

# **FRONTEND DEVELOPMENT**

## **WINTERSEMESTER 2021**

# Traditional

Every request for new information gives you a new version of the whole page.

# SPA

You request just the pieces you need.

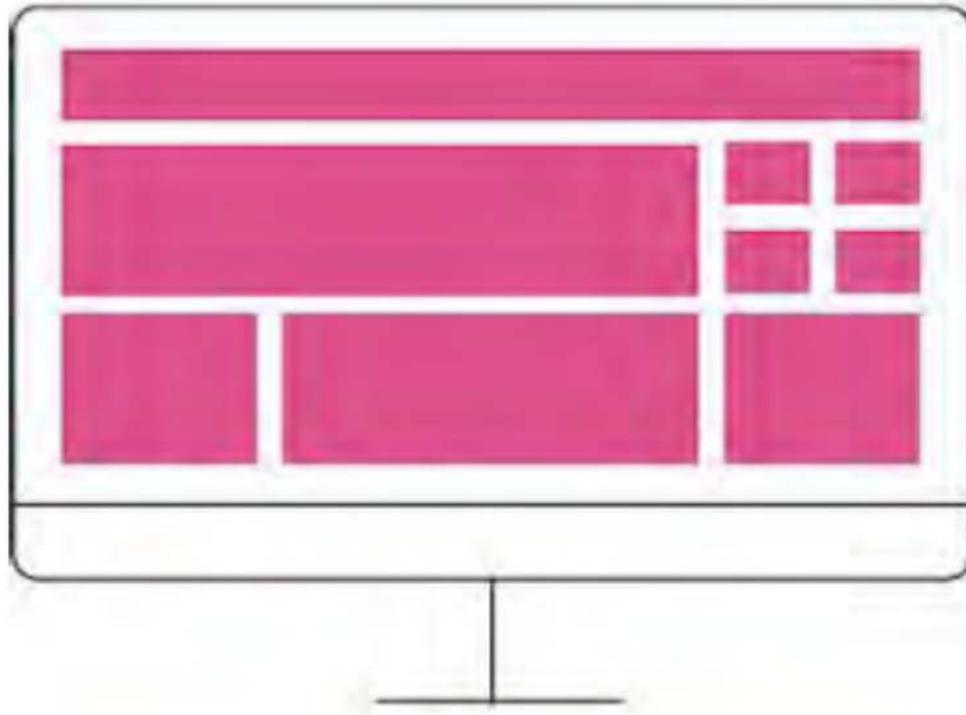
## SPA VS TRADITIONAL APPS



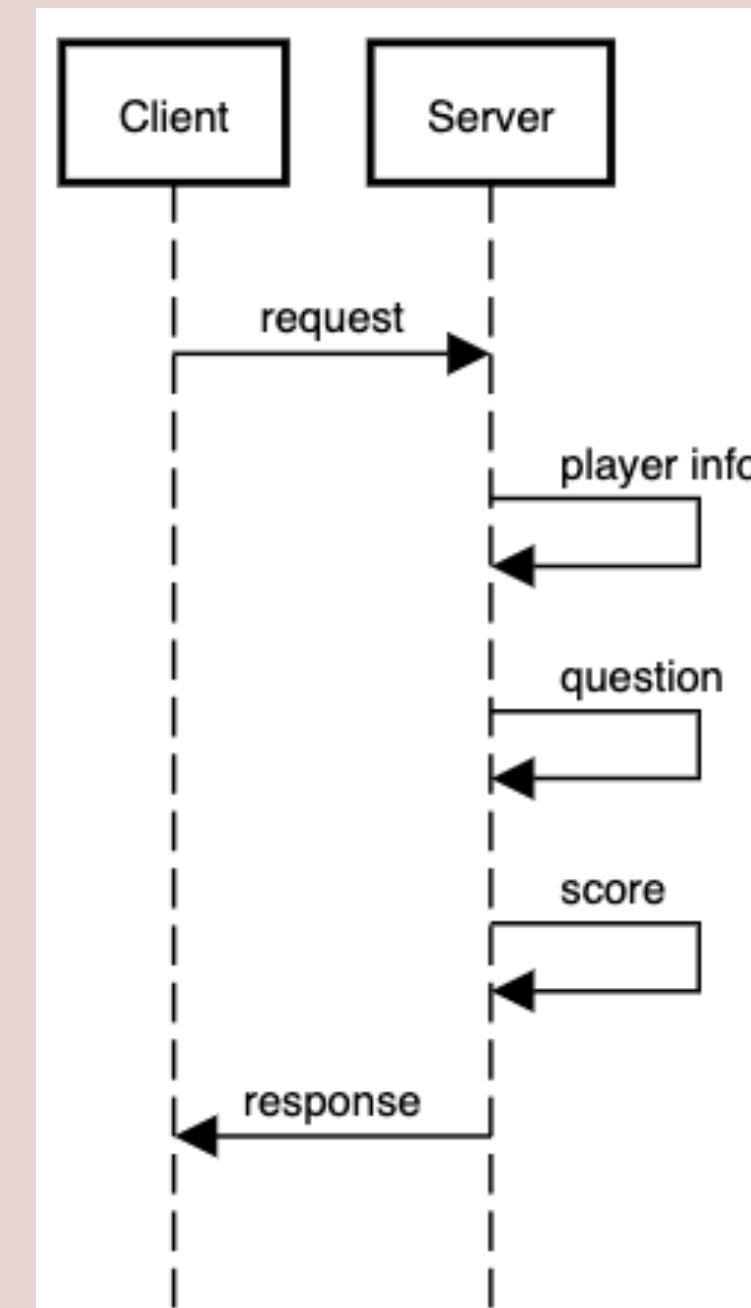
# TRADITIONAL WEB APPLICATIONS

## Traditional

Every request for new information gives you a new version of the whole page.



# TRADITIONAL WEB APPLICATIONS



# TRADITIONAL WEB APPLICATIONS

- » client /quiz
- » server gets request
  - » fetches player info
  - » fetches question
  - » fetches score
- » when all data fetched returns rendered HTML

# TRADITIONAL WEB APPLICATIONS

» Video

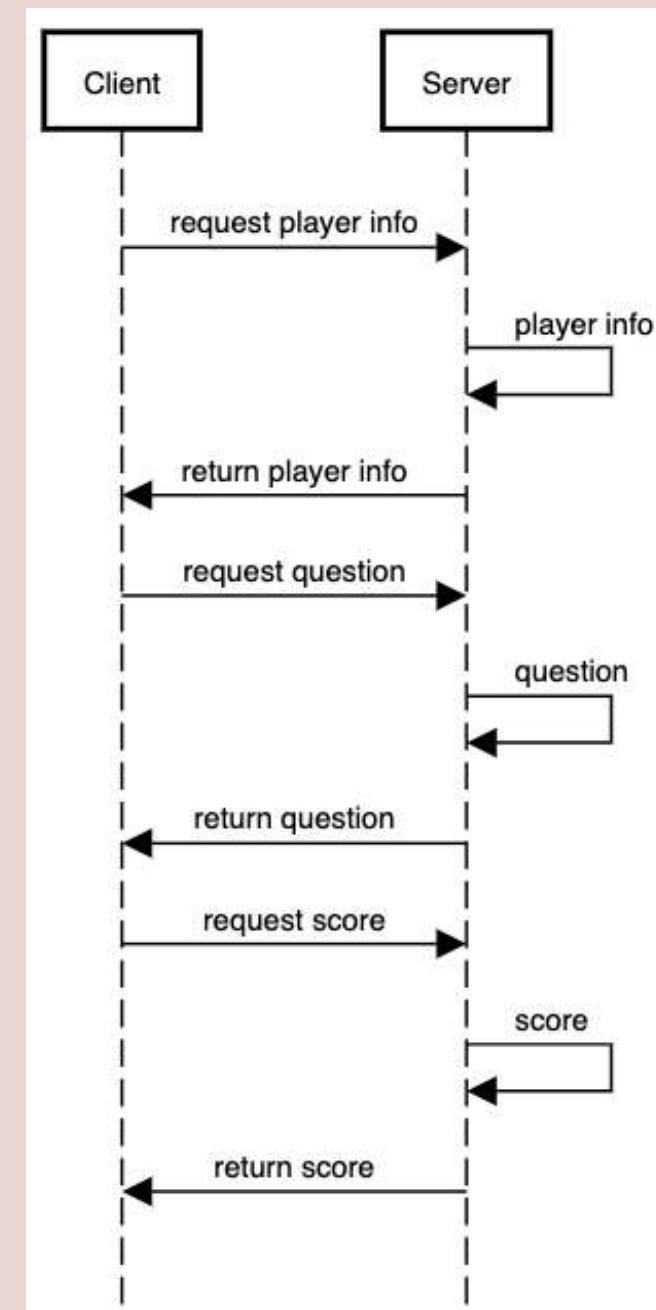
# SINGLE PAGE APPLICATIONS



# SINGLE PAGE APPLICATIONS

“A single-page application (SPA) is a web application or web site that fits on a single web page with the goal of providing a user experience similar to that of a desktop application.”

# SINGLE PAGE APPLICATIONS



# SINGLE PAGE APPLICATIONS

- » client requests /quiz
- » client requests in parallel
  - » player-info
  - » question
  - » score
- » when one of these requests return
  - » client displays data immediately

# SINGLE PAGE APPLICATIONS

» video

# SINGLE PAGE APPLICATIONS

## ADVANTAGES<sup>2</sup>

- » No redundant Queries to Server
- » Fast and Responsive Front-end Built
- » Enhanced User Experiences

<sup>2</sup> <https://www.bloomreach.com/en/blog/2018/07/what-is-a-single-page-application.html#whatssingle-page-application>

# SINGLE PAGE APPLICATIONS

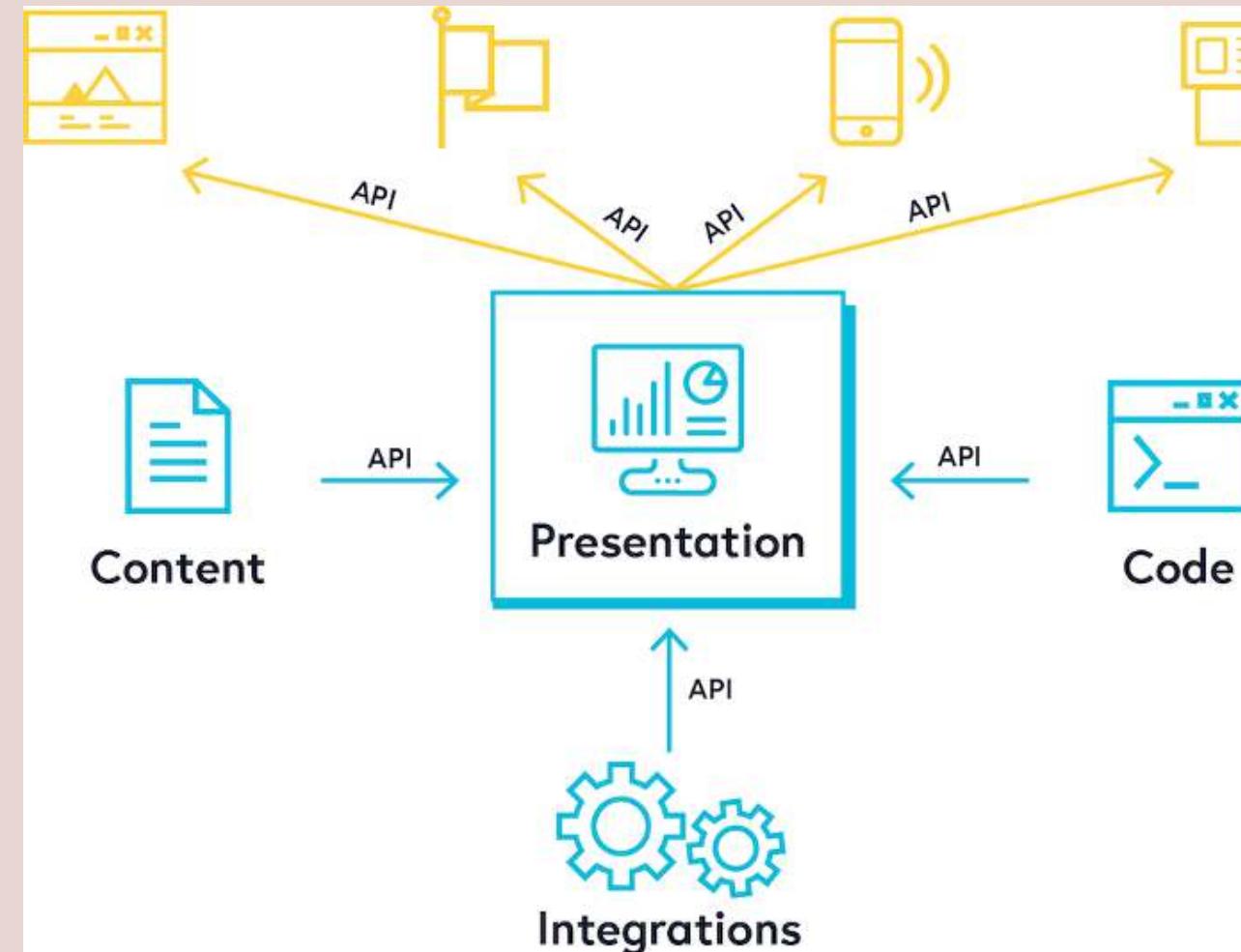
## NO REDUNDANT QUERIES<sup>2</sup>

- » client requests data which he needs
- » no need for a full rerender of app
- » removes unnecessary/expensive DB queries

<sup>2</sup> <https://www.bloomreach.com/en/blog/2018/07/what-is-a-single-page-application.html#whatssingle-page-application>

# SINGLE PAGE APPLICATIONS FAST AND RESPONSIVE FRONT- END BUILT

# SINGLE PAGE APPLICATIONS FAST AND RESPONSIVE FRONT-END BUILT<sup>2</sup>



<sup>2</sup> <https://www.bloomreach.com/en/blog/2018/07/what-is-a-single-page-application.html#whatssingle-page-application>

# SINGLE PAGE APPLICATIONS FAST AND RESPONSIVE FRONT-END BUILT<sup>2</sup>

- » many clients can be built with same backend
- » one client could be composed of different backends
  - » blog served from own backend
  - » comments served from third party service (eg. facebook/disqus)

<sup>2</sup> <https://www.bloomreach.com/en/blog/2018/07/what-is-a-single-page-application.html#whatssingle-page-application>

# SINGLE PAGE APPLICATIONS

## ENHANCED USER EXPERIENCE

- » no full page refresh required
- » dynamic content loading possible
- » faster page transitions
- » HTML/CSS already loaded

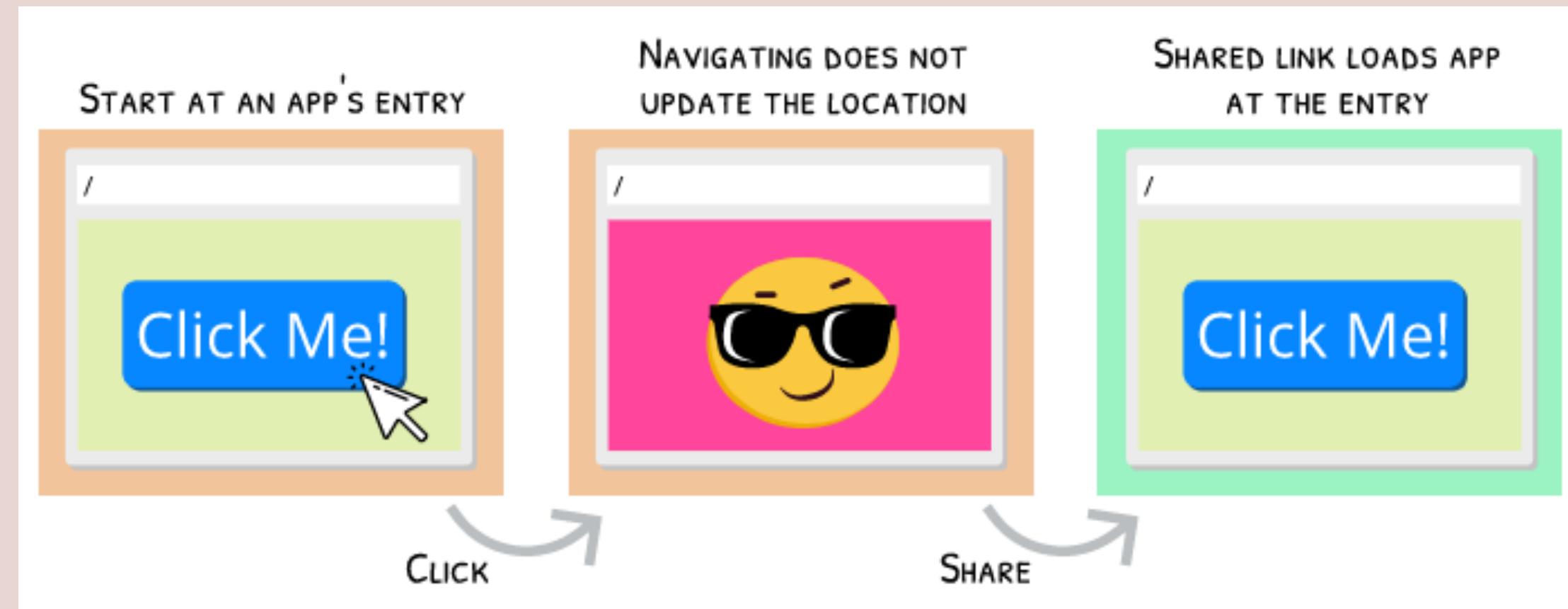
# BUILDINGAN SPA

# BUILDING AN SPA

- » An SPA consists of the following:
  - » Data (see Async JS slides)
  - » Routing
  - » Templates

# ROUTING

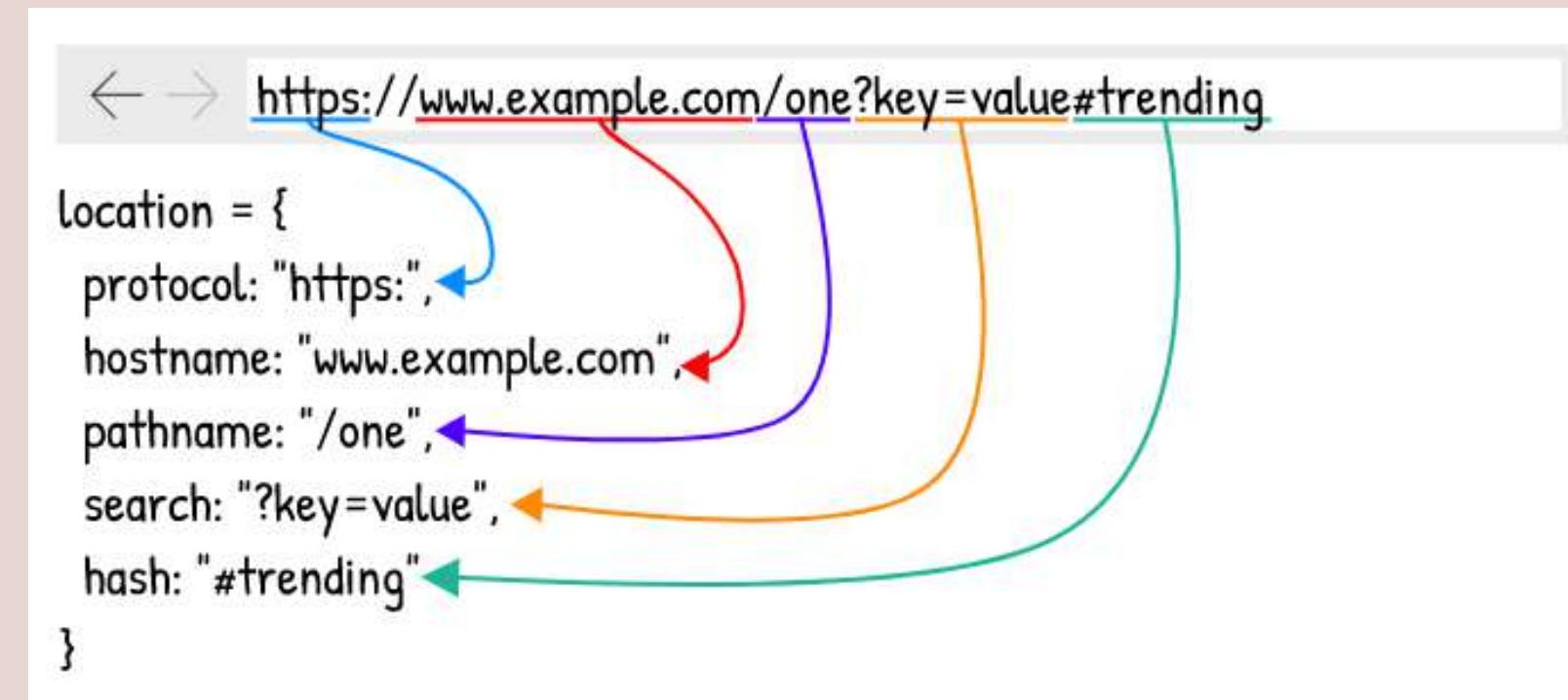
“A location-based SPA render can immediately render the desired content<sup>1</sup>”



<sup>1</sup> <https://blog.pshrmn.com/how-single-page-applications-work/>

# ROUTING LOCATION

“The `window.location` properties map directly from the URL<sup>1</sup>”



<sup>1</sup> <https://blog.pshrmn.com/how-single-page-applications-work/>

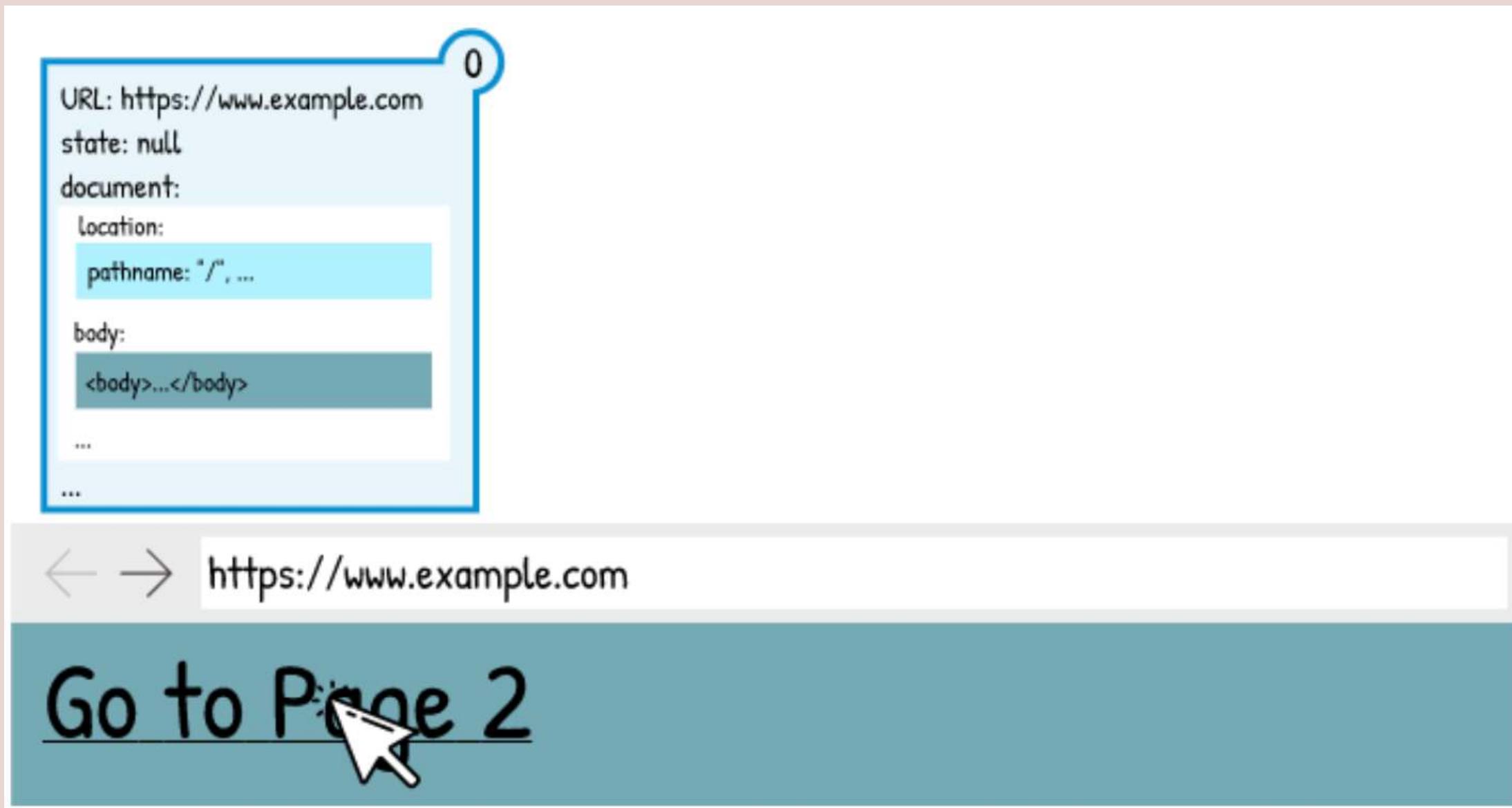
# ROUTING LOCATION

window.location // returns information about the current location

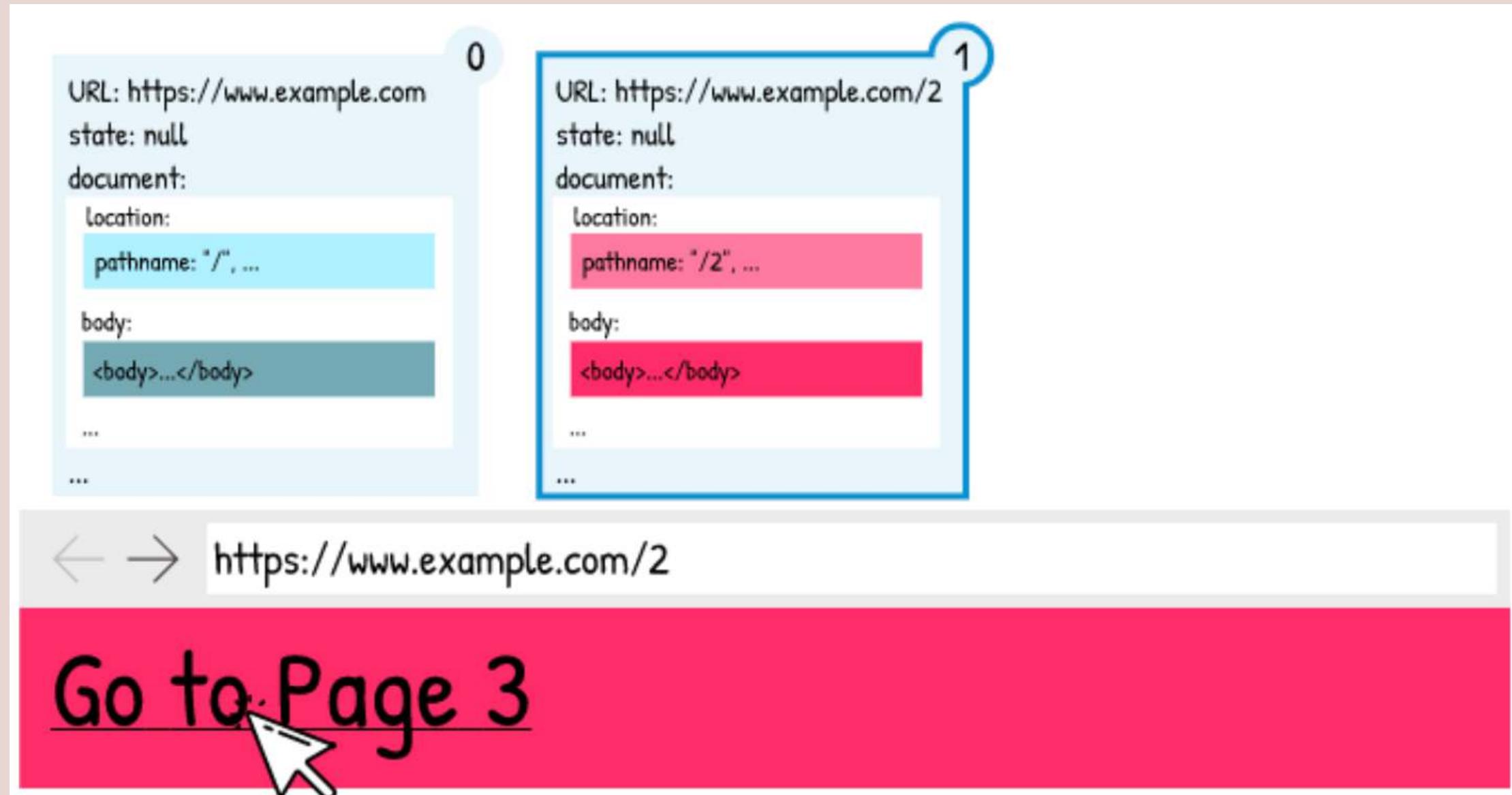
```
// {  
//   "ancestorOrigins": {},  
//   "href": "https://www.fh-salzburg.ac.at/studium/dmk/multimediatechnology-bachelor",  
//   "origin": "https://www.fh-salzburg.ac.at",  
//   "protocol": "https:",  
//   "host": "www.fh-salzburg.ac.at",  
//   "hostname": "www.fh-salzburg.ac.at",  
//   "port": "",  
//   "pathname": "/studium/dmk/multimediatechnology-bachelor",  
//   "search": "",  
//   "hash": ""  
// }
```

# BROWSER SESSION

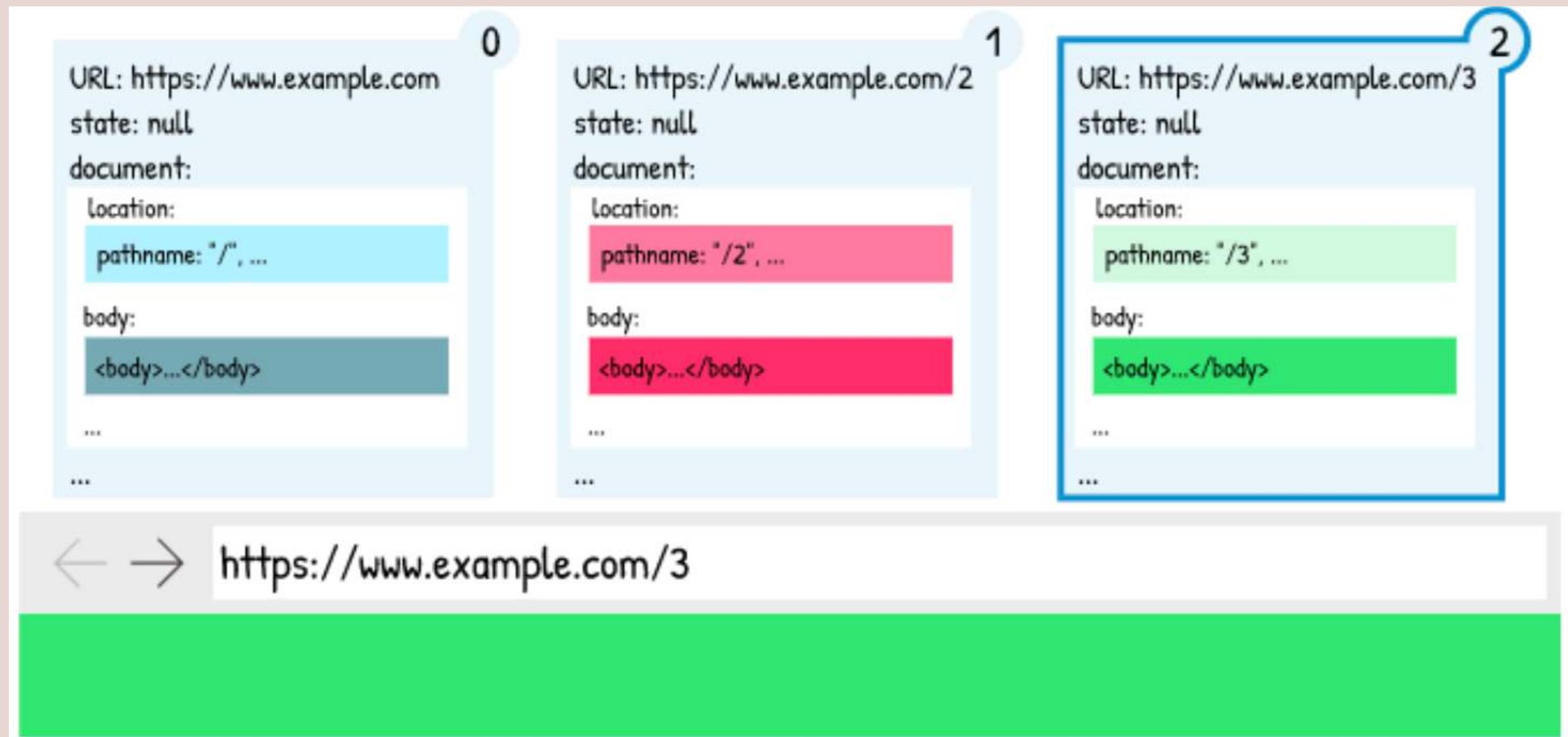
# BROWSER SESSION LINKS



# BROWSER SESSION LINKS



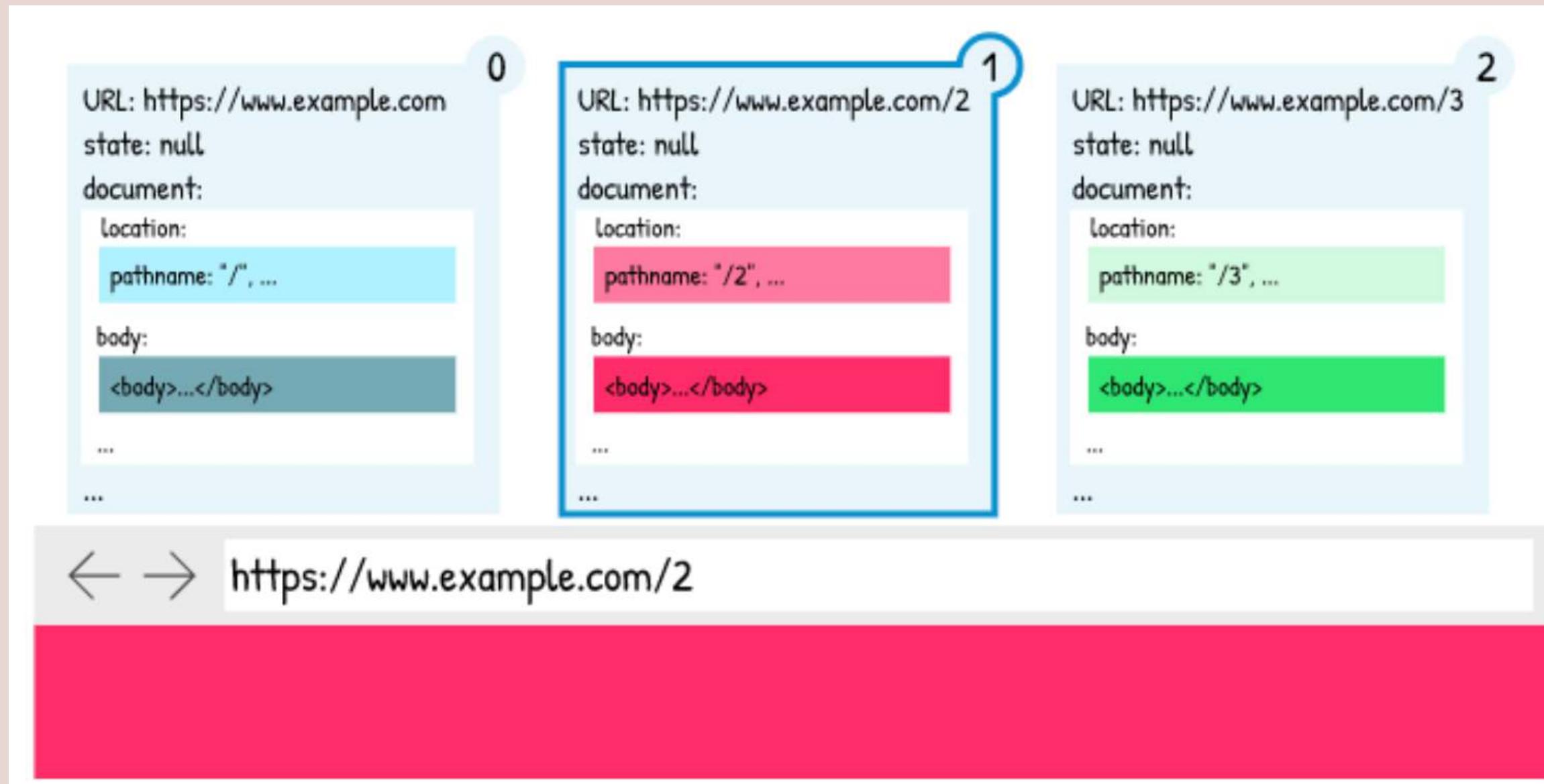
# BROWSER SESSION LINKS



# BROWSER SESSION BACK BUTTON



# BROWSER SESSION BACK BUTTON



# ROUTING LOCATION

“Each browser tab has a “browsing context”. The browsing context maintains a “session history”, which is essentially an array of location entries.”



# BROWSER HISTORY

The History API has three core functions:

- » `pushState()`
- » `replaceState()`
- » `go()`

# BROWSER HISTORY

## PUSHSTATE

```
// history.pushState(<state>, <title>, <url>)

history.pushState({}, 'page 1', '/page1')
// `pushState` adds a new entry to the history
// current history object ['/page1']

history.pushState({}, 'page 2', '/page2')
// current history object ['/page1', '/page2']
//                                         ^
//                                         ^
//                                         ^
//                                         ^
//                                         ^
//                                         ^
// adds /page2 to the history
```

# BROWSER HISTORY

## HISTORY.PROTOTYPE.REPLACESTATE

```
// history.replaceState(<state>, <title>, <url>)

history.pushState({}, 'page 1', '/page1')
// current history object ['/page1']

history.replaceState({}, 'page 2', '/page2')
// current history object ['/page2']
//                                         ^
// replaces /page1 with /page2
```

# BROWSER HISTORY

## HISTORY.PROTOTYPE.BACK

```
history.pushState({}, 'page 1', '/page1')
// current history object ['/page1']
```

```
history.pushState({}, 'page 2', '/page2')
// current history object ['/page1', '/page2']
```

```
history.back()
// history ['/page1', '/page2']
//           ^^^^^^
```

# BROWSER HISTORY EXERCISE

- » Go to any webpage (e.g. medium.com) and navigate around
- » Then open the console and type history or window.history
- » Type history.go(-1) or history.back()

# BROWSER HISTORY EXERCISE

- » Push a new state: `history.pushState({ name: 'FHS' }, '', '/user')`
- » Check the bar on top and see how it changes the path
- » Check the history object again with `history`
- » Replace the current state with `replaceState()`, see how the length of the history doesn't change
- » Check the history state with `history.state`

# BROWSER HISTORY EXERCISE

» Assign `window.location = "/somerule"` in comparison and look at the `window.location`

# ROUTING IN SPA

# LINKS & NAVIGATION

- » Classic Website
- » Click on a link
- » Browser send request
- » Presents document

# LINKS & NAVIGATION

- » Single Page Application
- » Click on a link
- » Browser might do something (e.g. fetch data)
- » A certain area or complete page gets replaced within the current document

# SINGLE PAGE APPLICATIONS

## ROUTING

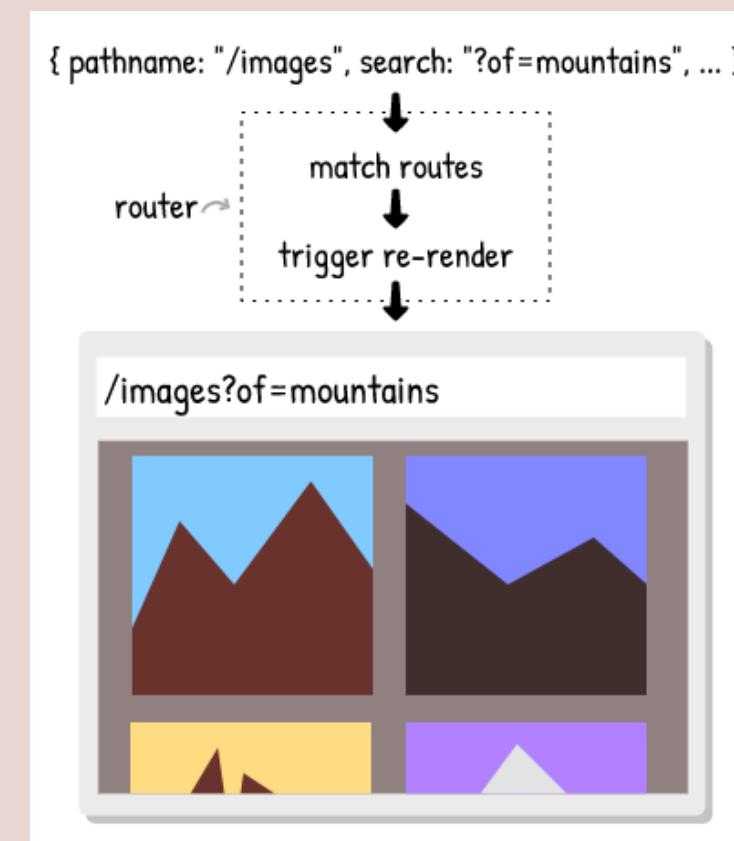
- » Single-page application generally rely on a router.
- » Routers are made up of routes, which describe the location that they should match.

```
const routes = [
  { path: '/' },
  { path: '/about' }, // static
  { path: '/album/:id' } // dynamic
  //           ^^^^
  // dynamic part
];
```

# SINGLE PAGE APPLICATIONS

## ROUTE MATCHING

- » The application renders based on the route that matches the location



# NAVIGATING IN SPAs

## LINK HANDLING

- » add click handler to link
  - » call event.preventDefault()
  - » removes native behavior
- » call history.pushState / history.replaceState

# NAVIGATING IN SPAs

## LINK HANDLING

```
const link = document.querySelector('#my-link')

link.addEventListener('click', (evt) => {
// ^^^^^^
// attach event listener on click

  evt.preventDefault()
// removes default behaviour (no navigation will take place)

  history.pushState(null, "My new page", evt.target.href)
// navigate to URL from link
})
```

# NAVIGATING IN SPAs

## ATTACH TO ALL LINKS

```
const allLinks = document.querySelectorAll('a')
// ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
// find all links in the document

Array.from(allLinks).forEach(link => {
// ^^^^^^^^^^
// convert NodeList to Array

  link.addEventListener('click', evt => {
    evt.preventDefault() // remove default behaviour from link
    history.pushState(null, "My new page", evt.target.href)
  })
})
```

# NAVIGATING IN SPAs

## READING THE CURRENT URL

```
const url = new URL(window.location);
//                                     ^^^^^^
// reads the current URL as string

url.host // "website.com"
url.hostname // "website.com"
url.href // "https://website.com/homepage"
url.origin // "https://website.com"
url.pathname // "/homepage"
url.protocol // "https:"
```

# NAVIGATING IN SPAs

## RENDER SOME CONTENT

```
const onRouteChange = () => {
  const pathname = new URL(window.location).pathname;
  const domElement = document.querySelector('#content')
  //           ^^^^^^^^^^
  // the element to render our content

  if (pathname === '/test1') {
    domElement.innerHTML = 'test1'
  } else if (pathname === '/test2') {
    domElement.innerHTML = 'test1'
  } else {
    domElement.innerHTML = 'not found =('
  }
}
```

# FULL EXAMPLE

» [https://gist.github.com/webpapaya/  
f97f430b7c4f2c894f68644d2cd5ced5](https://gist.github.com/webpapaya/f97f430b7c4f2c894f68644d2cd5ced5)

# TEMPORARIES

# TEMPLATES

- » Presentation of Data
- » Reuse of Code
- » The View Part of an Application

# TEMPLATES

- » Server vs. Client
- » Logic vs. Logic-less Templates
- » EJS vs . Handlebars
- » Native Templates aka. Tagged Template Literals

# TEMPLATING

- » Best Practices
- » Templates should:
  - » not include business logic
  - » not include a lot of logic
  - » be easy to read
  - » be easy to maintain

# TEMPLATES

## TAGGED TEMPLATE LITERALS<sup>3</sup>

```
const myTemplate = () => {
  return html(`<section>
    <h1>Hello FHS</h1>
  </section>
`)}
}
```

```
render(document.getElementById('content'), myTemplate())
```

<sup>3</sup> `html` and `render` are non-standard function and needs to be added to your code

# TEMPLATES

## TAGGED TEMPLATE LITERALS

```
const myTemplate = () => {
  return html(`
    <section>
      <h1>Hello FHS</h1>
    </section>
  `)
}
```

# TEMPLATES ADDING DYNAMIC DATA

```
const myTemplate = () => {
  return html(`
    <section>
      <h1>Today is: ${new Date().toLocaleDateString()}</h1>
    </section>
  `)
}
```

# TEMPLATES CONDITIONALS

```
const myTemplate = (someCondition) => {
  return html(
    <section>
      <h1>${someCondition
        ? 'This is the correct'
        : 'This is the wrong'
      }</h1>
    </section>
  )
}
```

# TEMPLATES LOOP OVER LISTS

```
const myTemplate = (someArray) => {
  return html(
    <section>
      <ul>
        ${someArray.map((value) => `<li>${value}</li>`).join('')}
      </ul>
    </section>
  )
}
```

# TEMPLATES CONVERT TEMPLATE STRING TO DOM NODES<sup>5</sup>

```
const html = (templateString) => {
  return new DOMParser()
    .parseFromString(templateString, 'text/html')
    .body

const render = (nodeToRenderTo, nodeToRender) => {
  nodeToRenderTo.innerHTML = ''
  nodeToRenderTo.appendChild(nodeToRender)
}
```

<sup>5</sup> not relevant for the exam. (next semester we'll look into react). For the exercise simply copy this functions.

# TEMPLATES USER INTERACTIONS

```
const myTemplate = (someCondition) => {
  const domElements = html(`
    <section>
      <h1>My heading</h1>
    </section>
  `)

  domElements.querySelector('h1').addEventListener('click', () => {
    // 1) ^^^^^^^^^^^^^^^^^^^^^^
    // 2) ^^^^^^^^^^^^^^^^^^^^^^
    //
    // 1) select element on which you'd like to add a listener
    // 2) add a listener on the click event

    console.log('I was clicked, it worked!!')
  })

  return domElements
}
```

# HOMEWORK

» see wiki

# FEEDBACK

- » Questions: [tmayrhofer.lba@fh-salzburg.ac.at](mailto:tmayrhofer.lba@fh-salzburg.ac.at)
- » Feedback Link