Client-Side Rendering

Templating UI Elements

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#js-advanced

Table of Contents

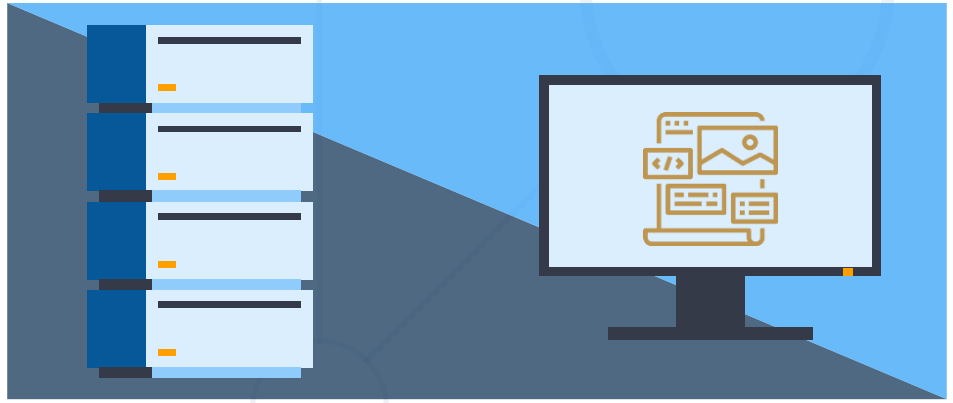
UI Rendering

Building Web Content

Rendering Concepts

- **Rendering** means to **dynamically generate** content
  - As opposed to having **static** HTML files
  - Can be **parts** of a web page, or an **entire web application**
  - Virtually **all contemporary sites** use dynamic generation
- Can be performed on the **server** and on the **client** (browser)

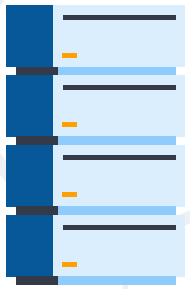
Server-Side Rendering



Client-Side Rendering

Server-Side vs Client-Side

- **Server-Side**
  - User sends request
  - Server **generates HTML**
  - **HTML is sent** to the client
  - Browser **interprets** HTML



- **Client-Side**
  - User sends request
  - CDN serves **files** and **JS**
  - JS **fetches** data
  - JS **generates** DOM elements

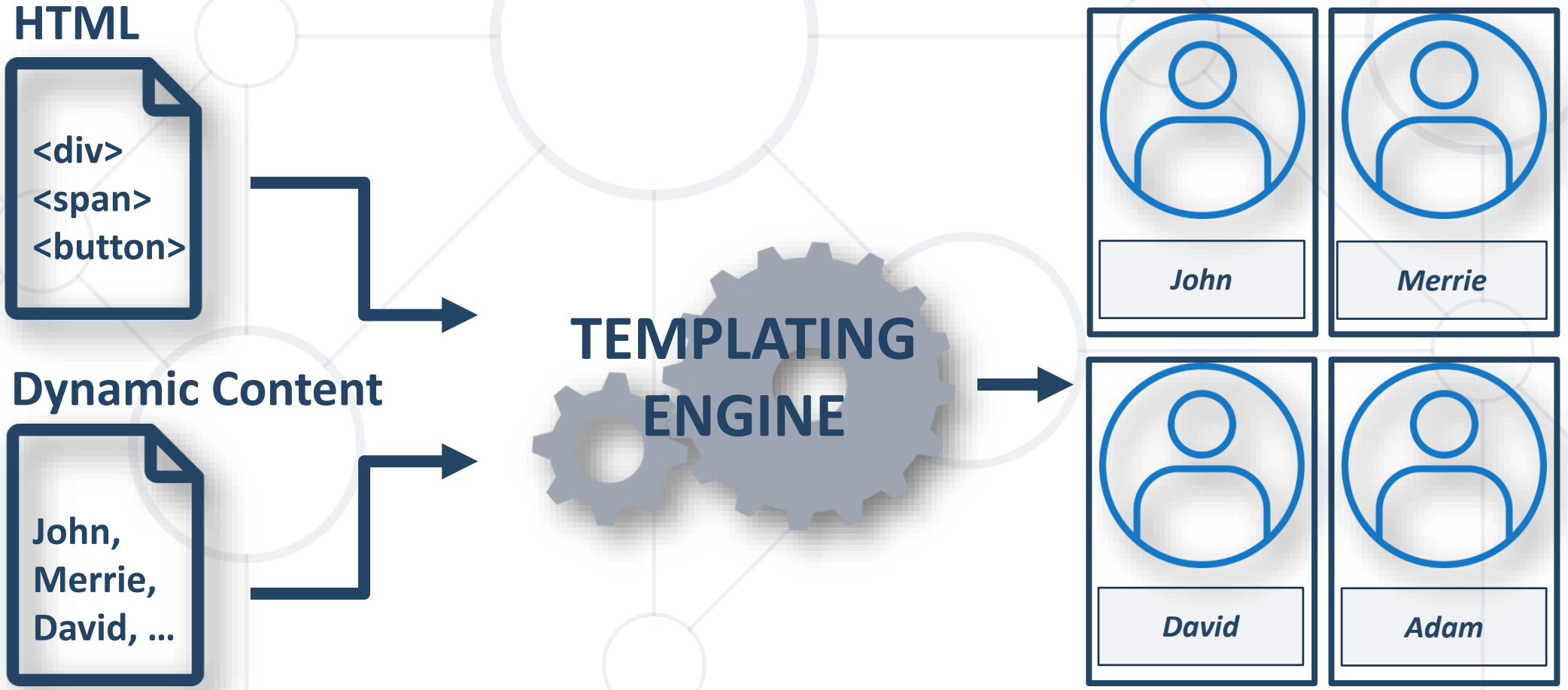
Pros and Cons of Client-Side Rendering

Software University

- **Benefits**:
  - The page is **never reloaded**, and interaction is **instant**
  - State and data can be **shared** across views
  - **Only** dynamic content needs to be **fetched** after start
- **Drawbacks**:
  - Longer **initial load** times
  - **Not** SEO-friendly
  - Poor **performance** with **slow client** machines

What is Templating?

- Templates allow similar content to be **replicated** in a web page, **without repeating** the corresponding markup everywhere



Templating Concepts

- On the **server**, templates are used to **generate HTML**
  - E.g., content from a **database** is inserted into **placeholders**
- On the **client**, templates are used to **create DOM elements**
  - The **template** defines the **structure** of a view
  - Content is **fetched** from a **REST service**
  - The structure is **recreated** and **populated** with the data
  - A **templating engine** is used to streamline the process

Templating Benefits

- **Productivity** – avoid repeating markup
- **Save bandwidth** – fetch just the dynamic content
- **Composability** – reuse elements on multiple pages
- **Separation of concerns** – separate views from logic
- **Interactivity** – instant feedback to the user

Templating Best Practices

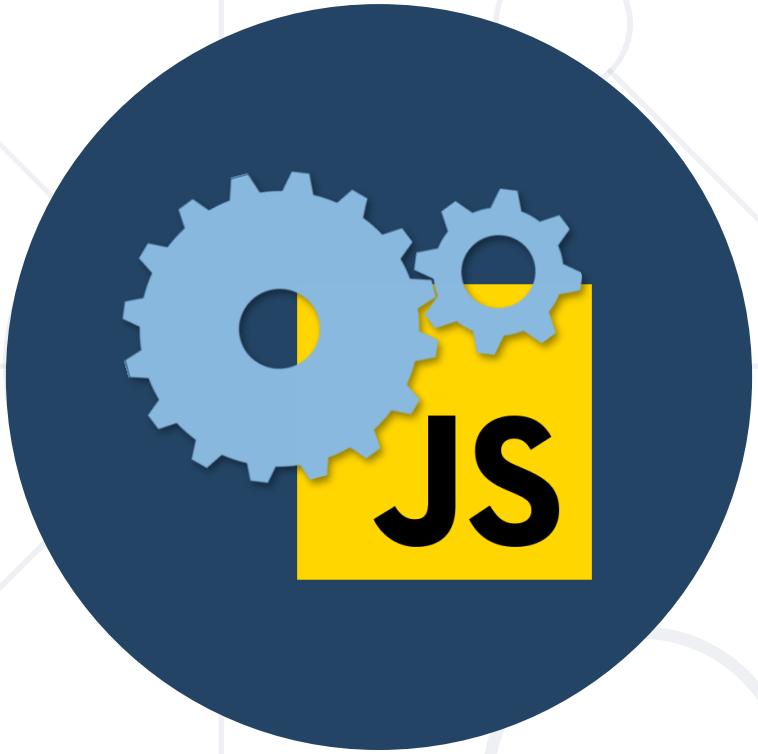
- Templates should be as **simple** as possible
  - **Do not** write business logic in the templates
- Follow the principles of **functional programming**
  - Templates are basically **pure functions**

Custom Templates

Creating a Simple Templating Engine

Project Requirements

- A templating engine **generally** allows:
  - Templates to be **defined** in files, **separate** from business logic
  - A **markup syntax** close to HTML to be used
  - Values to be inserted via **rendering context**
  - Templates to be **composed** to create **layouts**
- **Additional features** of some libraries:
  - **Caching** of template results
  - **Automating** diff-checking and **partial updates**

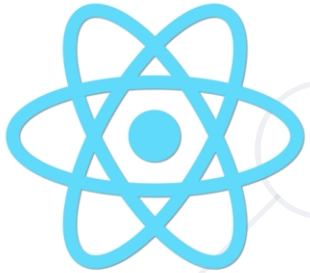


Templating Engines

Overview of Popular JS Libraries

Popular Templating Engines

- **Frameworks:**



React

Vue

Angular

- **Standalone Packages:**

Handlebars

lit-html

Web Components

External Templating Library

Using **lit-html** to Generate Content from Templates

What is lit-html?

- **lit-html** is and efficient, expressive **templating library**

```
let sayHello = (name) => html`<h1>Hello ${name}</h1>`;
render(sayHello('World'), document.body);
```

  - Part of the Polymer Project
  - Allows **rendering** and **partial updating** of templates
  - Uses **standard** JavaScript and HTML **syntax**
  - Can be **customized** and extended
  - **Compatible** with all major browsers

Getting Started

Software University

- **Installation** via **npm** package:

```
npm install lit-html
```

- Direct **import** from online **CDN** (no installation):

```
import {html, render} from 'https://unpkg.com/lit-html?module';
```

- Online **live editors**:
  - CodeSandbox
  - JSBin
  - StackBlitz

Usage

- To use lit-html, **import** it as a module:

```
<script type="module">
  import { html, render }
      from './node_modules/lit-html/lit-html.js';
  …
</script>
```

Path to main file (use live-server to start)

```
let sayHello = (name) => html`<h1>Hello ${name}</h1>`;
render(sayHello('World'), document.body);
```

Rendering a Template

Software University

- lit-html has two main APIs:
  - The **html template tag** used to write templates.
  - The **render()** function used to render a template to DOM container

```
const template = // Template definition
```

```
render(template(state), document.body);
```

Populate with data

Parent node

Tag Functions / Tagged Templates

- A tagged template is a **function call** that uses a **template literal** from which to get its arguments

```
// Tag Function Call
greet`I'm ${name}. I'm ${age} years old.`
```

- Create a greet function and just log the arguments:

```
function greet() {
    console.log(arguments[0]); // array
    console.log(arguments[1]); // name
    console.log(arguments[2]); // age
}
```

Attribute Binding

- In addition to using expressions in the text content of a node, you can bind them to a node's attribute and property values, too:

```
const myTemplate = (data) => html`<div
    class=${data.cssClass}>Stylish text.</div>`
```

- Use the **?** prefix for a **boolean** attribute binding:

```
const myTemplate = (data) => html`<div
    ?disabled=${!data.active}>Stylish text.</div>`
```

Property Binding

- You can also bind to a **node's JavaScript properties** using the **.** **prefix** and the property name:

```
const myTemplate = (data) => html`<input
      .value=${data.value}></input>`;
```

- You can use property bindings to **pass** complex data **down** the tree to subcomponents:

```
const myTemplate = (data) => html`<my-list
      .listItems=${data.items}></my-list>`;
```

Handling Events

- Templates can also include declarative event listeners
- An event listener looks like an attribute binding, but with the **prefix @** followed by an **event name**:

```
const appRootTemplate = (ctx) => html`
    <div>
        <h1 @click=${ctx.handleClick}>${ctx.title}</h1>
    </div>
`
```

Conditional Statements

- lit-html has **no built-in control-flow** constructs. Instead you use normal JavaScript expressions and statements:

```
html`
  ${user.isloggedIn
      ? html`Welcome ${user.name}`
      : html`Please log in`
  }
`;
```

List Rendering

- To render lists, you can use `Array.map` to transform a list of data into a list of templates:

```
html`
  <ul>
    ${items.map((item) => html`<li>${item}</li>`)}
  </ul>
`;
```

Directives: classes and classMap

- The **classMap directive** lets you set a **group of classes** based on an object:

```
import { classMap } from './node_modules/lit-
html/directives/class-map.js';

const itemTemplate = (item) => {
  const classes = { selected: item.selected };
  return html`<div class="menu-item
    ${classMap(classes)}">Classy text</div>`;
}
```

Directives: styles and styleMap

- You can use the **styleMap directive** to set **inline styles** on an element in the template:

```
import { styleMap } from './node_modules/lit-
    html/directives/style-map.js';

const styles = {
 color: myTextColor,
 backgroundColor: highlight ? myHighlightColor :
    myBackgroundColor
};

html`<div style=${styleMap(styles)}>Hi there!</div>`;
```

Directives: repeat

- Repeats a **series of values** generated from an iterable, and **updates** those items **efficiently** when the iterable changes:

```
import { repeat } from './node_modules/lit-
     html/directives/repeat';

const myTemplate = () => html`
  <ul>
    ${repeat(items, (i) => i.id, (i, index) => html`
      <li>${index}: ${i.name}</li>`)}
  </ul>
`;
```

Summary

- Client-side rendering is used in most modern applications
- Templates speed up and simplify the development process
- For easier and more efficient rendering use lit-html

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Software University

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg