Modern web platform without a framework

You build the workshop shop using only the browser's HTML, CSS, and JavaScript: accessible, fast, offline-capable, and with Web Components.

Mike Bild

Exercises: github.com/MikeBild/introduction-web-plattform

HTML, accessibility, and forms

Semantic HTML carries half of accessibility · Can you get through the shop using only the keyboard? · The browser validates before your code runs

What does a screen reader see on your page?

- Role, name, and state of each element
- Landmarks as jump points through the page
- Headings as a table of contents

Landmarks give the page a map

- header, nav, main, and footer instead of div containers
- Exactly one main per page
- An aria-label when there are two nav elements

```
<body>
  <header><a href="/">Werkstatt-Shop</a></header>
  <nav aria-label="Hauptnavigation">
    <a href="/werkzeug">Werkzeug</a>
    <a href="/warenkorb">Warenkorb</a>
  </nav>
  <main>
    <h1>Werkzeug</h1>
  </main>
  <footer><a href="/impressum">Impressum</a></footer>
</body>
```

The product list is a list

- ul and li announce how many products there are
- article groups one product with its own heading
- Images get alt that describes the content

```
<ul class="products">
  <li>
    <article>
      <h2>Akkuschrauber</h2>
      <img src="schrauber.webp" alt="Akkuschrauber mit zwei Akkus"
           width="320" height="240">
      <p><data value="8990">89,90 €</data></p>
      <button type="button">In den Warenkorb</button>
    </article>
  </li>
</ul>
```

A button can do more than a div

- button for actions, a with href for navigation
- Focus, Enter, and Space are built in
- type button prevents unintended submits

```
<!-- Bad: no focus, no keyboard, no role -->
<div class="btn" onclick="addToCart('p-1')">In den Warenkorb</div>

<!-- Good: everything built in -->
<button type="button" class="btn" data-product-id="p-1">
  In den Warenkorb
</button>
```

HTML first, then ARIA

- The first rule of ARIA: no ARIA when HTML is enough
- ARIA changes the announcement, not the behavior
- The Accessibility Tree in DevTools shows the result

Who uses your shop without a mouse?

- People with a screen reader, switch, or voice control
- Power users who are faster with Tab and Enter
- The European Accessibility Act applies to many shops

Focus is always visible

- shows the ring only during keyboard use
- outline: none without a replacement is a barrier
- The Tab order follows the order in the HTML

```
:focus-visible {
  outline: 3px solid var(--color-focus, #1d4ed8);
  outline-offset: 2px;
}

/* Never remove without a replacement */
button:focus:not(:focus-visible) {
  outline: none;
}
```

dialog handles focus for you

- showModal sets focus and blocks the rest of the page
- Escape closes it, and the close event reports it
- You remember the trigger for returning focus

```
export function openCart(dialog, trigger) {
  dialog.addEventListener('close', () => trigger.focus(), { once: true })
  dialog.showModal()
}
```

Announce changes without stealing focus

- aria-live with polite announces text as soon as it is quiet
- The region is in the HTML from the start
- Short sentences instead of whole lists

```
// <p id="cart-status" aria-live="polite" class="visually-hidden"></p>
export function announce(region, message) {
  region.textContent = ''
  requestAnimationFrame(() => { region.textContent = message })
}
```

Tools find a lot, but not everything

- Lighthouse and axe check contrasts and names
- The accessibility tree shows what gets announced
- Only keyboard and screen reader show the flow

What can a form do without JavaScript?

- Check required fields, email format, and lengths
- Show errors at the field and set focus
- Send data to the server with POST

Attributes describe the rules

- required, type, and pattern validate without code
- autocomplete fills in address and email
- A hint at the field names the format up front

```
<form id="checkout" method="post" action="/api/orders">
  <label for="email">E-Mail</label>
  <input id="email" name="email" type="email" required
         autocomplete="email">

  <label for="postal-code">Postleitzahl</label>
  <input id="postal-code" name="postalCode" required inputmode="numeric"
         pattern="\d{5}" autocomplete="postal-code"
         aria-describedby="postal-code-hint">
  <p id="postal-code-hint">Fünf Ziffern, zum Beispiel 04109</p>

  <button>Bestellung absenden</button>
</form>
```

You report business rules with setCustomValidity

- Non-empty text makes the field invalid
- Empty text makes it valid again
- reportValidity shows the message right away

```
export function checkQuantity(input, stock) {
  const quantity = input.valueAsNumber
  input.setCustomValidity(
    quantity > stock ? `Nur noch ${stock} Stück auf Lager. Wähl eine kleinere Menge.` : '',
  )
  return input.checkValidity()
}

quantityInput.addEventListener('input', () => {
  checkQuantity(quantityInput, product.stock)
})
```

FormData reads the whole form

- One call instead of one line per field
- The name attribute becomes the key
- On submit, validate first, then read

```
export function readCheckout(form) {
  const data = new FormData(form)
  return {
    email: String(data.get('email') ?? ''),
    postalCode: String(data.get('postalCode') ?? ''),
    quantity: Number(data.get('quantity') ?? 0),
  }
}

form.addEventListener('submit', (event) => {
  event.preventDefault()
  if (!form.reportValidity()) return
  submitOrder(readCheckout(form))
})
```

The server still validates

- Browser validation is convenience, not security
- The server checks the same rules again
- You show its errors at the field again

Modern CSS

Cascade layers end the specificity fight · Grid for the grid, Flexbox for the row · The component asks its container

Why does the wrong rule win?

- Specificity beats file order
- Every fix needs a longer selector
- In the end, only !important still helps

Layers are decided before specificity

- The order appears once at the very top
- A later layer always wins against an earlier one
- Styles without a layer win against all layers

```
@layer reset, base, components, utilities;

@layer base {
  #checkout button { background: gray; }
}

@layer components {
  .button { background: var(--color-accent); }
}
```

Custom properties are runtime variables

- Tokens for colors, spacing, and radii in
- Override locally instead of copying rules
- The second value in var() is the fallback

```
:root {
  --color-accent: #1d4ed8;
  --color-surface: white;
  --space-2: 0.5rem;
  --radius: 0.5rem;
}

.card--sale {
  --color-accent: #b91c1c;
}

.button {
  padding: var(--space-2) calc(var(--space-2) * 2);
  border-radius: var(--radius, 4px);
  background: var(--color-accent);
}
```

Nesting keeps a component together

- & stands for the outer selector
- States and variants stay directly on the component
- Nest no more than two levels deep

```
@layer components {
  .button {
    background: var(--color-accent);
    color: white;

    &:hover {
      background: color-mix(in oklch, var(--color-accent), black 15%);
    }
    &[disabled] {
      opacity: 0.5;
    }
    .card & {
      inline-size: 100%;
    }
  }
}
```

Third-party CSS goes in its own layer

- @import with layer() tames libraries
- Your own components then win without tricks
- Utilities sit at the top as the last layer

```
@import url("vendor/datepicker.css") layer(vendor);

@layer reset, vendor, base, components, utilities;

@layer utilities {
  .visually-hidden {
    position: absolute;
    inline-size: 1px;
    block-size: 1px;
    overflow: hidden;
    clip-path: inset(50%);
  }
}
```

One direction or two?

- Flexbox distributes elements along one axis
- Grid defines rows and columns at the same time
- Both use gap for spacing

A grid without media queries

- auto-fill creates as many columns as fit
- minmax gives each card a minimum width
- 1fr distributes the remaining space evenly

```
.products {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(14rem, 1fr));
  gap: var(--space-4, 1rem);
  padding: 0;
  list-style: none;
}
```

The cart row is a Flexbox

- align-items: center aligns everything in the middle
- margin-inline-start: auto pushes the total to the right
- min-width: 0 lets long names wrap

```
.cart-line {
  display: flex;
  align-items: center;
  gap: var(--space-2, 0.5rem);

  .name {
    flex: 1 1 auto;
    min-width: 0;
    overflow-wrap: anywhere;
  }
  .total {
    margin-inline-start: auto;
    font-variant-numeric: tabular-nums;
  }
}
```

Subgrid keeps cards aligned

- Cards inherit the rows of the outer Grid
- Title, price, and button sit on one line
- Subgrid is available in all current browsers

```
.products {
  grid-template-rows: auto;
  grid-auto-rows: auto auto 1fr auto;
}

.products > li {
  display: grid;
  grid-row: span 4;
  grid-template-rows: subgrid;
}
```

Logical properties follow writing direction

- inline-size instead of width, block-size instead of height
- margin-inline-start instead of margin-left
- Layouts also work from right to left

The viewport says nothing about the card

- The same card appears in the grid, the sidebar, and the cart
- Media queries know only the window width
- Container queries ask about the component’s space

First the container, then the query

- container-type makes an element queryable
- @container checks the width of the nearest container
- The card itself keeps no fixed width

```
.product {
  container: product / inline-size;
}

.card {
  display: grid;
  gap: var(--space-2);
}

@container product (inline-size >= 28rem) {
  .card {
    grid-template-columns: 10rem 1fr;
    align-items: start;
  }
}
```

() looks down

- A parent element reacts to its children
- No more state class from JavaScript
- () is available in all current browsers

```
/* Dim card with “sold out” notice */
.card:has(.sold-out) {
  opacity: 0.6;

  & button { pointer-events: none; }
}

/* Card with image gets two columns */
.card:has(> img) {
  grid-template-columns: 6rem 1fr;
}
```

Form feedback only after input

- applies only after input
- () highlights the whole form
- The submit hint appears without a script

```
input:user-invalid {
  border-color: var(--color-danger);
}

form:has(:user-invalid) .submit-hint {
  display: block;
}

form:has(:invalid) button[type="submit"] {
  opacity: 0.7;
}
```

When does the media query stay?

- For page layout with header and sidebar
- For user settings such as prefers-color-scheme
- For everything else, the component asks its container

Modules and data from the network

ES Modules run in the browser without a build · Every request needs an end · Streams show data before everything is there

Does every page need a bundler?

- All current browsers load ES Modules natively
- HTTP/2 loads many small files in parallel
- A build remains an optimization, not a requirement

A module script starts the graph

- Modules run in Strict Mode and have their own scope
- They run deferred automatically
- Relative paths need ./ and the file extension

```
// index.html: <script type="module" src="./main.js"></script>
// main.js
import { addItem, totalCents } from './lib/cart.js'
import { formatCents } from './lib/format.js'

let cart = []
cart = addItem(cart, { productId: 'p-1', priceCents: 8990 })
document.querySelector('#total').textContent = formatCents(totalCents(cart))
```

An import map replaces path aliases

- Bare specifiers like shop/ get an address
- Prefixes with a slash map whole folders
- The map comes before the first module

```
<script type="importmap">
  {
    "imports": {
      "shop/": "./lib/",
      "lit": "https://cdn.jsdelivr.net/npm/lit@3/+esm"
    }
  }
</script>
<script type="module">
  import { formatCents } from 'shop/format.js'
  console.log(formatCents(8990))
</script>
```

modulepreload shortens the waterfall

- The browser discovers imports only while parsing
- modulepreload loads important modules right away
- DevTools show the waterfall in the Network tab

```
<link rel="modulepreload" href="./lib/cart.js">
<link rel="modulepreload" href="./lib/format.js">
<script type="module" src="./main.js"></script>
```

When is a build still worth it?

- When TypeScript, JSX, or Svelte must be translated
- When metrics show too many requests
- Vite then adds what the browser already can do

fetch throws less often than you think

- A status 404 or 500 is a success for fetch
- Only network errors and aborts throw
- You check response.ok yourself

```
export class HttpError extends Error {
  constructor(status, url) {
    super(`Request an ${url} scheiterte mit Status ${status}`)
    this.name = 'HttpError'
    this.status = status
  }
}

const response = await fetch('/api/products')
if (!response.ok) throw new HttpError(response.status, response.url)
const products = await response.json()
```

Without a timeout, fetch waits indefinitely

- A hanging server blocks the interface
- AbortSignal.timeout sets a fixed limit
- The abort throws a TimeoutError

```
export async function loadProducts({ signal, timeoutMs = 5000 } = {}) {
  const signals = [AbortSignal.timeout(timeoutMs)]
  if (signal) signals.push(signal)

  const response = await fetch('/api/products', {
    signal: AbortSignal.any(signals),
    headers: { accept: 'application/json' },
  })
  if (!response.ok) throw new HttpError(response.status, response.url)
  return response.json()
}
```

The new search aborts the old one

- Each input starts a new search
- One AbortController per search
- Older responses must not overtake new ones

```
export function createSearch(fetchFn = fetch) {
  let controller
  return async function search(term) {
    controller?.abort()
    controller = new AbortController()
    const url = `/api/products?q=${encodeURIComponent(term)}`
    const response = await fetchFn(url, { signal: controller.signal })
    if (!response.ok) throw new HttpError(response.status, url)
    return response.json()
  }
}
```

An abort is not an error

- AbortError means intentionally ended
- TimeoutError means too slow, show a hint
- Report everything else with a next step

```
input.addEventListener('input', async () => {
  try {
    render(await search(input.value))
  } catch (error) {
    if (error.name === 'AbortError') return
    showMessage(error.name === 'TimeoutError'
      ? 'Die Suche dauert zu lange. Versuch es gleich noch einmal.'
      : 'Die Produkte konnten nicht geladen werden. Lad die Seite neu.')
  }
})
```

Waiting until everything is there costs time

- response.json() waits for the last byte
- A stream delivers data in chunks
- The first products appear on the page right away

NDJSON is JSON per line

- Each line is a complete JSON object
- The server writes as soon as a record is ready
- A line break separates the records

```
{"id":"p-1","name":"Akkuschrauber","priceCents":8990,"stock":12}
{"id":"p-2","name":"Bitset 32-teilig","priceCents":1990,"stock":40}
{"id":"p-3","name":"Wasserwaage 60 cm","priceCents":2490,"stock":0}
```

A line buffer puts the chunks together

- TextDecoderStream turns bytes into text
- The buffer keeps the started line
- After the last chunk, the rest is processed

```
export async function* readNdjson(response) {
  const reader = response.body.pipeThrough(new TextDecoderStream()).getReader()
  let buffer = ''
  try {
    while (true) {
      const { value, done } = await reader.read()
      if (done) break
      buffer += value
      const lines = buffer.split('\n')
      buffer = lines.pop() ?? ''
      for (const line of lines) if (line.trim()) yield JSON.parse(line)
    }
    if (buffer.trim()) yield JSON.parse(buffer)
  } finally {
    reader.releaseLock()
  }
}
```

for await renders each product right away

- The async generator yields one object per line
- Abort with the same AbortSignal as the request
- Direct async iteration over streams is now baseline

```
const controller = new AbortController()
const response = await fetch('/api/products/stream', {
  signal: controller.signal,
})
if (!response.ok) throw new HttpError(response.status, response.url)

for await (const product of readNdjson(response)) {
  list.append(renderProduct(product))
}
```

Web Components and View Transitions

Custom Elements are components without a framework · Shadow DOM encapsulates, slots open with intent · View Transitions animate state changes

Custom HTML elements for the shop

- One hyphenated name, one class, one define
- The browser calls lifecycle callbacks
- Works in every framework and without one

```
<article>
  <h2>Akkuschrauber</h2>
  <shop-price cents="8990"></shop-price>
  <add-to-cart product-id="p-1"></add-to-cart>
</article>
```

Attributes are the inputs

- observedAttributes declares what is observed
- attributeChangedCallback renders on every change
- connectedCallback renders when attached

```
import { formatCents } from './shop.js'

export class ShopPrice extends HTMLElement {
  static observedAttributes = ['cents']

  connectedCallback() {
    this.render()
  }

  attributeChangedCallback() {
    this.render()
  }

  render() {
    this.textContent = formatCents(Number(this.getAttribute('cents') ?? 0))
  }
}

customElements.define('shop-price', ShopPrice)
```

Events are the outputs

- CustomEvent with detail carries the data
- bubbles lets the event move upward
- composed lets it leave the Shadow DOM

```
export class AddToCart extends HTMLElement {
  connectedCallback() {
    this.innerHTML = '<button type="button">In den Warenkorb</button>'
    this.querySelector('button').addEventListener('click', () => {
      this.dispatchEvent(new CustomEvent('cart:add', {
        detail: { productId: this.getAttribute('product-id') },
        bubbles: true,
        composed: true,
      }))
    })
  }
}

customElements.define('add-to-cart', AddToCart)
```

Cleanup is part of the lifecycle

- connectedCallback can run more than once
- disconnectedCallback removes listeners and timers
- An AbortController cleans up everything at once

```
export class CartCount extends HTMLElement {
  #controller

  connectedCallback() {
    this.#controller = new AbortController()
    document.addEventListener('cart:add', () => {
      this.textContent = String(Number(this.textContent || 0) + 1)
    }, { signal: this.#controller.signal })
  }

  disconnectedCallback() {
    this.#controller?.abort()
  }
}
```

When is a Custom Element worth it?

- For components that run in multiple frameworks
- For design systems with a long life span
- For islands in server-side rendered pages

Why does page CSS break my card?

- Global selectors also hit components
- Internal classes collide with outside classes
- Shadow DOM draws a boundary around the inside

Template and Shadow Root

- The template is parsed but not rendered
- attachShadow attaches a Shadow Root
- Styles in the Shadow Root apply only there

```
const template = document.createElement('template')
template.innerHTML = `
  <style>
    :host { display: grid; gap: var(--space-2, 0.5rem); }
    button { background: var(--color-accent, #1d4ed8); color: white; }
  </style>
  <slot name="name"></slot>
  <slot name="price"></slot>
  <slot></slot>
  <button part="buy" type="button">In den Warenkorb</button>`

export class ProductCard extends HTMLElement {
  constructor() {
    super()
    this.attachShadow({ mode: 'open' })
      .append(template.content.cloneNode(true))
  }
}
```

Slots bring the content in

- Elements with a slot attribute land in the matching slot
- Everything else lands in the default slot
- Slotted content stays in the Light DOM

```
<product-card>
  <h2 slot="name">Akkuschrauber</h2>
  <shop-price slot="price" cents="8990"></shop-price>
  <p>Mit zwei Akkus und Koffer.</p>
</product-card>
```

Parts and Custom Properties are the design API

- Custom Properties set colors and spacing
-
- Everything else stays private

```
product-card {
  --color-accent: #b91c1c;
}

product-card::part(buy) {
  border-radius: 999px;
  padding-inline: 1.5rem;
}

product-card::part(buy):hover {
  background: #7f1d1d;
}
```

Declarative Shadow DOM renders without JavaScript

- A template with shadowrootmode in the HTML
- The browser builds the Shadow Root while parsing
- The Custom Element takes it over later

```
<product-card>
  <template shadowrootmode="open">
    <style>:host { display: grid; }</style>
    <slot name="name"></slot>
    <button part="buy" type="button">In den Warenkorb</button>
  </template>
  <h2 slot="name">Akkuschrauber</h2>
</product-card>
```

A jump confuses; motion explains

- Animation shows where an element came from
- Before, this needed clones, measurements, and a library
- Today the browser does it with one call

startViewTransition creates before and after

- The browser captures the old state
- Your callback changes the DOM
- Between them, it cross-fades smoothly

```
export function transition(update) {
  const reduced = matchMedia('(prefers-reduced-motion: reduce)').matches
  if (!document.startViewTransition || reduced) {
    return Promise.resolve(update())
  }
  return document.startViewTransition(update).updateCallbackDone
}

await transition(() => {
  list.hidden = true
  detail.hidden = false
})
```

view-transition-name connects two elements

- Same name before and after means same element
- The browser animates position and size
- Each name may appear only once on the page

```
.card img {
  view-transition-name: var(--vt-name);
}

.detail img {
  view-transition-name: var(--vt-name);
}

::view-transition-group(*) {
  animation-duration: 250ms;
  animation-timing-function: ease-out;
}
```

Between pages only as an extra

- @view-transition animates real page changes
- Chrome and Safari support it; Firefox does not yet
- Without support, the page loads normally

```
@view-transition {
  navigation: auto;
}

@media (prefers-reduced-motion: reduce) {
  @view-transition {
    navigation: none;
  }
}
```

Offline and performance

Offline starts with a cache strategy · Measure Core Web Vitals instead of guessing

Which storage fits which data?

- localStorage for small, synchronous values
- IndexedDB for large and structured data
- Cache API for complete responses

Saved data can be broken

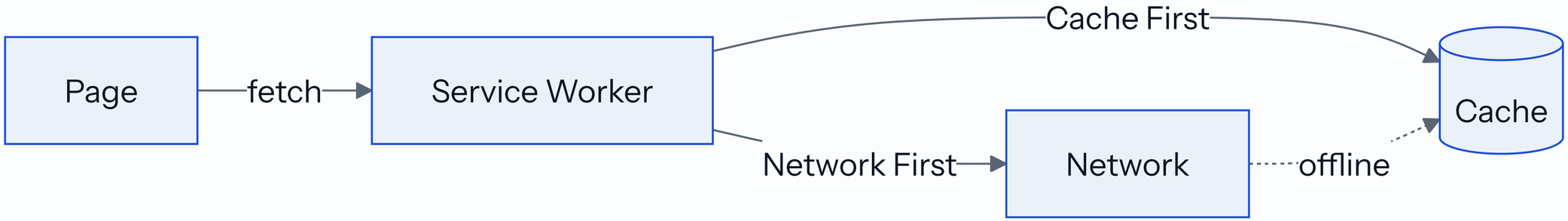
- A version prevents old formats
- Parse errors lead to an empty cart
- Quota errors must not stop the page

```
const VERSION = 2

export function createCartStorage(storage, key = 'cart') {
  const load = () => {
    try {
      const saved = JSON.parse(storage.getItem(key) ?? 'null')
      return saved?.version === VERSION ? saved.items : []
    } catch { return [] }
  }
  const save = (items) => {
    try {
      storage.setItem(key, JSON.stringify({ version: VERSION, items }))
      return true
    } catch { return false }
  }
  return { load, save }
}
```

A Service Worker sits between page and network

- It intercepts every request from its pages
- It runs separately from the page, without a DOM
- It needs HTTPS, only localhost is exempt



Two strategies cover the shop

- Cache First for HTML, CSS, scripts, and images
- Network First for products and prices
- The cache is the fallback when the network is missing

```
export async function cacheFirst(request, cache, fetchFn) {
  const cached = await cache.match(request)
  if (cached) return cached
  const response = await fetchFn(request)
  if (response.ok) await cache.put(request, response.clone())
  return response
}

export async function networkFirst(request, cache, fetchFn) {
  try {
    const response = await fetchFn(request)
    if (response.ok) await cache.put(request, response.clone())
    return response
  } catch (error) {
    const cached = await cache.match(request)
    if (cached) return cached
    throw error
  }
}
```

Register, update, clean up

- Register after the load event
- One new cache name per release
- activate deletes old caches

```
// sw.js
import { cacheFirst, networkFirst } from './strategies.js'
const CACHE = 'shop-v3'

self.addEventListener('activate', (event) => {
  event.waitUntil(caches.keys().then((keys) => Promise.all(
    keys.filter((key) => key !== CACHE).map((key) => caches.delete(key)),
  )))
})

self.addEventListener('fetch', (event) => {
  const url = new URL(event.request.url)
  const strategy = url.pathname.startsWith('/api/') ? networkFirst : cacheFirst
  event.respondWith(
    caches.open(CACHE).then((cache) => strategy(event.request, cache, fetch)),
  )
})
```

Three numbers for the feel of speed

- LCP: When is the largest content shown?
- INP: How quickly does the page respond to input?
- CLS: How much does the layout jump?

Good means: 75 percent of visits

- LCP is good up to 2.5 s, poor from 4 s
- INP is good up to 200 ms, poor from 500 ms
- CLS is good up to 0.1, poor from 0.25

```
const thresholds = {
  LCP: [2500, 4000],
  INP: [200, 500],
  CLS: [0.1, 0.25],
}

export function rate(name, value) {
  const [good, poor] = thresholds[name]
  if (value <= good) return 'good'
  if (value <= poor) return 'needs-improvement'
  return 'poor'
}
```

Measure in the field with PerformanceObserver

- The browser reports entries for LCP and interactions
- buffered also returns entries from before startup
- You send the last value when the user leaves the page

```
export function observeLcp(report) {
  const observer = new PerformanceObserver((list) => {
    const last = list.getEntries().at(-1)
    if (last) report({ name: 'LCP', value: last.startTime })
  })
  observer.observe({ type: 'largest-contentful-paint', buffered: true })
  return () => observer.disconnect()
}

observeLcp(({ name, value }) => {
  navigator.sendBeacon('/api/vitals', JSON.stringify({ name, value }))
})
```

Images decide LCP and CLS

- width and height reserve the space
- fetchpriority high loads the hero image first
- loading lazy defers images farther down

```
<img src="hero.webp" alt="Werkbank mit Akkuschrauber"
     width="1200" height="600" fetchpriority="high">

<ul class="products">
  <li>
    <img src="p-2.webp" alt="Bitset 32-teilig"
         width="320" height="240" loading="lazy">
  </li>
</ul>
```

Fast responses need short tasks

- Long tasks block every click
- Respond visibly first, then calculate
- DevTools shows long tasks in the Performance tab

```
button.addEventListener('click', async () => {
  button.disabled = true
  status.textContent = 'Wird in den Warenkorb gelegt …'

  await new Promise((resolve) => setTimeout(resolve, 0))
  recalculateRecommendations(cart)
})
```

Questions?

Mike Bild · mike.bild@gmail.com · mikebild.dev

Workshops für Teams, die Architekturentscheidungen selbst treffen