Web Advanced

Svelte

Component based development

SAXION HOGESCHOOL

Last week

Svelte introduction

- Components
- Composition
- Events
- Templating

Components

Importing components

- A component is just a *.svelte* file. A component can load and render multiple components.

```
<script>
import Counter from './lib/Counter.svelte'
</script>

<main>
<Counter />
</main>
```

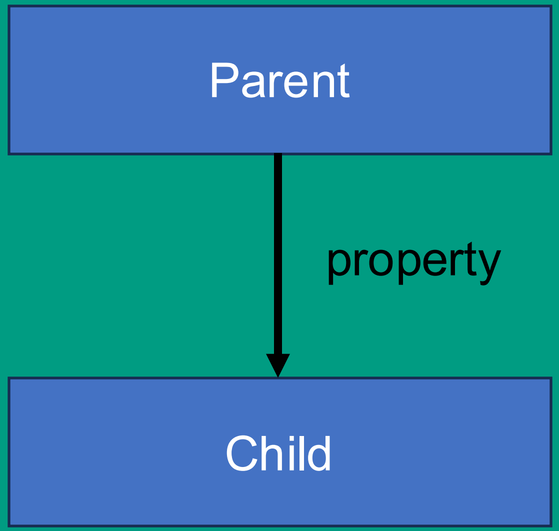
Component communication

Communication

One of the challenges in working with components, is how to tackle communication between these components. Svelte offers us:

- Parent to child communication: using properties
- Child to parent communication: using events
- Parent to indirect children: using context
  (week 5)
- One object that stores all variables, accessible by each component: using stores
  (next week)

Component communication



Parent to child communication

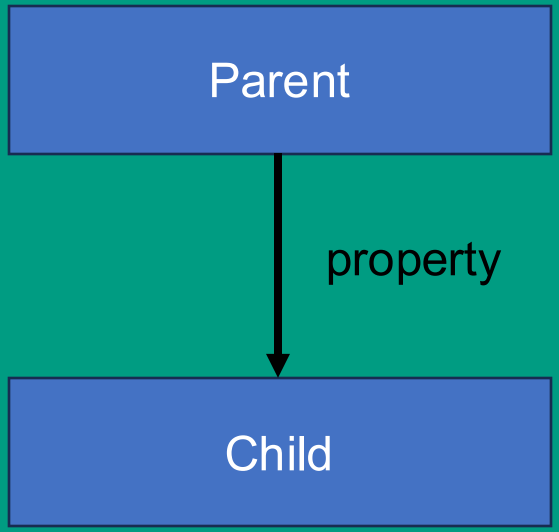
Each component can define a couple of properties. These properties can be set by it's parent component.

Parameters passed to other components are called properties or props.

```
const { items } = $props();
```

Properties can be destructured from the `$props()` function.

Component communication



Parent to child communication

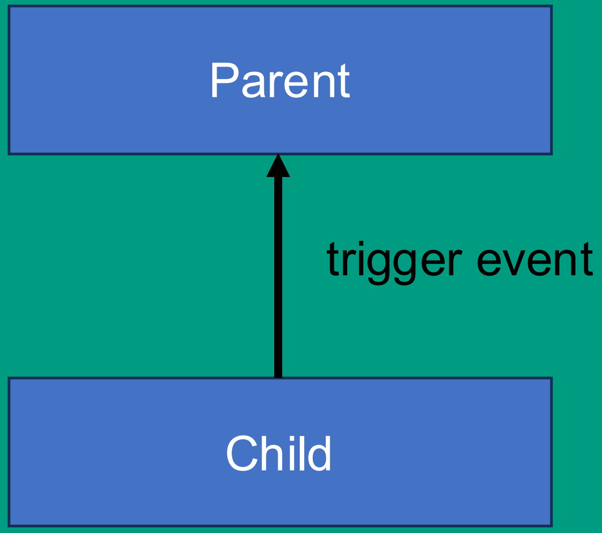
The parent component can then use the new component and pass values to the component by setting the properties.

```
<List items={items} />
```

If the variable name and property have the same name in this case `items` we can use a short hand notation.

```
<List {items} />
```

Component communication



Child to parent communication

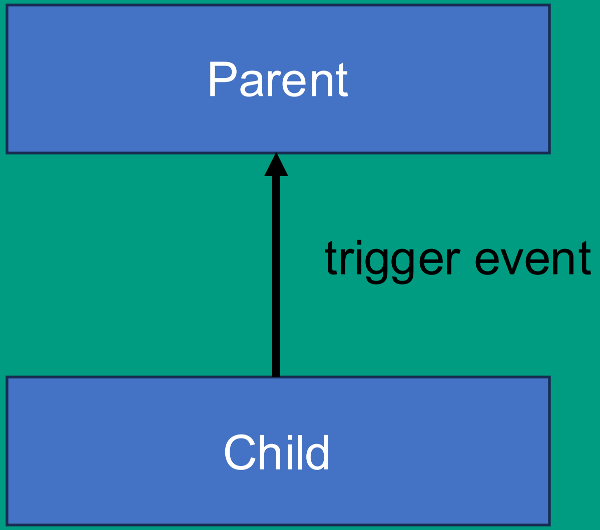
A child can communicate with its parent by notifying its parent using an event.

```
<script>
    const { items, addItem } = $props();
</script>

<button onclick={() => addItem("new item")}>Add</button>
```

The event is the `onclick` on the button. As with all events it requires an event handler. In this case it is a function that calls the function that is passed in through the `addItem` property.

Component communication



Child to parent communication

The parent component must pass the function as a property.

```
<List {items} {addItem} />
```

The parent component declares the function and has it perform the required actions.
In this case a new item is pushed onto the array of items.

```
const items = $state(['item 1']);


const addItem = newItem => items.push(newItem);
```

SAXION HOGESCHOOL

Demo

Component based shopping list

- We are going to update the shopping list application by refactoring it into multiple components

- We need components for:
  - Adding new ingredients
  - List items

- We want to add the possibility to add a quantity to the ingredients and change the value as well.

Routing

Create browser pages

- Svelte is a component framework
- We use it to build components
- It does not natively support pages
- Pages are needed to allow the user to navigate directly to content.
- As we saw last week, a page can be a component too.

Routing

Routing

Routing in Single Page Applications

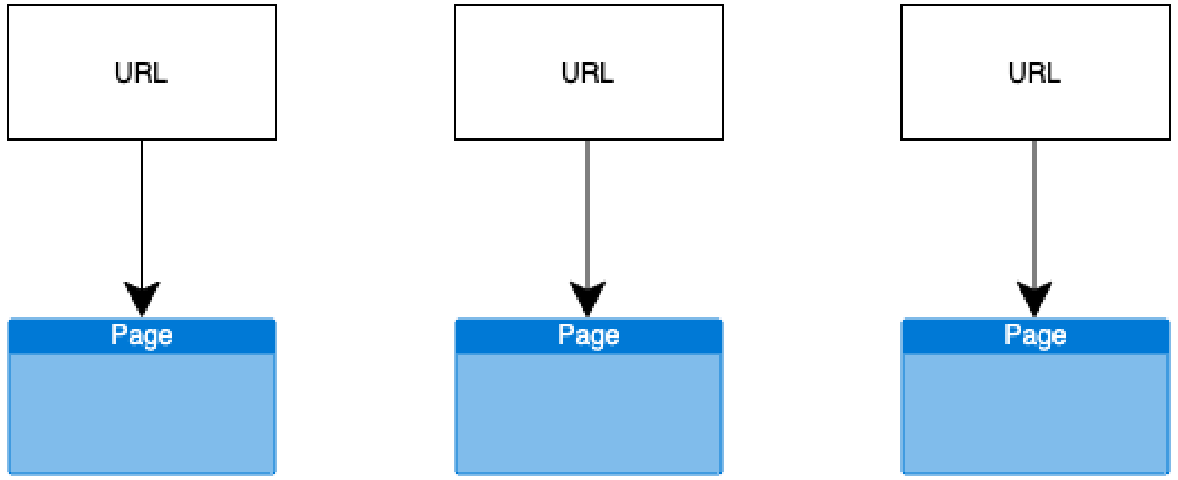
- Routing allows you to navigate between different views or sections of the app <u>without</u> triggering a full page reload.

- Svelte does not have a built-in mechanism for this, therefore we need to rely on an external library, called **page.js** (https://github.com/visionmedia/page.js)

Routing

What is routing

Originally each url pointed to a single resource

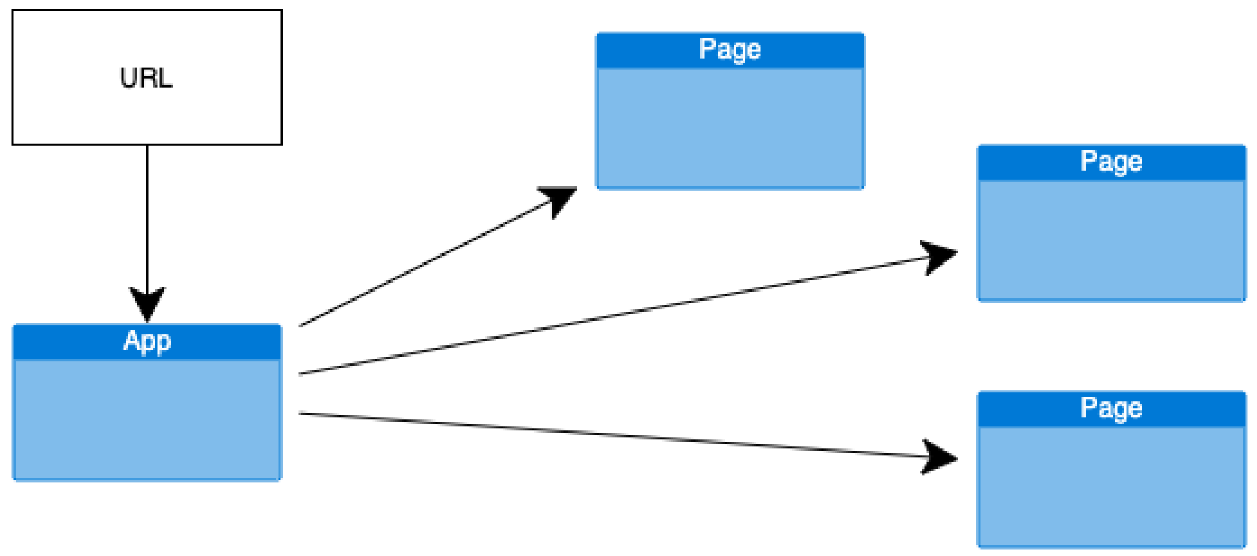


```
https://localhost:5000/index.html
```

Routing

What is routing

A single page app is served from a single resource. The app is then responsible for showing the right content.



SAXION HOGESCHOOL

Routing

How does the router work?

1. We need to define our routes (the url's and the corresponding components)
2. When you are using a `<a href="/about">` tag, instead of directly following the url, the router will intercept the click event and check if the url is part of the stored routes.
3. When the url is stored in the routing table, the page won't follow the link, but the router will update the contents of your page dynamically and change the url using the History API.

Routing

Page.js 1 - Import router and pages

- For each page create a new component

- Import the page library

```
import router from "page"
```

- Import the different pages in `App.svelte`

```
import Home from './components/Home.svelte'
import About from './components/About.svelte'
```

Routing

Page.js 2 - Setup routing

Create a route for each page and start the router.

```
let Page = $state();

router('/', () => { Page = Home; });
router('/about', () => {Page = About; });

router.start();
```

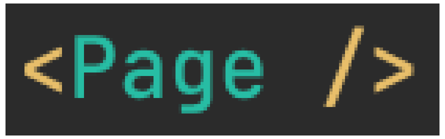
The assignment template has more examples

Routing

Page.js 3 - Render the component

Svelte now has a `page` variable that contains the requested page component.

All we have to do is render the component.



```
<Page />
```

Check the assignment template for more elaborate examples including the use of router context.

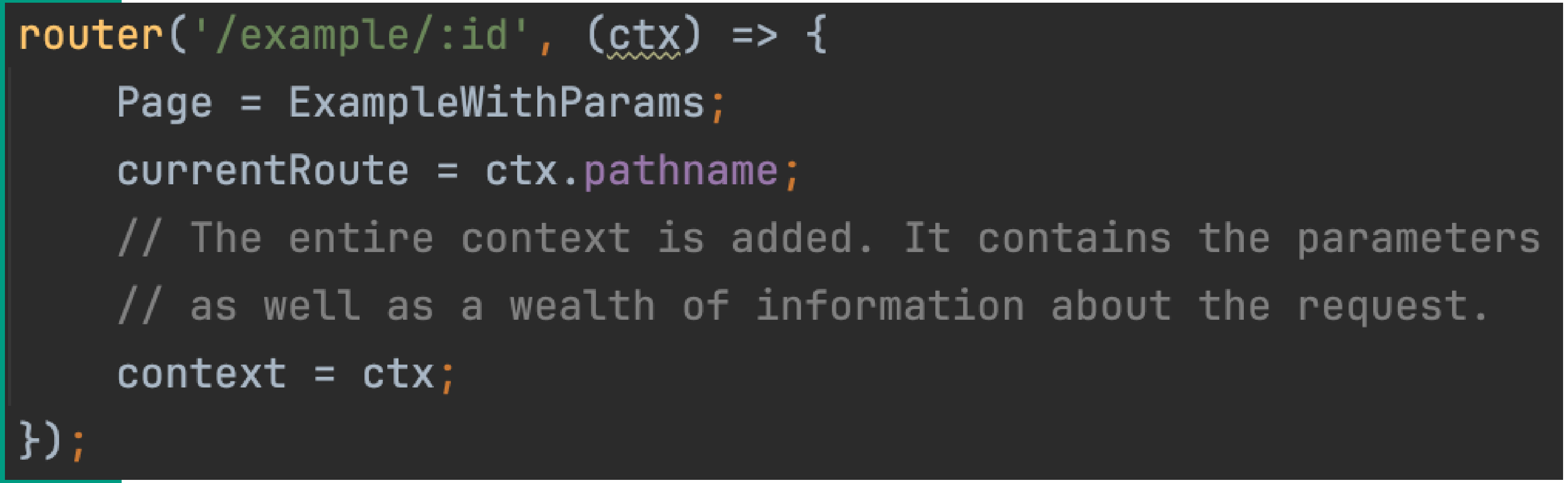
Routing

Page.js 4 - Parameters

Sometimes we want to request a page based on a parameter

http://localhost:5173/item/1

We need to tell the router how to handle those parameters.



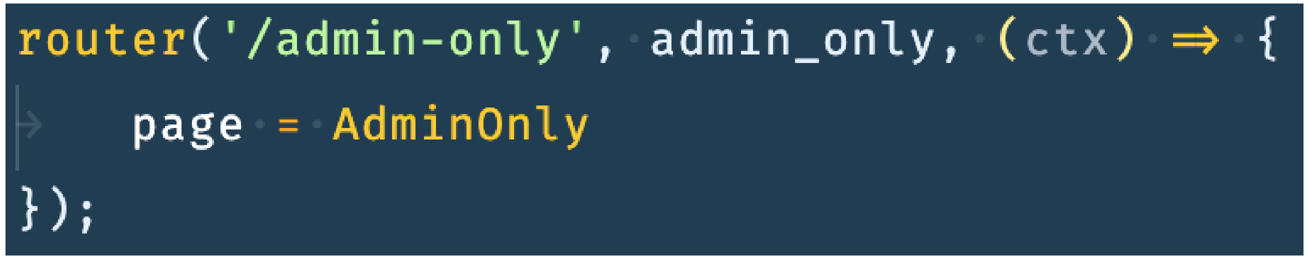
```
router('/example/:id', (ctx) => {
    Page = ExampleWithParams;
    currentRoute = ctx.pathname;
    // The entire context is added. It contains the parameters
    // as well as a wealth of information about the request.
    context = ctx;
});
```

SAXION HOGESCHOOL

Routing

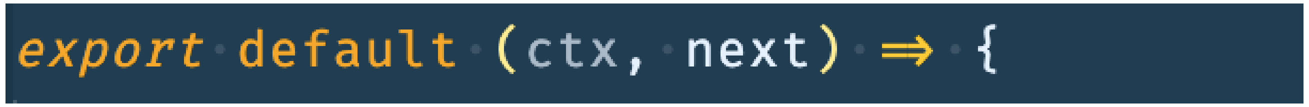
Page.js 5 Middleware

Just like with Express, we want to check if a user has access. We can do that in a similar way with middleware.



```
router('/admin-only', admin_only, (ctx) => {
    page = AdminOnly
});
```

`admin_only` is a middleware function that accepts the context and next. Just like express has `(req, res, next)`



```
export default (ctx, next) => {
```

Questions