PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Tarea 4: Web Components

Lit Elements + Lit-html

José Pedro Del Río
Manuel Espinoza

Profesor Jaime Navon
Ayudante Benjamín Vicente G.

Lit

Simple.

Fast.

Web Components.

```
> npm i lit
```

Get Started

Simple

Skip the boilerplate

Building on top of the Web Components standards, Lit adds just what you need to be happy and productive: reactivity, declarative templates and a handful of thoughtful features to reduce boilerplate and make your job easier. Every Lit feature is carefully designed with web platform evolution in mind.

Fast

Tiny footprint, instant updates

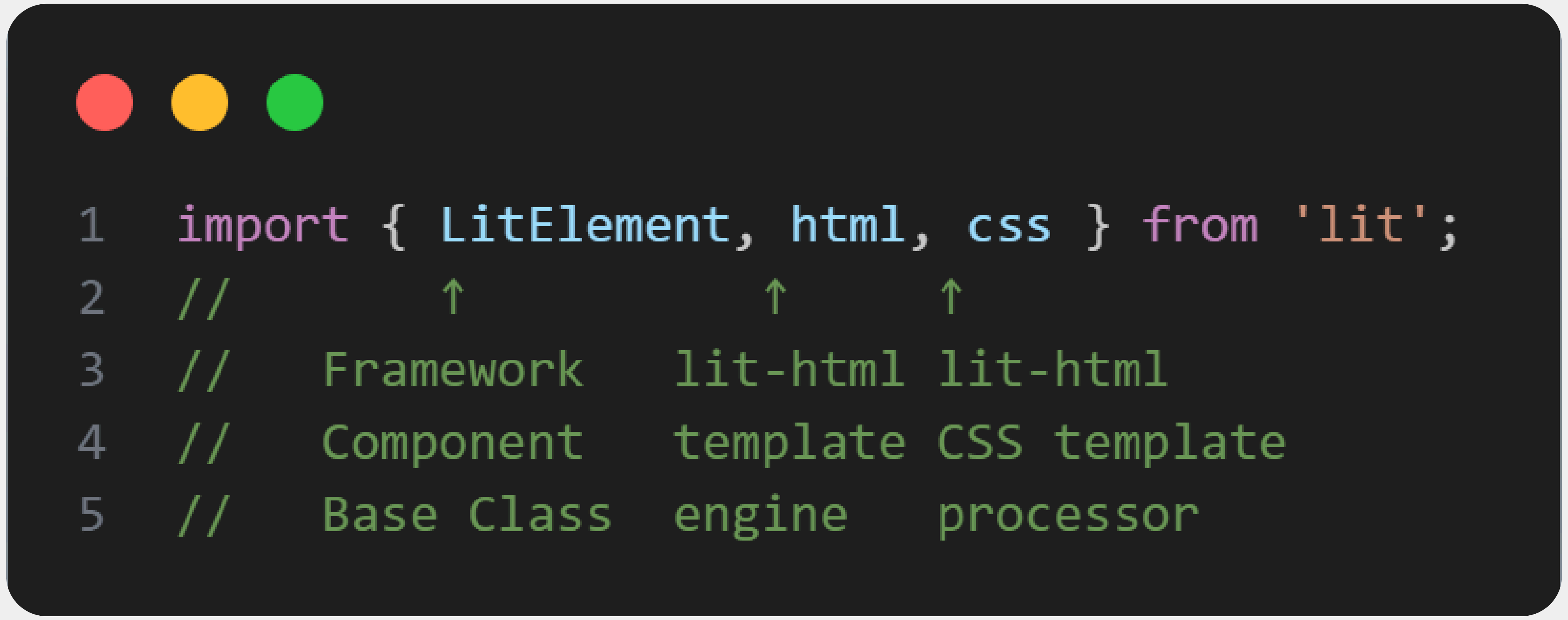
Weighing in at around 5 KB (minified and compressed), Lit helps keep your bundle size small and your loading time short. And rendering is blazing fast, because Lit touches only the dynamic parts of your UI when updating — no need to rebuild a virtual tree and diff it with the DOM.

Web Components

Interoperable & future-ready

Every Lit component is a native web component, with the superpower of interoperability. Web components work anywhere you use HTML, with any framework or none at all. This makes Lit ideal for building shareable components, design systems, or maintainable, future-ready sites and apps.

Import de Lit: Separación de responsabilidades



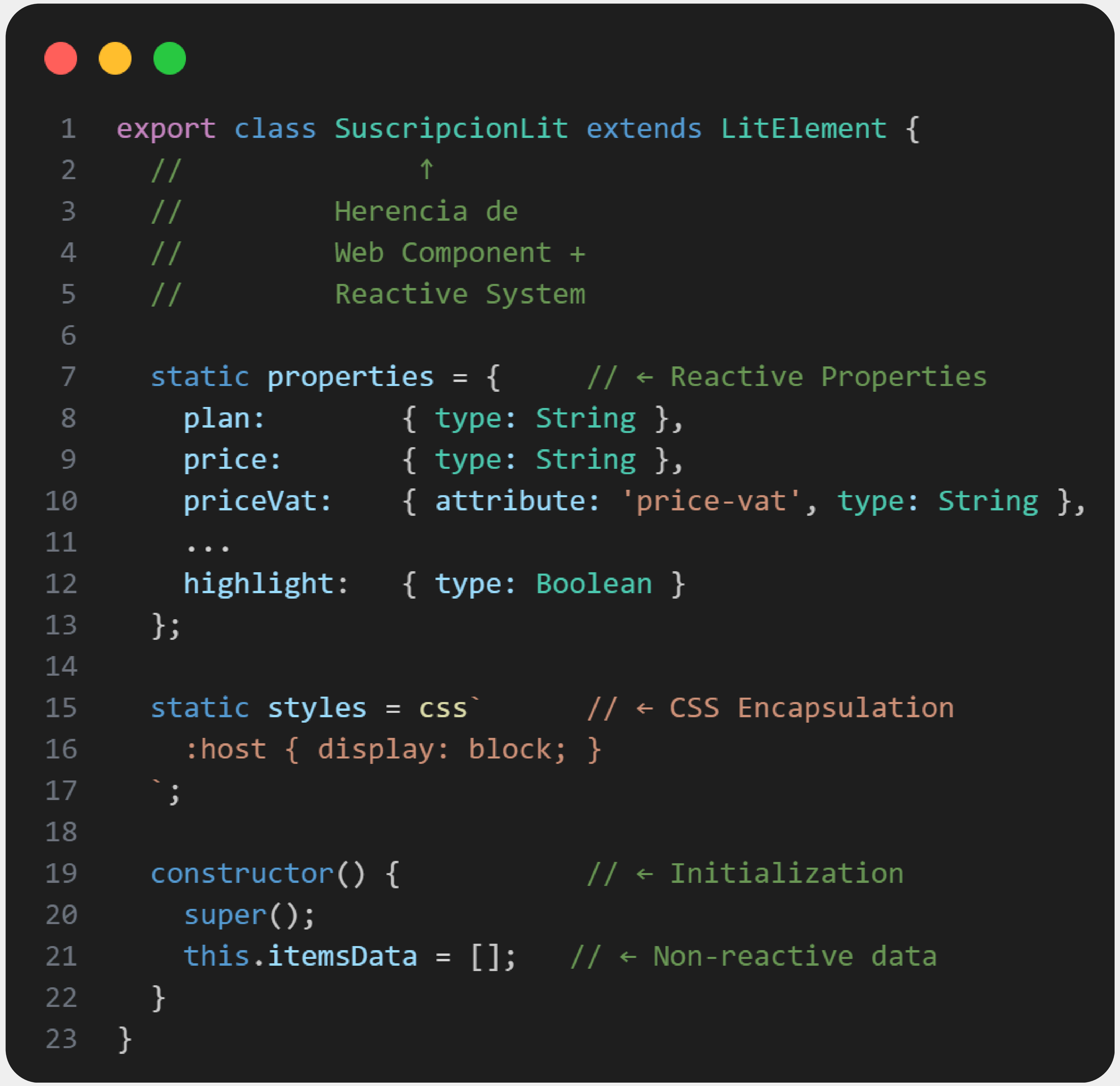
```
import { LitElement, html, css } from 'lit';
//       ↑          ↑     ↑
//   Framework   lit-html lit-html
//   Component   template CSS template
//   Base Class  engine   processor
```

ListElement: Clase padre que proporciona lifecycle hooks, property system, y integración automática con lit-html para rendering

html: Convierte template literals en TemplateResult objects optimizados para DOM

css: Processor para CSS-in-JS. Genera CSSResult objects, para encapsulación en Shadow DOM.

Estructura Base del Componente



```
export class SuscripcionLit extends LitElement {
  //              ↑
  //         Herencia de
  //         Web Component +
  //         Reactive System

  static properties = {     // ← Reactive Properties
    plan:        { type: String },
    price:       { type: String },
    priceVat:    { attribute: 'price-vat', type: String },
    ...
    highlight:   { type: Boolean }
  };

  static styles = css`      // ← CSS Encapsulation
    :host { display: block; }
  `;

  constructor() {           // ← Initialization
    super();
    this.itemsData = [];   // ← Non-reactive data
  }
}
```

Arquitectura de Herencia: Proporciona automatic change detection y property-to-attribute synchronization.

Sistema de Propiedades Reactivas: Declaración de reactive properties que triggean re-render automático.

`type:` `Automatic conversion (String, Number, Boolean)`

`attribute:` `Custom attribute name mapping`

CSS y State Management: CSS encapsulation via Shadow DOM. Entrega style isolation sin runtime overhead.

`Reactive properties:` `Automatic re-rendering`

`Non-reactive data (itemsData):` `Manual update control para performance`

Lifecycle Hooks y Procesamiento de Datos

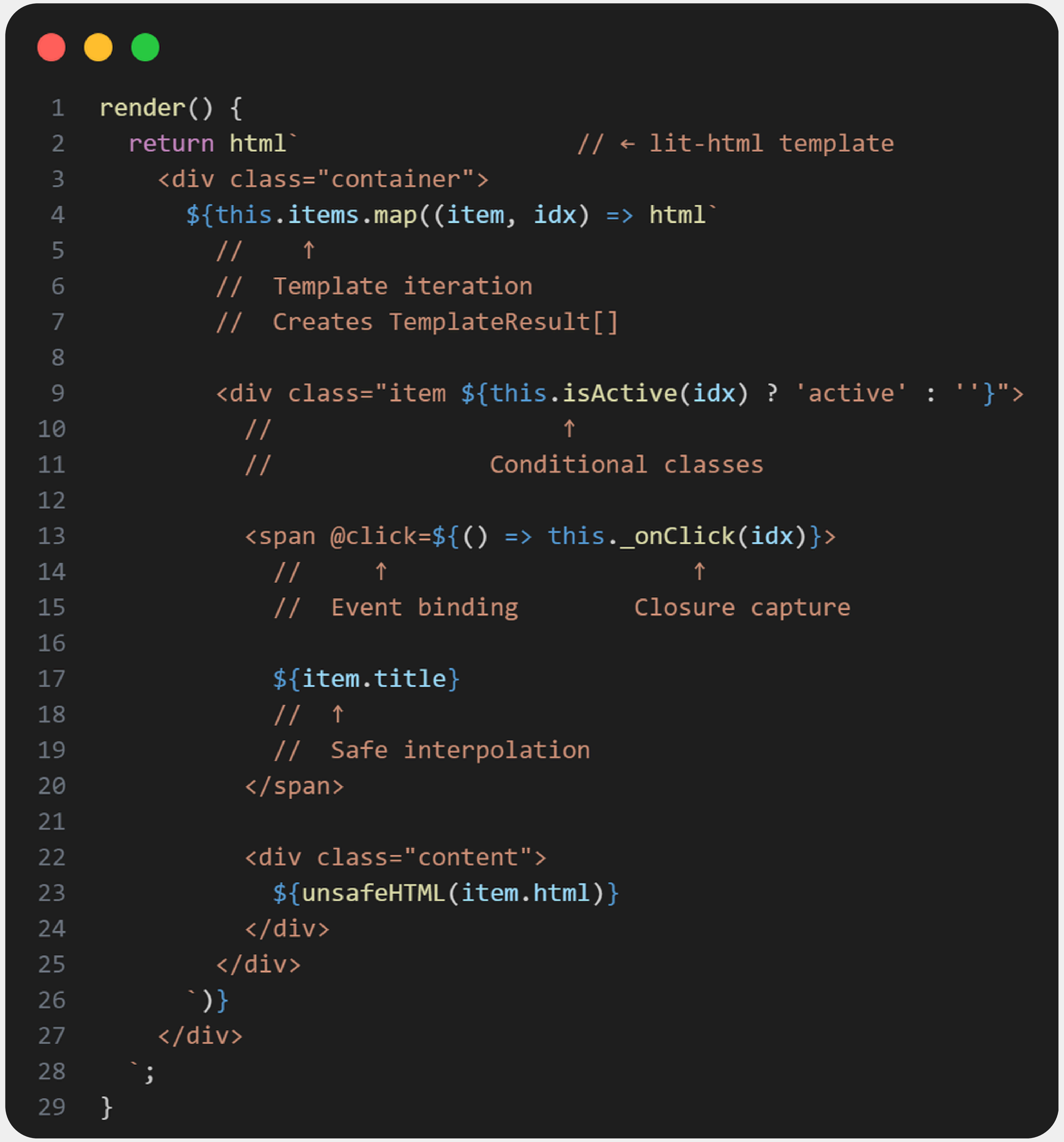
<acordion-lit.js>

```
firstUpdated() {
  super.firstUpdated();
  this._collectItemsFromLightDOM();
  this.requestUpdate();
}

```

firstUpdated(): Se ejecuta después del primer render, garantizando que DOM está disponible. Ideal para setup inicial, Light DOM processing, y configuración one-time.

Otros Hooks: `connectedCallback(), disconnectedCallback(), updated(changedProperties), willUpdate(changedProperties), requestUpdate()`

lit-html: Template Rendering y Directives



```
render() {
  return html`                   // ← lit-html template
    <div class="container">
      ${this.items.map((item, idx) => html`
        //    ↑
        //  Template iteration
        //  Creates TemplateResult[]

        <div class="item ${this.isActive(idx) ? 'active' : ''}">
          //                    ↑
          //               Conditional classes

          <span @click=${() => this._onClick(idx)}>
            //     ↑                     ↑
            //  Event binding        Closure capture

            ${item.title}
            //  ↑
            //  Safe interpolation
          </span>

          <div class="content">
            ${unsafeHTML(item.html)}
          </div>
        </div>
      `)}
    </div>
  `;
}
```

Template iteration: map() retorna array de TemplateResult objects.

Safe interpolation: ${expression} automatically escapes HTML content.

@click syntax: Lit's declarative event binding.

Ternary operators: ${condition ? 'active' : ''} para conditional classes.

Inserción en el Shadow DOM: El resultado de render() se escribe dentro del shadow root que Lit creó en super(), manteniendo encapsulado el HTML y CSS.

Re-render automático: Cuando cualquiera de estas propiedades cambia, Lit detecta la mutación y vuelve a invocar render(), actualizando solo los nodos afectados

Event Handling y State Management

```
this.dispatchEvent(new CustomEvent('subscribe', {
  bubbles: true,     // ← Event propagation hacia arriba
  composed: true,    // ← Shadow DOM boundary crossing
  detail: { data }   // ← Event payload
}));
```

Custom Event Communication:

- **bubbles: true:** Event travels up DOM tree
- **composed: true:** Event crosses Shadow DOM boundaries
- **Together:** Permite true component communication architecture
- **detail:** Payload para pasar data con el event

`Event Lifecycle Management: connectedCallback(), disconnectedCallback()`

CSS Encapsulation Patterns

```
static styles = css`
  :host {                    // ← Component root element
    display: block;
    --accordion-bg: #f0f0f0;  // ← CSS custom property
  }

  :host([disabled]) {        // ← Attribute-based styling
    opacity: 0.5;
  }

  .header {
    background: var(--accordion-bg, white);
    //             ↑           ↑
    //        Custom prop   Fallback

    part: accordion-header;  // ← Expose for external styling
  }
`;
```

:host selector: Apunta al host element desde dentro del Shadow DOM. Permite styling del container sin exponer estructura interna.

::part() pseudo-element: Expone elementos internos especificos para external styling. Más granular que CSS variables.

Style scoping: Browser optimiza CSS matching al scope reducido del Shadow DOM
Style isolation: No CSS selector conflicts ni specificity wars
Adoptable Stylesheets: Estilos compartidos entre multiples shadow roots.

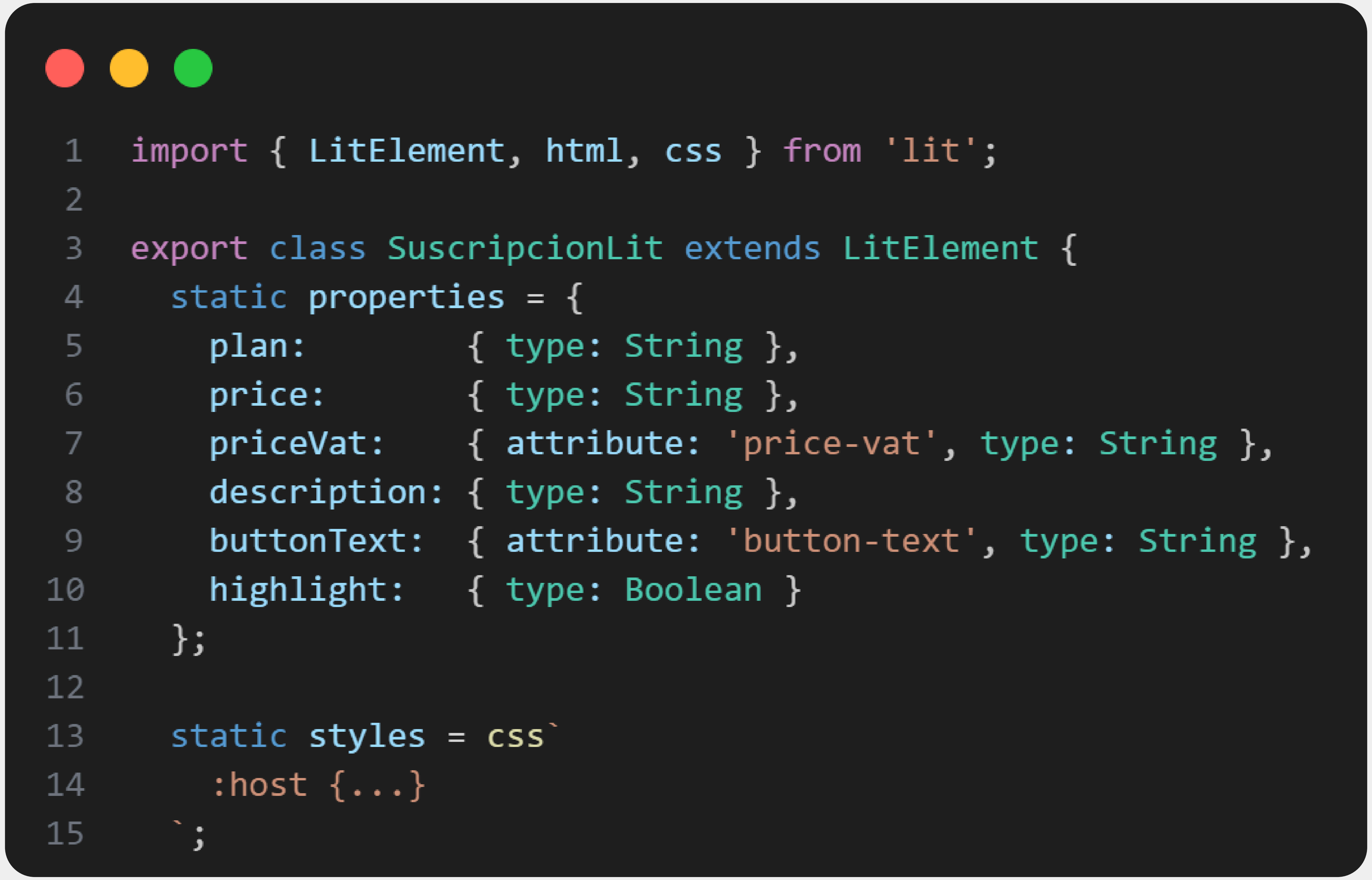
Comparación Vanilla vs Lit

```javascript
// Vanilla Web Components
class VanillaAccordion extends HTMLElement {
  static get observedAttributes() {
    return ['active-index'];
  }

  attributeChangedCallback(name, old, value) {
    if (name === 'active-index') {
      this._activeIndex = parseInt(value);
      this._render();  // ← Manual re-render
    }
  }

  _render() {
    // Manual DOM manipulation
    this.shadowRoot.innerHTML = `...`;
  }
```

```javascript
// Lit Element
class LitAccordion extends LitElement {
  static properties = {
    activeIndex: { type: Number }
  };

  render() {
    return html`...`;  // ← Declarative template
  }
  // ↑
  // Auto re-render on property change
}
```

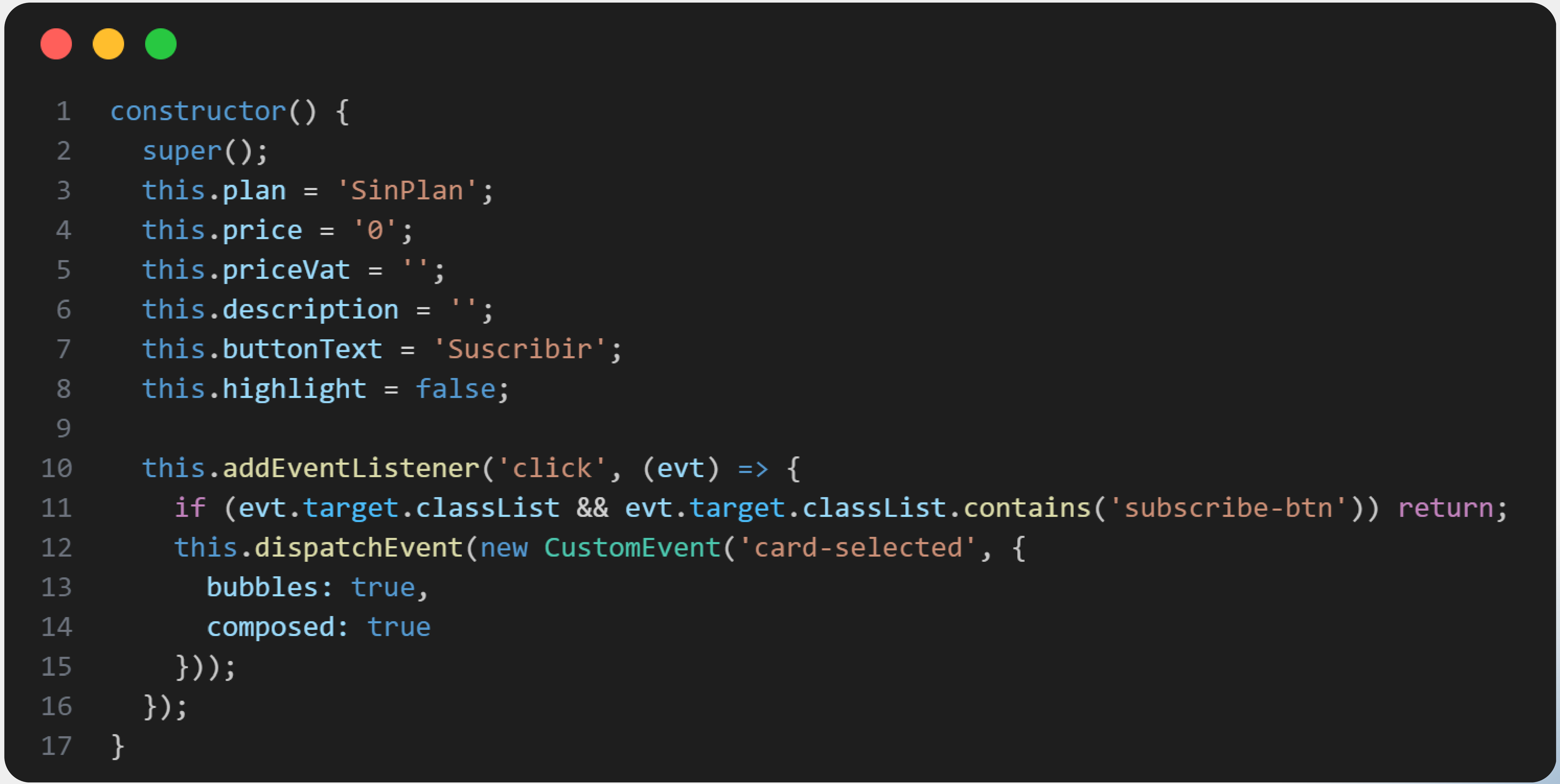
Demo

<subscription>



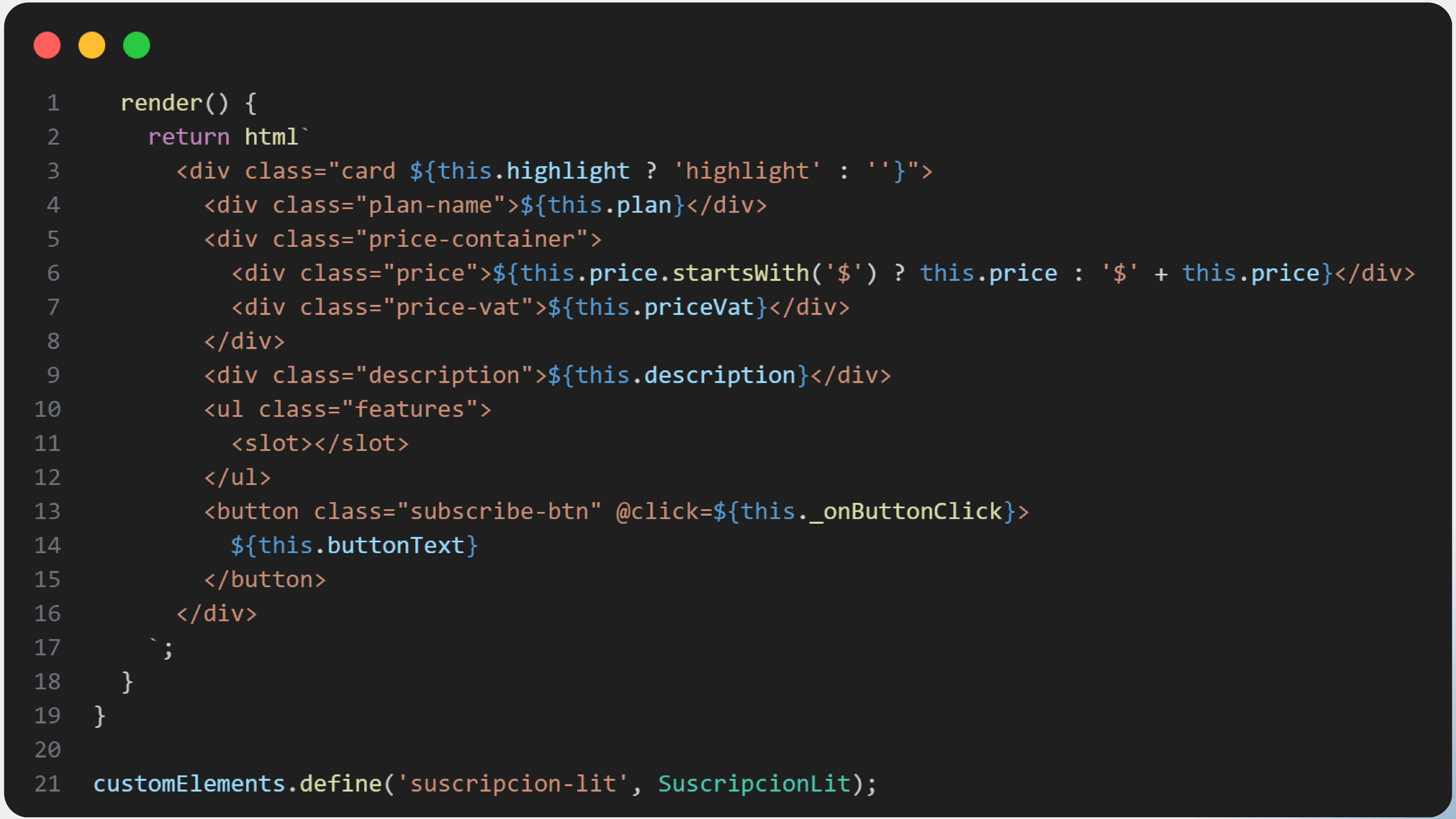
```
import { LitElement, html, css } from 'lit';

export class SuscripcionLit extends LitElement {
  static properties = {
    plan:        { type: String },
    price:       { type: String },
    priceVat:    { attribute: 'price-vat', type: String },
    description: { type: String },
    buttonText:  { attribute: 'button-text', type: String },
    highlight:   { type: Boolean }
  };

  static styles = css`
    :host {...}
  `;
```

<subscription>



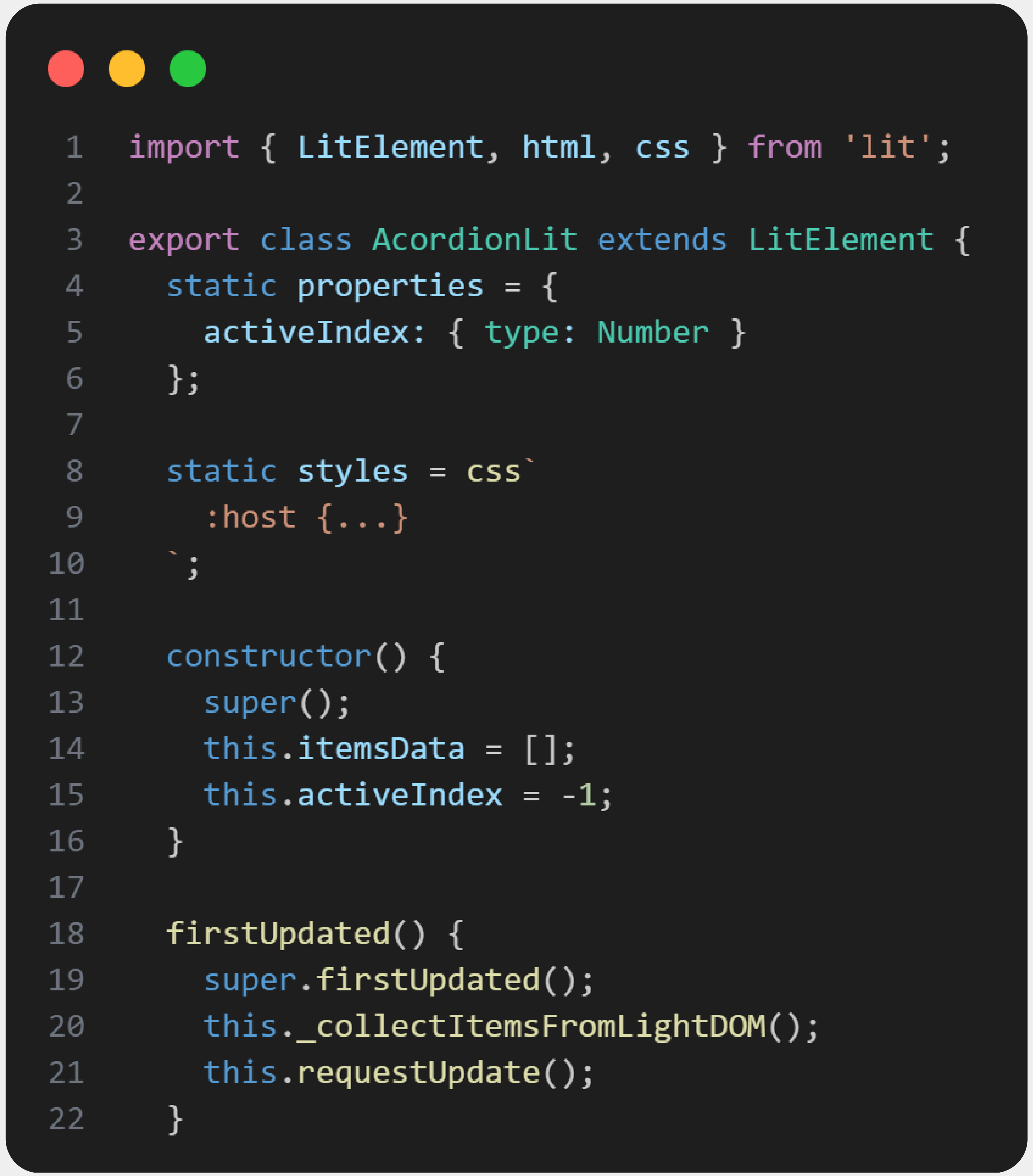
```javascript
constructor() {
  super();
  this.plan = 'SinPlan';
  this.price = '0';
  this.priceVat = '';
  this.description = '';
  this.buttonText = 'Suscribir';
  this.highlight = false;

  this.addEventListener('click', (evt) => {
    if (evt.target.classList && evt.target.classList.contains('subscribe-btn')) return;
    this.dispatchEvent(new CustomEvent('card-selected', {
      bubbles: true,
      composed: true
    }));
  });
}
```

<subscription>

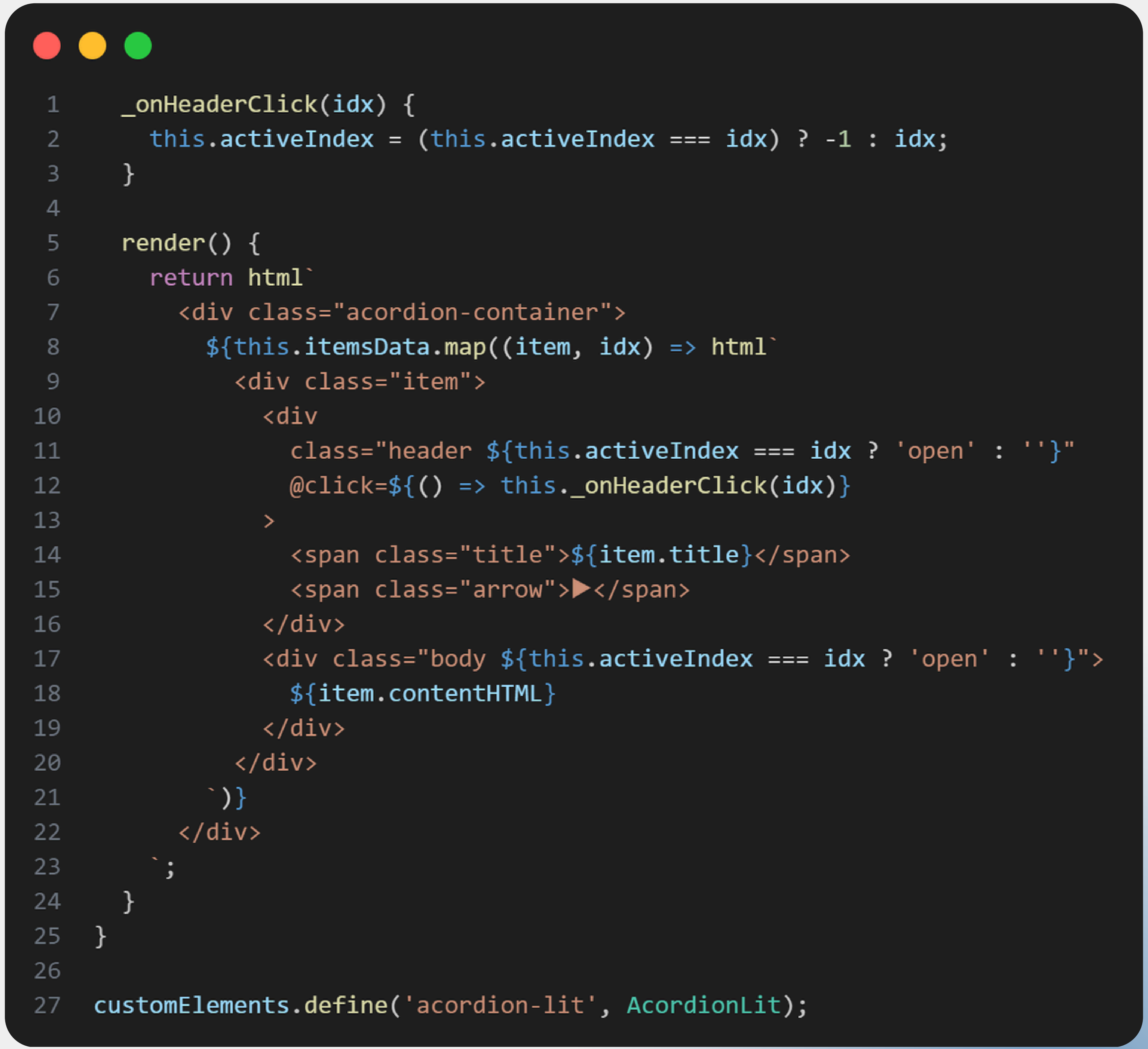


```
  render() {
    return html`
      <div class="card ${this.highlight ? 'highlight' : ''}">
        <div class="plan-name">${this.plan}</div>
        <div class="price-container">
          <div class="price">${this.price.startsWith('$') ? this.price : '$' + this.price}</div>
          <div class="price-vat">${this.priceVat}</div>
        </div>
        <div class="description">${this.description}</div>
        <ul class="features">
          <slot></slot>
        </ul>
        <button class="subscribe-btn" @click=${this._onButtonClick}>
          ${this.buttonText}
        </button>
      </div>
    `;
  }
}

customElements.define('suscripcion-lit', SuscripcionLit);
```

<acordion>

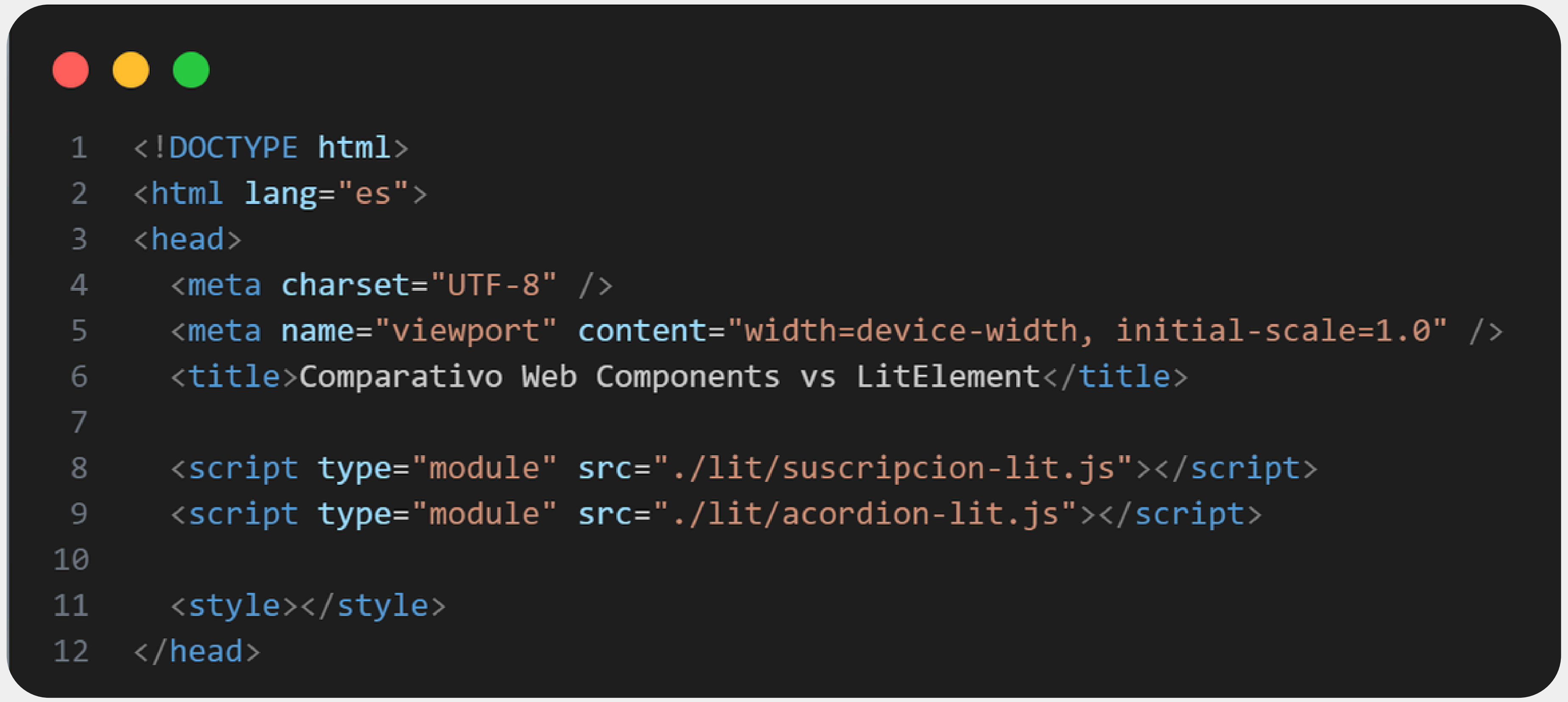


```js
import { LitElement, html, css } from 'lit';

export class AcordionLit extends LitElement {
  static properties = {
    activeIndex: { type: Number }
  };

  static styles = css`
    :host {...}
  `;

  constructor() {
    super();
    this.itemsData = [];
    this.activeIndex = -1;
  }

  firstUpdated() {
    super.firstUpdated();
    this._collectItemsFromLightDOM();
    this.requestUpdate();
  }
```

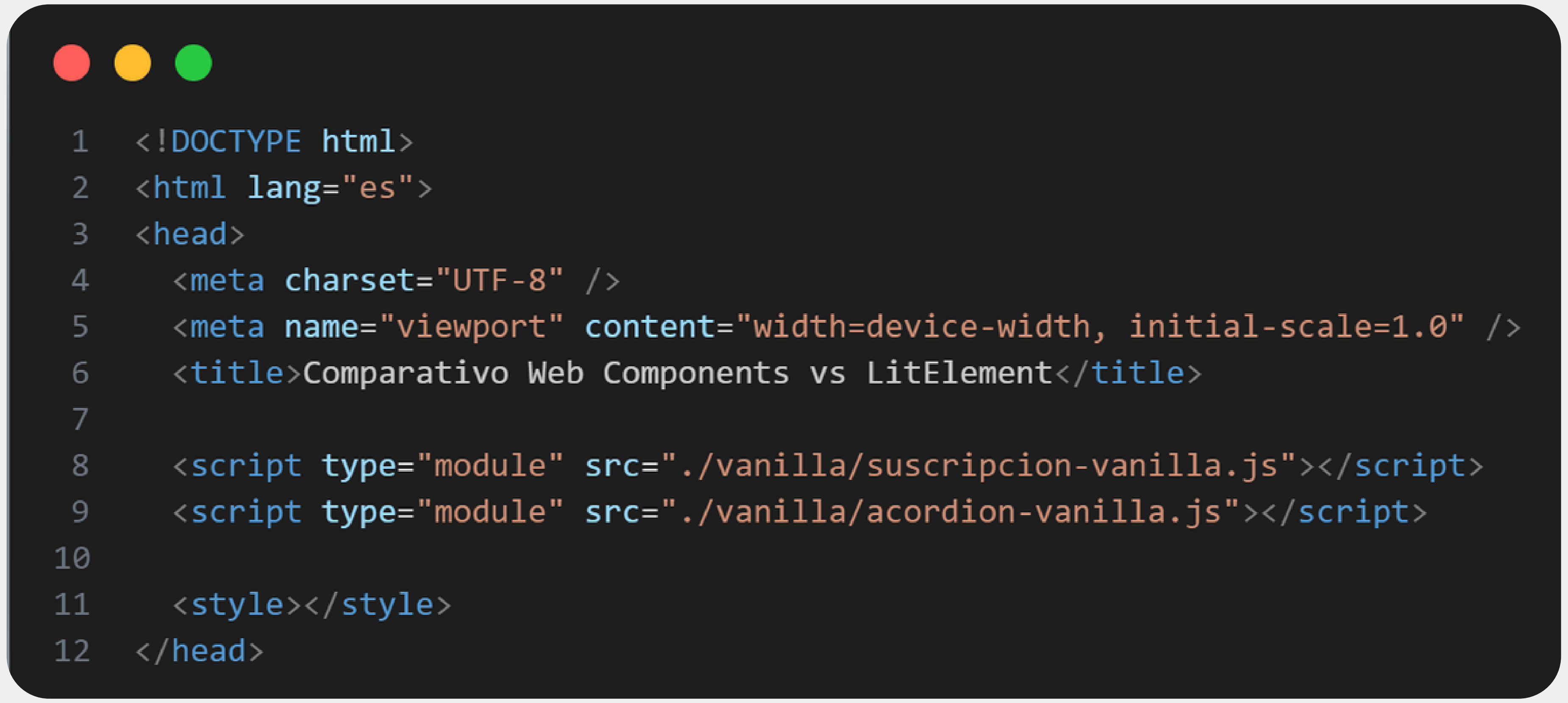


```js
  _onHeaderClick(idx) {
    this.activeIndex = (this.activeIndex === idx) ? -1 : idx;
  }

  render() {
    return html`
      <div class="acordion-container">
        ${this.itemsData.map((item, idx) => html`
          <div class="item">
            <div
              class="header ${this.activeIndex === idx ? 'open' : ''}"
              @click=${() => this._onHeaderClick(idx)}
            >
              <span class="title">${item.title}</span>
              <span class="arrow">▶</span>
            </div>
            <div class="body ${this.activeIndex === idx ? 'open' : ''}">
              ${item.contentHTML}
            </div>
          </div>
        `)}
      </div>
    `;
  }
}

customElements.define('acordion-lit', AcordionLit);
```

Importación de WebComponents en HTML

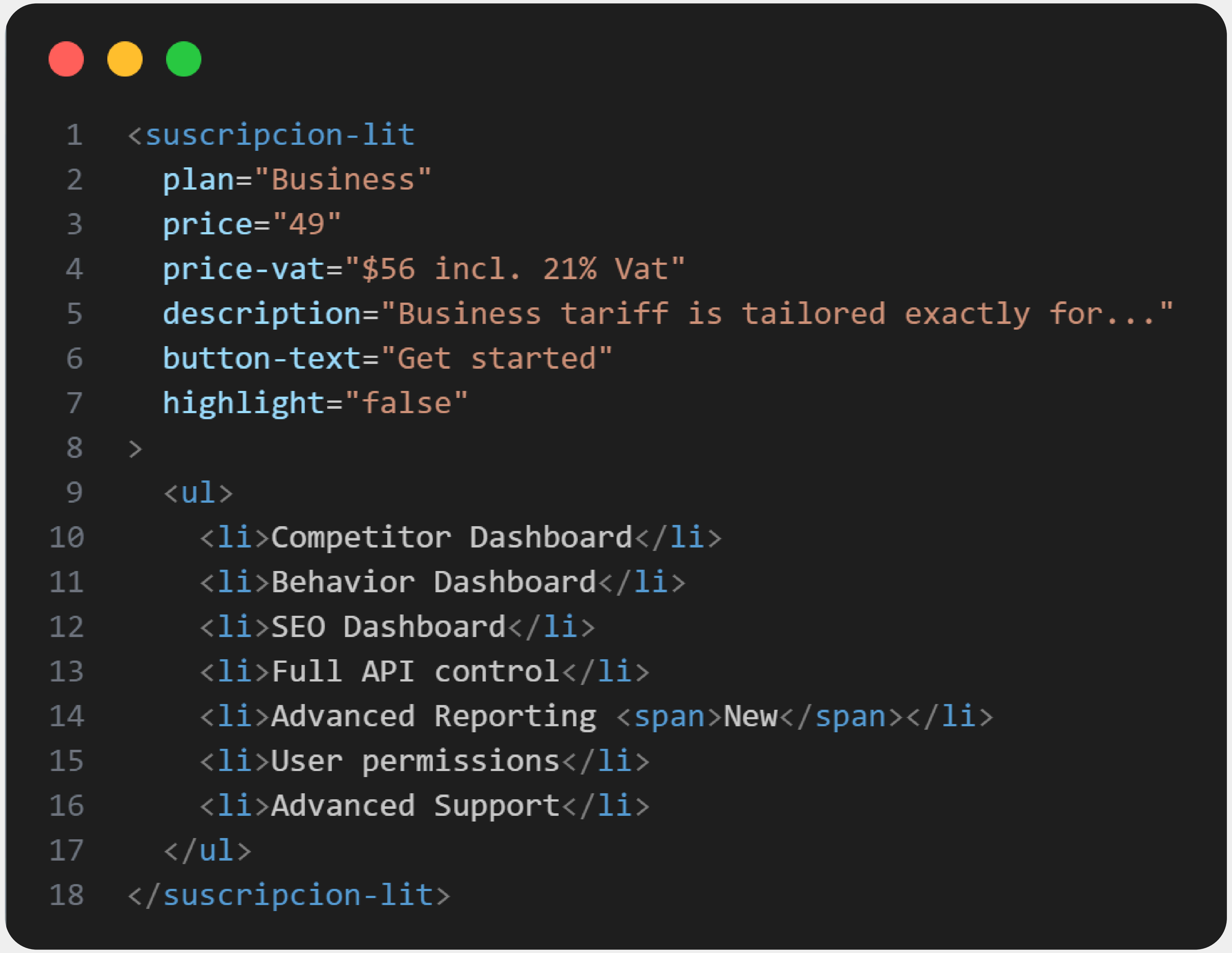


```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Comparativo Web Components vs LitElement</title>

  <script type="module" src="./lit/suscripcion-lit.js"></script>
  <script type="module" src="./lit/acordion-lit.js"></script>

  <style></style>
</head>
```



```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Comparativo Web Components vs LitElement</title>

  <script type="module" src="./vanilla/suscripcion-vanilla.js"></script>
  <script type="module" src="./vanilla/acordion-vanilla.js"></script>

  <style></style>
</head>
```

WebComponents en HTML



```html
<suscripcion-lit
  plan="Business"
  price="49"
  price-vat="$56 incl. 21% Vat"
  description="Business tariff is tailored exactly for..."
  button-text="Get started"
  highlight="false"
>
  <ul>
    <li>Competitor Dashboard</li>
    <li>Behavior Dashboard</li>
    <li>SEO Dashboard</li>
    <li>Full API control</li>
    <li>Advanced Reporting <span>New</span></li>
    <li>User permissions</li>
    <li>Advanced Support</li>
  </ul>
</suscripcion-lit>
```

WebComponents en HTML

```html
<acordion-lit>
  <div data-title="▶ The first item">
    <p>Con LitElement el renderizado de cada sección es más declarativo, y el diff
       de lit-html solo actualiza lo necesario.</p>
  </div>
  <div data-title="▶ The second item">
    <p>Observa cómo cada hijo <div data-title="..."> se transforma en un ítem
       de acordeón. Solo se abre uno a la vez.</p>
  </div>
  <div data-title="▶ The third, longer item">
    <p>Al abrir uno nuevo, el lit-element cierra automáticamente los demás. Esto se
       logra con una propiedad reactiva que almacena el índice activo.</p>
  </div>
</acordion-lit>
```

🔥 Desafios de implementación

Encapsulación vs Flexibilidad: Inicialmente se uso solo un template insertado dentro del shadow DOM, aislando el HTML y CSS del componente, pero no se podía personalizar
Solucion: Uso de variables CSS dentro del :host

Problema de estados compartidos sin romper encapsulación: Coordinar comportamientos entre componentes, sin usar selectores globales ni librerias.
Solucion: Crear eventos personalizados