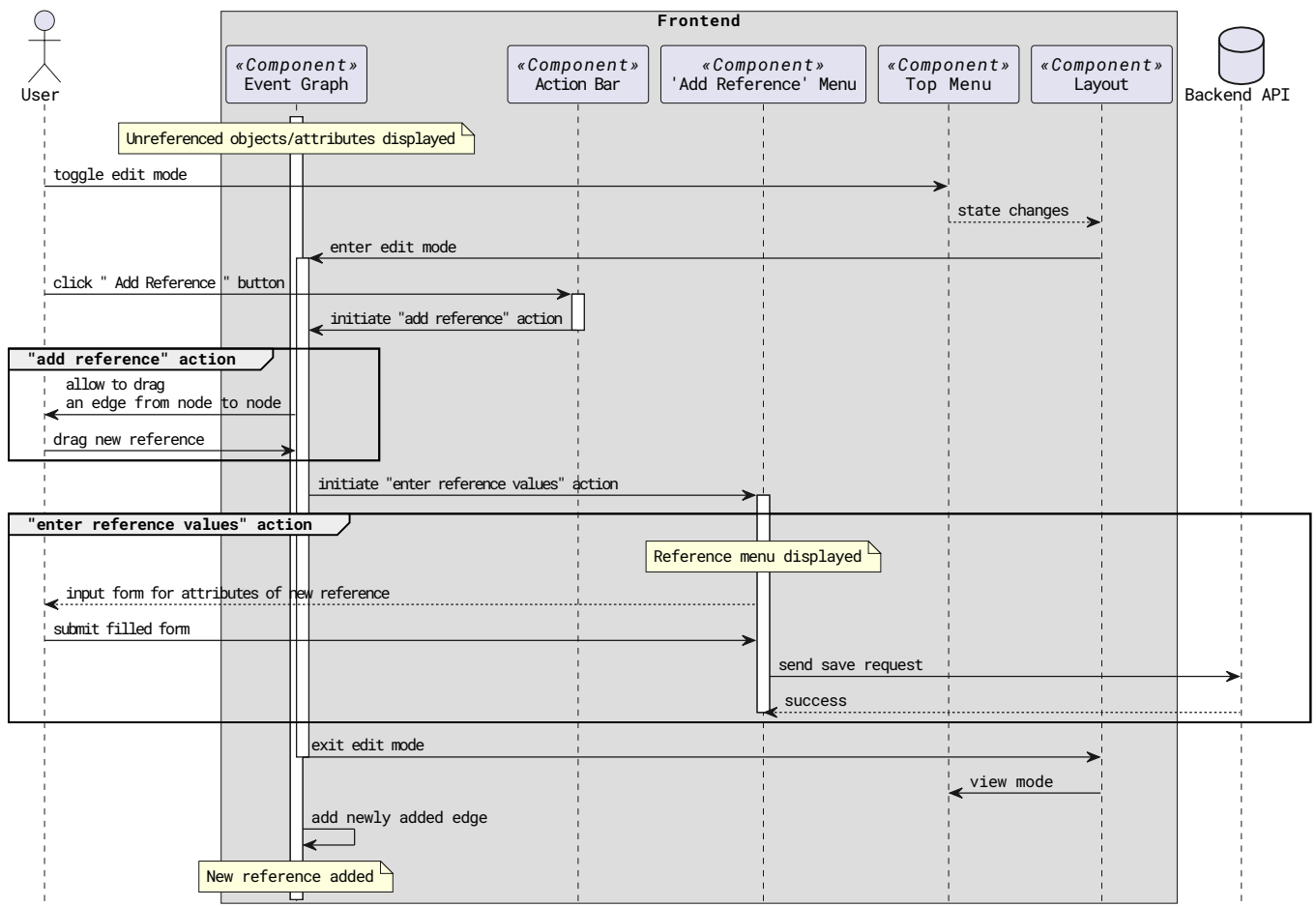
4.4 Publish Event

Publish an already existing event.



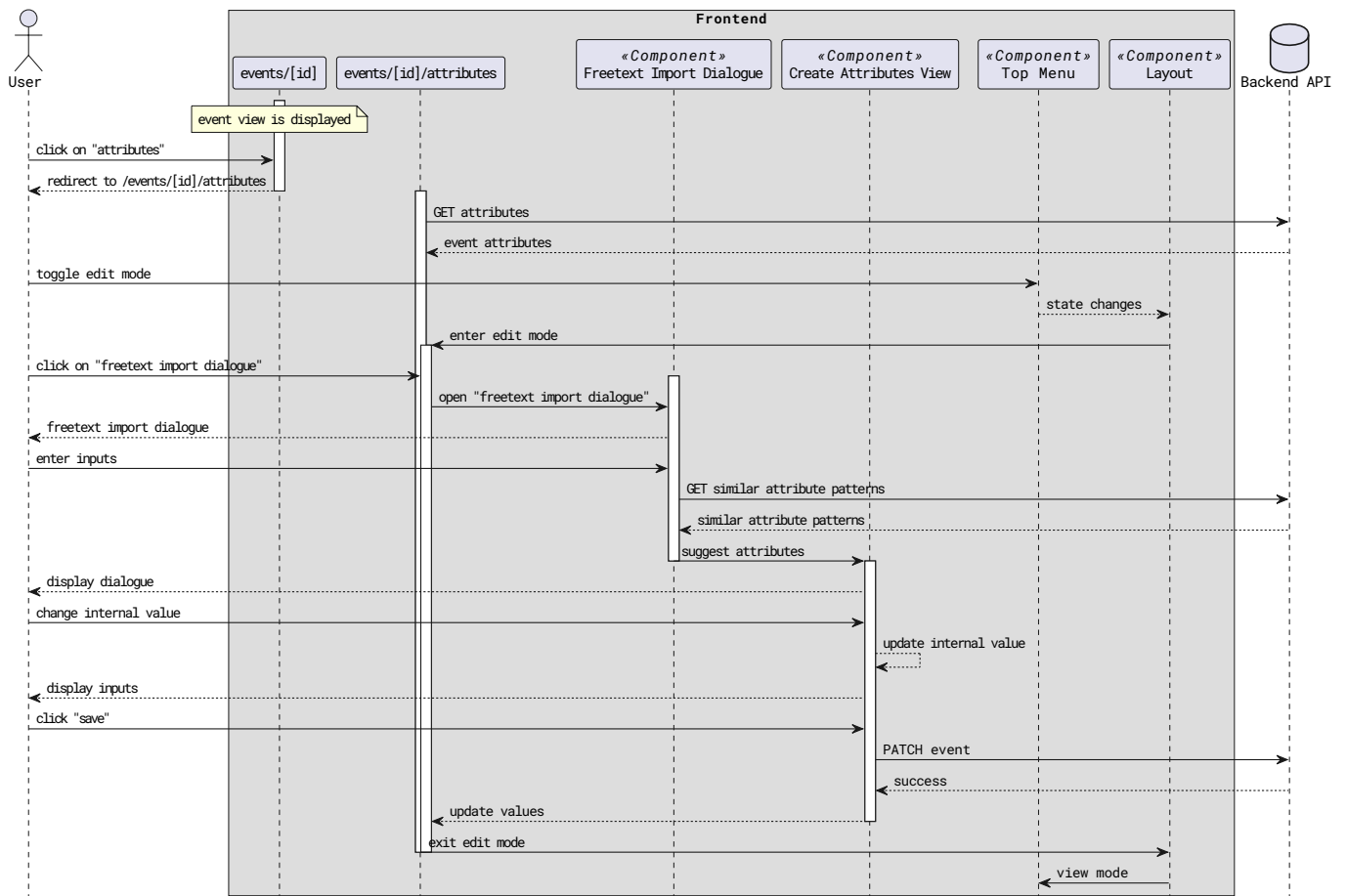
4.5 Add Reference

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.



5. Components

5.1 Info

«Component» Info
Properties
○ text?: string
○ class?: string
Slots
○ default

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

«Component» Pill
Properties
○ label?: string
○ text?: string
○ icon?: string
○ class?: string
○ style?: string
Slots
○ default

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.

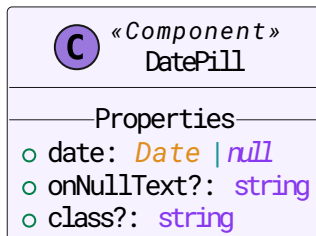
style: `string` | `undefined`

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

«Component»
RelativeDatePill
Properties
<ul style="list-style-type: none"> date: <i>Date</i> <i>null</i> onNullText?: <i>string</i>

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

«Component»
HrefPill
Properties
<ul style="list-style-type: none"> label?: <i>string</i> text?: <i>string</i> icon?: <i>string</i> href: <i>string</i>

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.

5.7.1 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.

5.8 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.

5.8.1 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.

5.9 Table

«Component»

Table

Slots

○ default

Creates an HTML `table` element with specific styling.

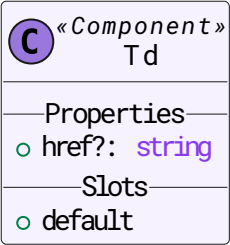
Slot:

The full table.

5.9.1 Slots

default

5.10 Td



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

Slot:

- The content of the `td`.

5.10.1 Props

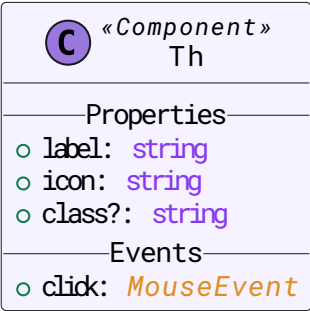
`href: string | undefined`

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

5.10.2 Slots

default

5.11 Th



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

Slot:

- The content of the `th`.

5.11.1 Props

`label: string`

The label of the column.

icon: `string`

The icon of the column.

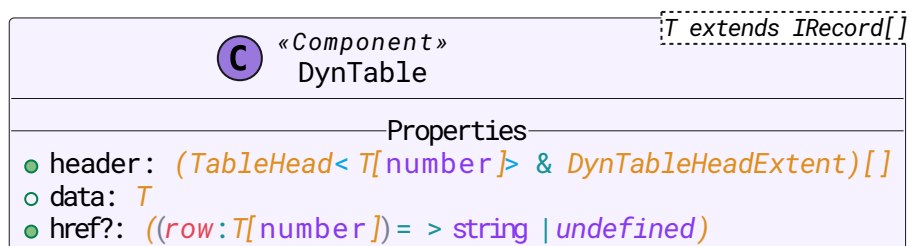
class: `string | undefined`

The class to be applied to the table head.

5.11.2 Events

click: `MouseEvent`

5.12 DynTable



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.

5.12.1 Props

Generics: `T extends IRecord[]`

header: `(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.

5.13 Breadcrumbs

«Component» Breadcrumbs
Properties
○ routes?: <i>Route[]</i>

Displays a `breadcrumb trail` with the given routes.

5.13.1 Props

routes: `Route[]` | `undefined`

The route that will be displayed in the breadcrumbs.

5.14 Checkbox

«Component» Checkbox
Properties
○ checked: <i>boolean</i>
○ name?: <i>string</i>
Events
○ change: <i>Event</i>

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.

5.14.1 Props

checked: `boolean`

Whether the checkbox is checked or not.

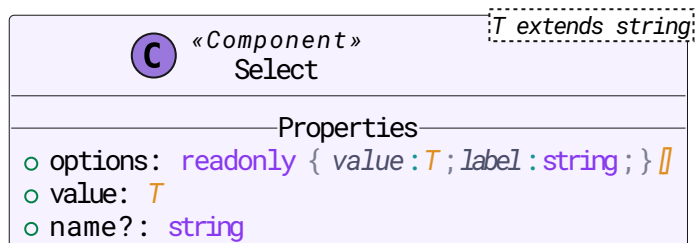
name: `string` | `undefined`

The form name of this checkbox.

5.14.2 Events

change: Event

5.15 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.

5.15.1 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.

5.16 Input

<div> <div>C</div> <div>«Component» Input</div> </div>
<div>Properties</div> <ul style="list-style-type: none"> placeholder?: string name?: string value?: string icon?: string type?: HTMLInputTypeAttribute class?: string
<div>Slots</div> <ul style="list-style-type: none"> icon suffix
<div>Events</div> <ul style="list-style-type: none"> blur: FocusEvent focus: FocusEvent value: CustomEvent< string>

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.

class: string | undefined

Additional classes to be applied.

5.16.2 Slots

icon

suffix

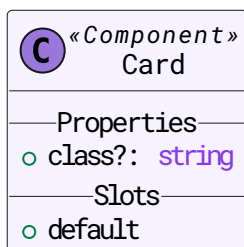
5.16.3 Events

blur: `FocusEvent`

focus: `FocusEvent`

value: `CustomEvent<string>`

5.17 Card



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

5.17.1 Props

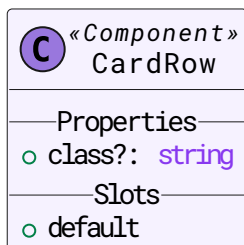
class: `string` | `undefined`

Additional classes to be applied to this component.

5.17.2 Slots

default

5.18 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.

5.18.1 Props

`class: string | undefined`

Additional classes to be applied to this component.

5.18.2 Slots

`default`

5.19 Button

<div><div>C</div><div>«Component» Button</div></div>
Properties
<div><div>○</div><div>class?: string</div></div>
Slots
<div><div>○</div><div>default</div></div>
Events
<div><div>○</div><div>click: MouseEvent</div></div>

A button with a slot for content.

5.19.1 Props

`class: string | undefined`

Additional classes to be applied to this component.

5.19.2 Slots

`default`

5.19.3 Events

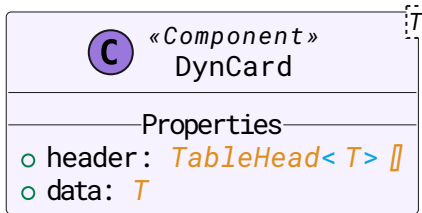
`click: MouseEvent`

5.20 AddTagForm

<div><div>C</div><div>«Component» AddTagForm</div></div>

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

5.21 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.

5.21.1 Props

Generics: **T**

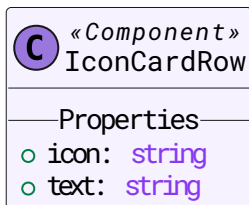
header: `TableHead<T>[]`

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

data: `T`

The data that will be displayed in the table.

5.22 IconCardRow



Uses [CardRow](#).

5.22.1 Props

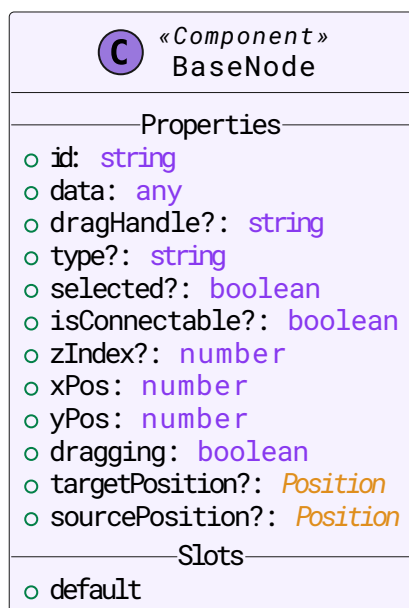
icon: `string`

The icon to be displayed in this row.

text: `string`

The text to be displayed in this row.

5.23 BaseNode



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

5.23.1 Props

id: string

data: any

dragHandle: string | undefined

type: string | undefined

selected: boolean | undefined

isConnectable: boolean | undefined

zIndex: number | undefined

xPos: number

yPos: number

dragging: boolean

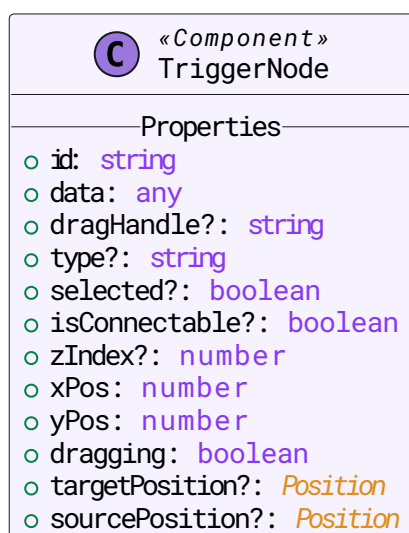
targetPosition: Position | undefined

sourcePosition: Position | undefined

5.23.2 Slots

default

5.24 TriggerNode



Uses [BaseNode](#).

A node representing a workflow trigger.

5.24.1 Props

id: string

data: any

dragHandle: string | undefined

type: string | undefined

selected: boolean | undefined

isConnectable: boolean | undefined

zIndex: number | undefined

xPos: number

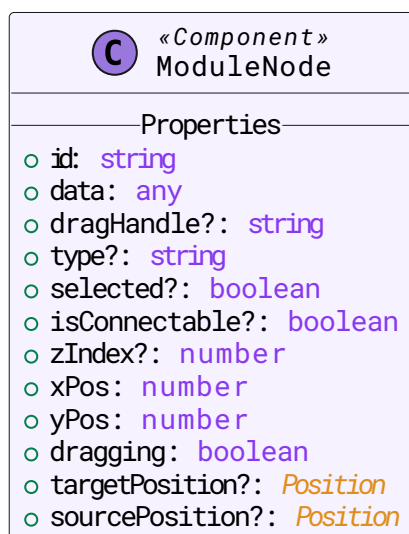
yPos: number

dragging: boolean

targetPosition: Position | undefined

sourcePosition: Position | undefined

5.25 ModuleNode



Uses [BaseNode](#).

A node representing a workflow module.

5.25.1 Props

- `id`: string
- `data`: any
- `dragHandle`: string | undefined
- `type`: string | undefined
- `selected`: boolean | undefined
- `isConnectable`: boolean | undefined
- `zIndex`: number | undefined
- `xPos`: number
- `yPos`: number
- `dragging`: boolean
- `targetPosition`: Position | undefined
- `sourcePosition`: Position | undefined

5.26 Flow

<div><div>C</div><div>«Component» Flow</div></div>	
Properties	
<ul style="list-style-type: none">nodes: Writable< Node[] >edges: Writable< Edge[] >snapGrid?: [number, number]	
Slots	
<ul style="list-style-type: none">default	
Events	
<ul style="list-style-type: none">nodeclick: CustomEvent< { node: Node< any, string undefined > ; event: MouseEvent TouchEvent ; } >	

Uses `TriggerNode`, `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.

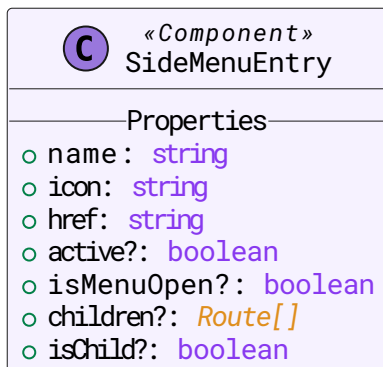
5.26.1 Props

- `nodes`: Writable<Node[]>
- Nodes that are rendered on the flow

5.28.1 Props

class: `string` | `undefined`

5.29 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.

5.29.1 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.

5.30 SideMenu

<div> <div>C</div> <div>«Component»</div> </div> <div>SideMenu</div>
<div>Properties</div> <ul style="list-style-type: none"> ○ <code>isOpen?: boolean</code> ○ <code>routes?: SideMenuRoute[]</code> ○ <code>activeRoute?: string null</code>
<div>Slots</div> <ul style="list-style-type: none"> ○ <code>logo</code> ○ <code>default</code>

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.

5.30.1 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.

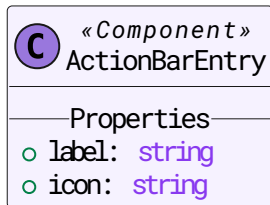
Should usually be the current URL provided by SvelteKit (`$page.url.href`).

5.30.2 Slots

logo

default

5.31 ActionBarEntry



Represents one of the entries of the [ActionBar](#).

5.31.1 Props

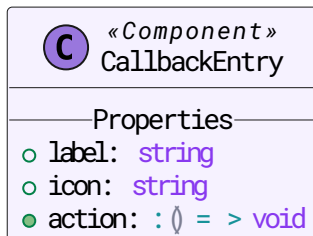
label: `string`

The label of this [ActionBar](#) entry.

icon: `string`

The icon of this [ActionBar](#) entry.

5.32 CallbackEntry



Uses [ActionBarEntry](#).

An [ActionBarEntry](#) with an `on:click` callback action associated with it.

5.32.1 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.

5.33 HrefEntry

C «Component» HrefEntry
Properties
<ul style="list-style-type: none"> ○ label: <code>string</code> ○ icon: <code>string</code> ○ action?: <code>string</code>

Uses `ActionBarEntry`.

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

5.33.1 Props

label: `string`

The label of this ActionBar entry.

icon: `string`

The icon of this ActionBar entry.

action: `string | undefined`

URL for hyperlink

5.34 ActionBar

C «Component» ActionBar
Properties
<ul style="list-style-type: none"> ○ entries: <code>ActionBarEntry[]</code>

Uses `CallbackEntry`, `HrefEntry`.

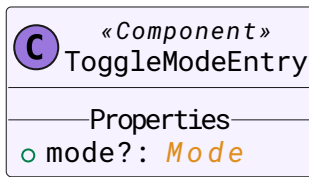
The action bar contains actions that can be performed when in edit mode.

5.34.1 Props

entries: `ActionBarEntry[]`

Actions that are displayed.

5.35 ToggleModeEntry



Uses [Checkbox](#).

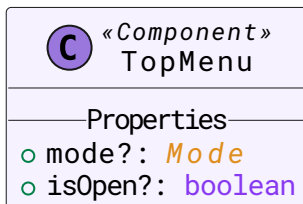
The [ActionBar](#) entry responsible for toggling modes.

5.35.1 Props

mode: [Mode](#) | undefined

The current mode of this Entry.

5.36 TopMenu



Uses [Input](#), [ActionBar](#), [ToggleModeEntry](#).

The top menu component.

The search bar and the [ActionBar](#) are located here.

5.36.1 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.

5.37 Layout

<div> <div>C</div> <div>«Component» Layout</div> </div>
<div>Properties</div> <ul style="list-style-type: none"> o routes: <i>SideMenuRoute[]</i> o currentRoute?: <i>Route[]</i>
<div>Slots</div> <ul style="list-style-type: none"> o sideMenu o default

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.

5.37.1 Props

routes: `SideMenuRoute[]`

The routes to be displayed in the side menu.

currentRoute: `Route[]` | `undefined`

The current route to be displayed in the [Breadcrumbs](#).

5.37.2 Slots

sideMenu

default

5.38 Pagination

<div> <div>C</div> <div>«Component» Pagination</div> </div>
<div>Properties</div> <ul style="list-style-type: none"> o page: <i>number</i> o length?: <i>number</i>

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

5.38.1 Props

page: `number`

The current page.

length: number | undefined

The total number of pages.

5.39 SettingsEntry

C «Component» SettingsEntry
Properties
○ label: string
Slots
○ default

Represents a single settings entry on `/settings`.

Slot:

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

5.39.1 Props

label: string

The label of the settings entry.

5.39.2 Slots

default

6. Pages

6.1 /admin/keys

P	«Page» /admin/keys
Properties	
o data:	PageData

Uses `DynTable`.

6.1.1 Props

data:

```
{ [x: string]: any; [x: number]: any; [x: symbol]: any; header: ((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?: { ...; } | undefin...
```

6.2 /admin/keys/[id]

P	«Page» /admin/keys/[id]

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

6.3 /admin/servers

P	«Page» /admin/servers
Properties	
o data:	PageData

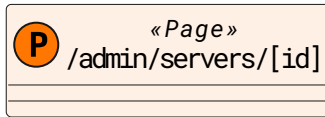
Uses `DynTable`.

Displays a list of all remote servers of the instance.

6.3.1 Props

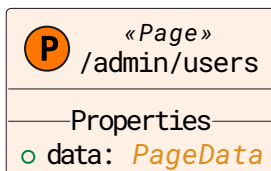
```
data: { [x: string]: any; [x: number]: any; [x: symbol]: any; data: { Server?: ({ id?: string | undefined; } & { name?: string | undefined; url?: string | undefined; authkey?: string | undefined; org_id?: string | undefined; ... 20 more ...; cache_timestamp?: boolean | undefined; }) | undefined;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

6.4 /admin/servers/[id]



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

6.5 /admin/users



Uses `DynTable`.

Displays a list of all users of the instance.

6.5.1 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?: { ...; } |...
```

6.6 /admin/users/[id]

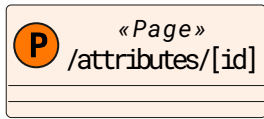


Displays information about a specific user, specified by `id`.

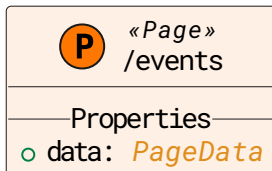
6.7 /attributes



6.8 /attributes/[id]



6.9 /events



Uses [Pagination](#), [DynTable](#).

Displays a list of all events.

6.9.1 Props

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

Page data

6.10 /events/[id]



Uses [DynCard](#), [AddTagForm](#), [EventGraph](#), [DynTable](#), [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.

6.10.1 Props

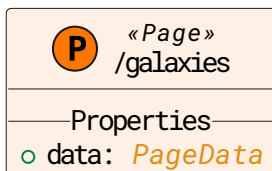
```
data: { [x: string]: any; [x: number]: any; [x: symbol]: any; event: ({ id?: string | undefined; } & { org_id?: string | undefined; distribution?: "0" | "1" | "2" | "3" | "4" | "5" | undefined; info?: string | undefined; ... 15 more ...; event_creator_email?: string | undefined; } & { ...; }) | undefined;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

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

6.11 /events/new



6.12 /galaxies



Uses `DynTable`.

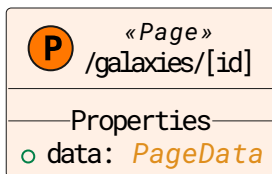
A list of all galaxies.

6.12.1 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

6.13 /galaxies/[id]



Uses `DynCard`, `DynTable`.

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

6.13.1 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.

6.14 /galaxies/clusters/[id]

P	«Page» /galaxies/clusters/[id]
Properties	
o data:	PageData

Uses [Boolean](#), [Card](#), [CardRow](#), [Info](#), [HrefPill](#), [LookupPill](#), [PillCollection](#), [DynTable](#).

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

6.14.1 Props

```
data: { [x: string]: any; [x: number]: any; [x: symbol]: any; galaxyCluster: { GalaxyCluster?: ({ id?: string | undefined; } & { uuid?: string | undefined; collection_uuid?: string | undefined; type?: string | undefined; ... 17 more ...; GalaxyElement?: { ...; }[] | undefined; } & { ...; }) | undefined; }; tableData: { .....
```

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

6.15 /settings

P	«Page» /settings

Uses [Checkbox](#), [Select](#), [SettingsEntry](#).

Exposes various global settings of the application.

6.16 /tags

P	«Page» /tags
Properties	
o data:	PageData

Uses [Pagination](#), [DynTable](#).

Displays a combined list of the tags of all events.

6.16.1 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...
```

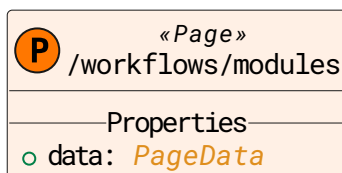
Page data

6.17 /tags/[id]



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

6.18 /workflows/modules

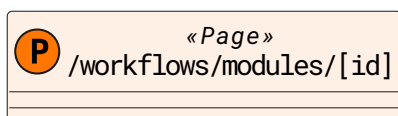


Uses [DynTable](#).

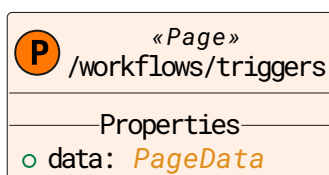
6.18.1 Props

```
data: { [x: string]: any; [x: number]: any; [x: symbol]: any; data: Record<string, never> | { AuthKey?: { id?: string | undefined; uuid?: string | undefined; authkey_start?: string | undefined; ... 7 more ...; last_used?: string | ... 1 more ... | undefined; } | undefined; User?: { ...; } | undefined; }[] | ... 41 more .....
```

6.19 /workflows/modules/[id]



6.20 /workflows/triggers



Uses [DynTable](#).

A list of all workflow triggers.

6.20.1 Props

data: { [x: string]: any; [x: number]: any; [x: symbol]: any; tableData: Record<string, never> | { AuthKey?: { id?: string | undefined; uuid?: string | undefined; authkey_start?: string | undefined; ... 7 more ...; last_used?: string | ... 1 more ... | undefined; } | undefined; User?: { ...; } | undefined; }[] | ... 41 mo...

The data that will be displayed on this page.

6.21 /workflows/triggers/[id]

P «Page» /workflows/triggers/[id]
Properties
o data: <i>PageData</i>

Uses [DynCard](#), [Flow](#).

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

6.21.1 Props

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

The data that will be displayed on this page.

6.22 /login

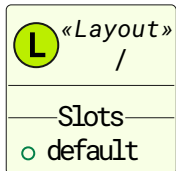
P «Page» /login

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 /

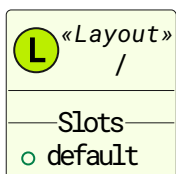


Root layout. Used to apply the theme and render the full application. The theme is based on css variables and [tailwindcss](#). Elements using the proper tailwind classes will be themed automatically according to the current theme when placed in this layout.

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

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>