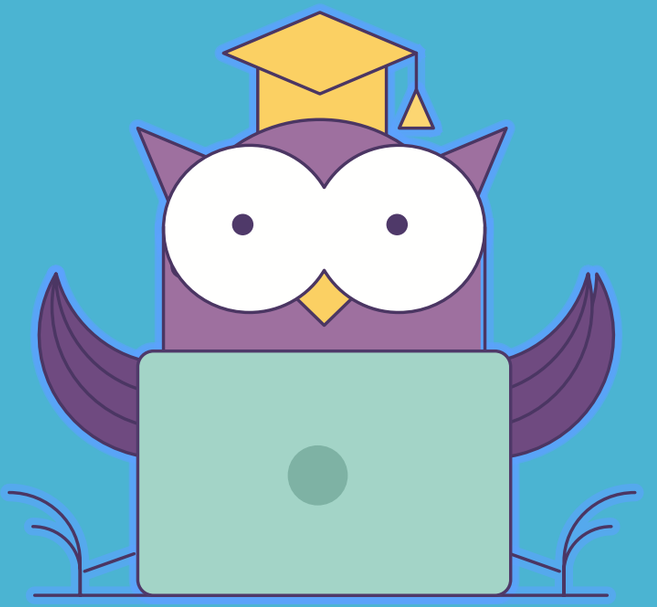
ОНЛАЙН-ОБРАЗОВАНИЕ

Modern JavaScript Frameworks

Стили и Шаблоны

Александр Коржиков

Как меня слышно и видно?



> Напишите в чат

+ если все хорошо

– если есть проблемы со звуком или с видео

Темы предыдущего занятия

- Custom Elements
  - Specification
  - Standalone Elements
  - Built-in Elements
  - LifeCycle Hooks

WEB COMPONENTS

TEMPLATES	SHADOW DOM	HTML IMPORTS	CUSTOM ELEMENTS
`<template id=“”> </template>`	`div #document-fragment span`	`<link rel=“import” href=“part.html”>`	`<my-elem> </my-elem>`

Темы

- Shadow DOM
  - Dom
  - Slots
  - Events
  - Styles
- HTML Template

Цели

Понимать и работать с веб спецификациями **Shadow DOM** и **HTML Template**

Docs

- Shadow DOM v1: Self-Contained Web Components
- Web Components - MDN
- CSS Scoping Module Level 1
- How JavaScript works: the internals of Shadow DOM + how to build self-contained components

Preferences

- [x] Display variable values inline while debugging
- [x] Enable CSS source maps

Default indentation: 4 spaces

Elements

- [ ] Show rulers

Color format: As authored

- [x] Show user agent shadow DOM

```
▼<input id="q" aria-hidden="true" autocomplete="off" name=
"q" tabindex="-1" type="url" jsaction="mousedown:
ntp.fkbxclk" style="opacity: 0;">
  ▼#shadow-root (user-agent) == $0
      <div></div>
  </input>
```

Custom Elements

"A custom element is an element that is custom" © WHATWG

```
customElements.define('hello-element',
  class extends HTMLElement {
    constructor() {
      super()
      this.attachShadow({
        mode: 'open'
      })
    }
    connectedCallback() {
      this.shadowRoot.innerHTML = '<a href="#">My link</a>'
    }
  }
)
```

```
<hello-element test="test">???</hello-element>
```

Shadow DOM

"Shadow DOM fixes CSS and DOM. It introduces scoped styles to the web platform"

"Shadow DOM removes the brittleness of building web apps"

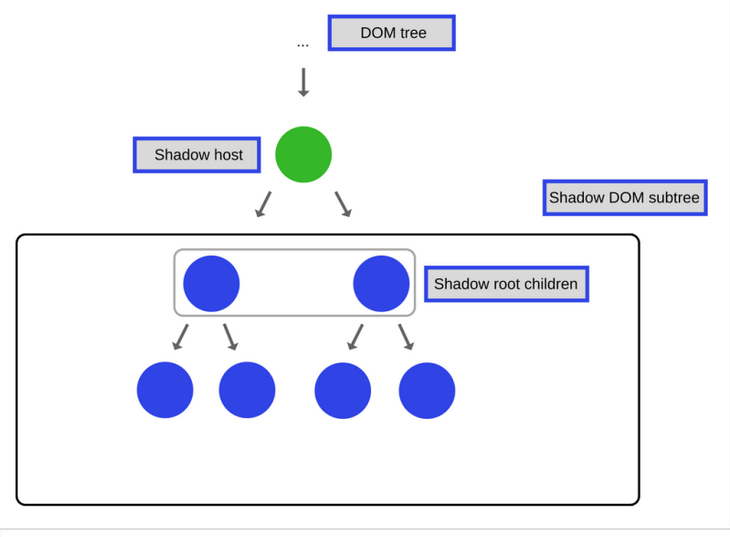
Features

- Isolated DOM - **document.querySelector()** не будет работать
- Composition - компонентный подход
- Scoped CSS - стили не применяются на документ
- Simplifies CSS - можно использовать простые селекторы

Почти **iframe**!

Shadow Definitions

- Tree - отдельный **DOM**
- Root - **Document Fragment**
- Host - элемент "родитель"
- **mode = 'open' || 'closed'**




```
const host = document.createElement('div')
const shadowRoot = host.attachShadow({ mode: 'open'})
shadowRoot.innerHTML = '<h1>ShadowDOM</h1>'

// host.shadowRoot === shadowRoot
// shadowRoot.host === host
// openOrCloseShadowRoot ?!
```

Shadow DOM

"...a method of combining multiple DOM trees into one hierarchy and how these trees interact with each other within a document, thus enabling better composition of the DOM"

© W3C

```
const host = document.createElement('div')
const shadowRoot = host.attachShadow({
  mode: 'open'
})

shadowRoot.innerHTML = `
  <style>h3{ color: red; }</style>
  <h3>Shadow DOM</h3>
`
```

Самостоятельная работа

Создать новый **Custom Element** с **Shadow DOM** внутри

```
class HelloComponent extends HTMLElement {
/* ?
/* const shadowRoot = this.attachShadow({ mode: 'open' })
/* shadowRoot.innerHTML = '<h1>Hello Shadow DOM</h1>' */
}


// customElements.define ?

<!-- ? -->
```

Slots

Slots are **placeholders** inside your component that users can fill with their own markup

```
<p>
  <slot name="my-text">default</slot>
</p>

<my-paragraph>
  <ul slot="my-text">
    <li>different text</li>
    <li>in a list</li>
  </ul>
</my-paragraph>
```

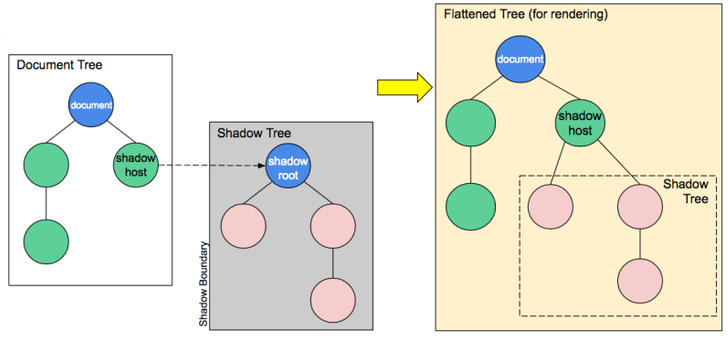
oh-my-slot

```
customElements.define('oh-my-slot', class extends HTMLElement {
  constructor() {
    super()
    this.attachShadow({
      mode: 'open'
    })
  }
  connectedCallback() {
    this.shadowRoot.innerHTML = `My Element
      <slot name="title">Default</slot>
      <slot>Default</slot>`
  }
})
```

```
<oh-my-slot>
  <h1 slot="title">Title</h1>
  <pre>Code</pre>
</oh-my-slot>
```

3 DOMs

- Light - пользователя
- Shadow - разработчика
- Flattened - результат




```
this.shadowRoot.innerHTML = `My Element
  <slot name="title">Default</slot>
  <slot>Default</slot>`
```

```
<oh-my-slot>
  <h1 slot="title">Title</h1>
  <pre>Code</pre>
</oh-my-slot>
```

slots

- **assignedNodes()**
- **slotchange**

events

- lose target
- custom events
- **composedPath()**

Styles

Shadow DOM позволяет описывать изолированные стили

```
#shadow-root
<style>
  #tabs {
    display: inline-flex;
    ...
  }
</style>
<div id="tabs">
  ...
</div>
```

Features

- CSS селекторы внутри Shadow DOM применяются локально
- CSS селекторы снаружи не применяются к компоненту

```
#shadow-root
 <style>
 :host {
 display: block;
/* by default, custom elements are display: inline */
/* :host(.blue) {color: blue;} */
 }
 </style>
```

- **:host** позволяет описывать стили применяемые к компоненте
- Снаружи **host** приритетнее
- **:host-context()** для контекста

Вопросы

- Что не так в изолированных стилях?
- Что с этим делать?

CSS custom properties (variables)

```
element { --main-bg-color: brown;}

element { background-color: var(--main-bg-color);}
element { color: var(--my-var, red);}
```

- Inherited (можно определить на элементе host)
- Fallback (можно задать стандартное значение)

Custom styling

- **:defined** - известные элементы
- **::slotted**
- Deprecated
  - @apply (CSS Mixins)
  - ::content
  - ::shadow/deep/

```
::slotted(h1) {
  color: red;
}

@keyframes foo {
  from {
    color: red;
  }
  to {
    color: blue;
  }
  @apply --not-a-style-rule;
}
```

Demo

- Использовать локальный **style** для нового **Custom Element**
- Переопределить стиль **host** элемента снаружи (**color** или **background**) Задать **CSS custom properties (variables)**

HTML Template

"The template element is used to declare fragments of HTML that can be cloned and inserted in the document by script"

© WHATWG

```
<template id="mytemplate">
  <img src="" alt="great image">
  <div class="comment"></div>
</template>
```

importNode

document.importNode() creates a new copy of the specified Node

```
const template = document.querySelector('#mytemplate')

template.content.querySelector('img').src = 'logo.png'
const clone = document.importNode(template.content, true)
document.body.appendChild(clone)
```

Features

- Content is effectively inert until activated
- Any content within a template won't have side effects
- content is considered not to be in the document
- **<template>** can be placed anywhere
- content is Yet Another documentFragment

Template & Shadow DOM

```
<template id="mytemplate">
  <style>
    :host {
      color: red
    }
  </style>
  <img src="" alt="great image">
  <div class="comment">that's me</div>
</template>
```

```
const template = document.querySelector('#mytemplate')

const clone = document.importNode(template.content, true)
const div = document.createElement('div')
const root = div.attachShadow({ mode: 'open' })
root.appendChild(clone)

document.body.appendChild(div)
```

Самостоятельная работа

Добавить элемент с помощью **template** на страницу со **script** внутри, склонировать и добавить копию в документ, использовать **style** и **Shadow DOM**

```
<template id="mytemplate">
 <style> /* ... */ </style>
 <!-- ... -->
</template>
// ...
const root = div.attachShadow({ mode: 'open' })
```

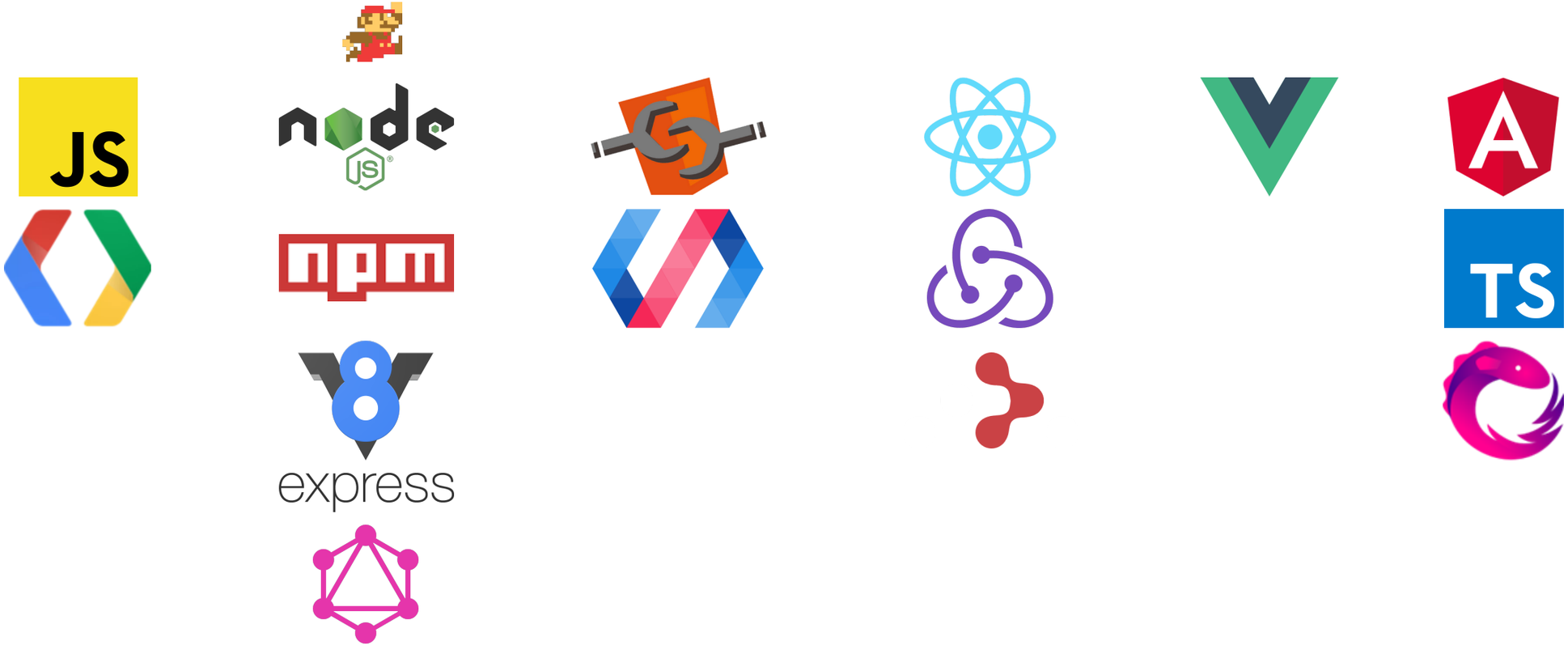
Переопределить стиль элемента снаружи

HTML Template

"Templates allow you to declare fragments of markup which are parsed as HTML, go unused at page load, but can be instantiated later on at runtime"

Разобрали веб спецификации **Shadow DOM** и **HTML Template**

Modern JavaScript Frameworks



Самостоятельная работа

Сделать приложение для показа дерева элементов с помощью **Custom Elements my-tree** и **my-leaf**

```
{
  id: 1,
  items: [{
    id: 2,
    items: [{
      id: 3
    }]
  }]
}
```

Спасибо за внимание!

Вы верите в **Shadow DOM**?

Вы верите в **Web Components**?

Пожалуйста, пройдите опрос в личном кабинете

TODO

- [ ] Спросить про каникулы