

# Rendering Patterns

sposoby procesowania  
i wyświetlania aplikacji webowych



**Michał Kidawa**

Senior full-stack developer@Chooose  
[github/mkidawa](https://github.com/mkidawa)

```
const org = interbyOrg ? study.lead_organization === interbyOrg : C  
const status = filterByStatus ? study.status === filterByStatus : true  
(matchStatus) {
```

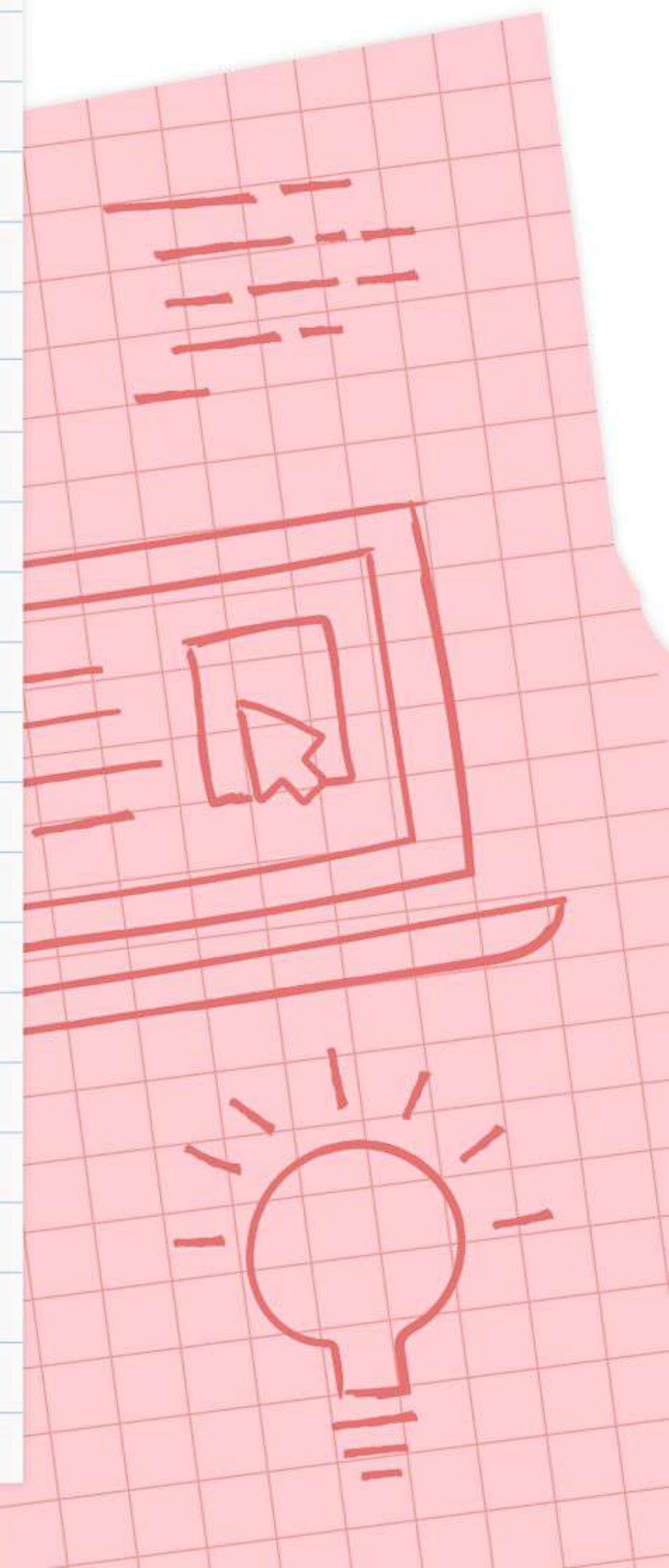
```
function filterStudies({ studies, filterByOrg :  
  studies.filter(study
```



# Rendering pattern

Czyli właściwie co?

Korzystając z jakich mechanizmów (i narzędzi) chcemy serwować użytkownikom zawartość strony (HTML + CSS + JS).

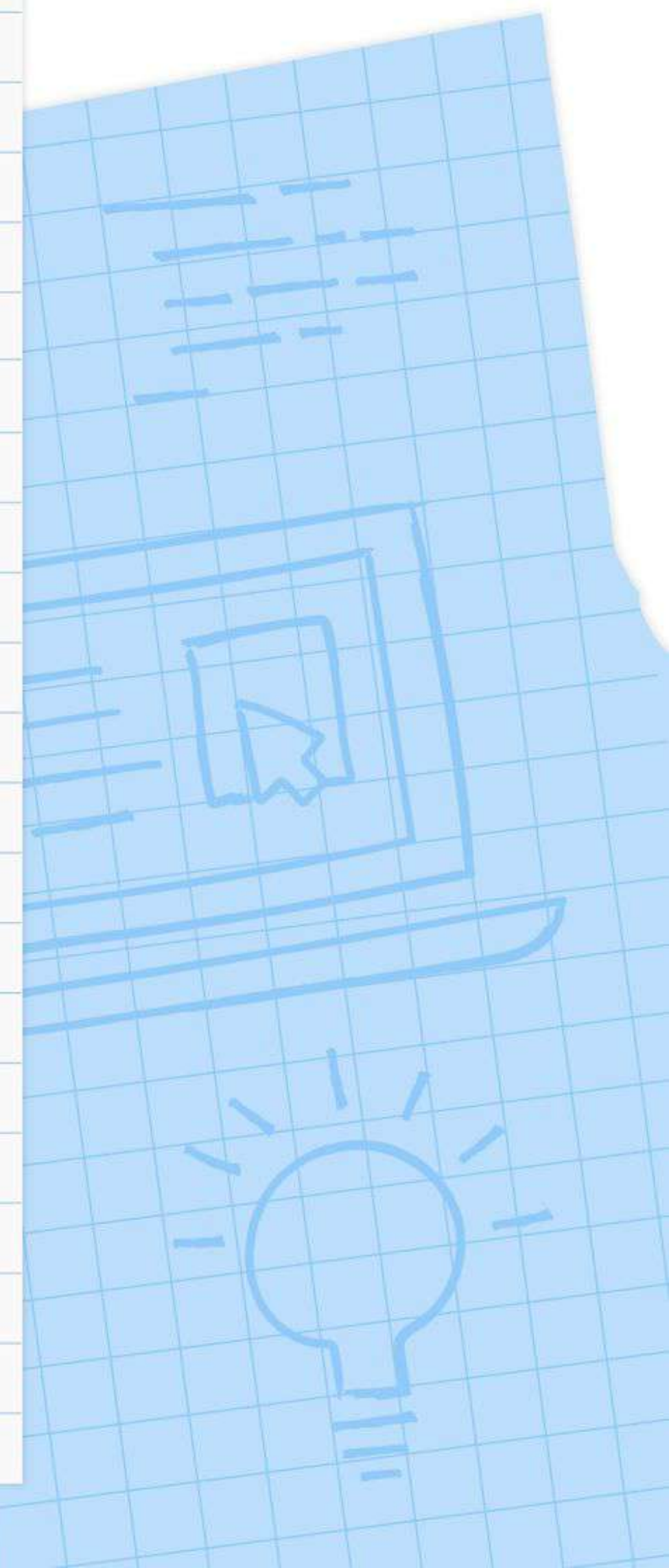




# Punkt startowy

## Rendering patterns

- prezentacja JS-centriczna
- zagadnienie wysokopoziomowe
- decyzja o doborze często wpływa na sposób tworzenia całej aplikacji
- zmiana podejścia w trakcie cyklu życia projektu absorbuje czas oraz generuje koszty





# Znaczenie wyboru

UX oraz DX

Jak użytkownik końcowy będzie wchodził w interakcję z naszym produktem?

Czy nam będzie się chciało wstawać rano do pracy?

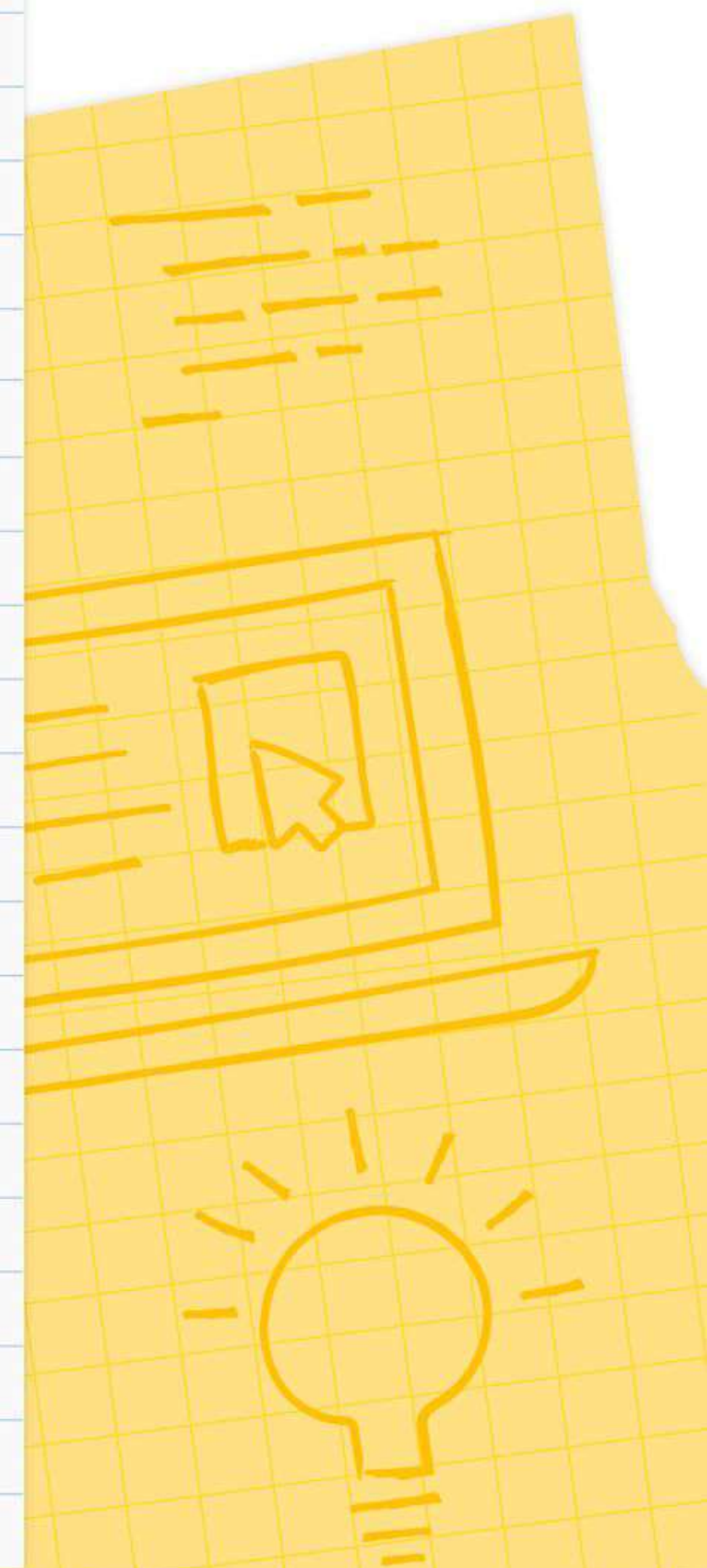




# User Experience

## Web Vitals

Użytkowniko-centriczne metryki pozwalające określić jak dobre (i szybkie) doświadczenia będą płynęły z obcowania z naszą aplikacją.





### **TTFB - Time to First Byte**

Czas potrzebny na pojawienie się pierwszego bajtu strony w odpowiedzi na żądanie użytkownika

### **TTI - Time To Interactive**

Czas od ładowania strony do stanu umożliwiającego użytkownikowi na interakcję z nią

### **FCP - First Contentful Paint**

Czas potrzebny na pokazanie (wyrenderowanie) pierwszego elementu treści po wejściu na nią

### **CLS - Cumulative Layout Shift**

Metryka związana ze zmianami obserwowanymi na stronie po załadowaniu

### **LCP - Largest Contentful Paint**

Czas potrzebny na załadowanie i wyrenderowanie głównej zawartości strony

### **FID - First Input Delay**

Czas od możliwości interakcji do stanu umożliwiającego emisję eventów przez handlers

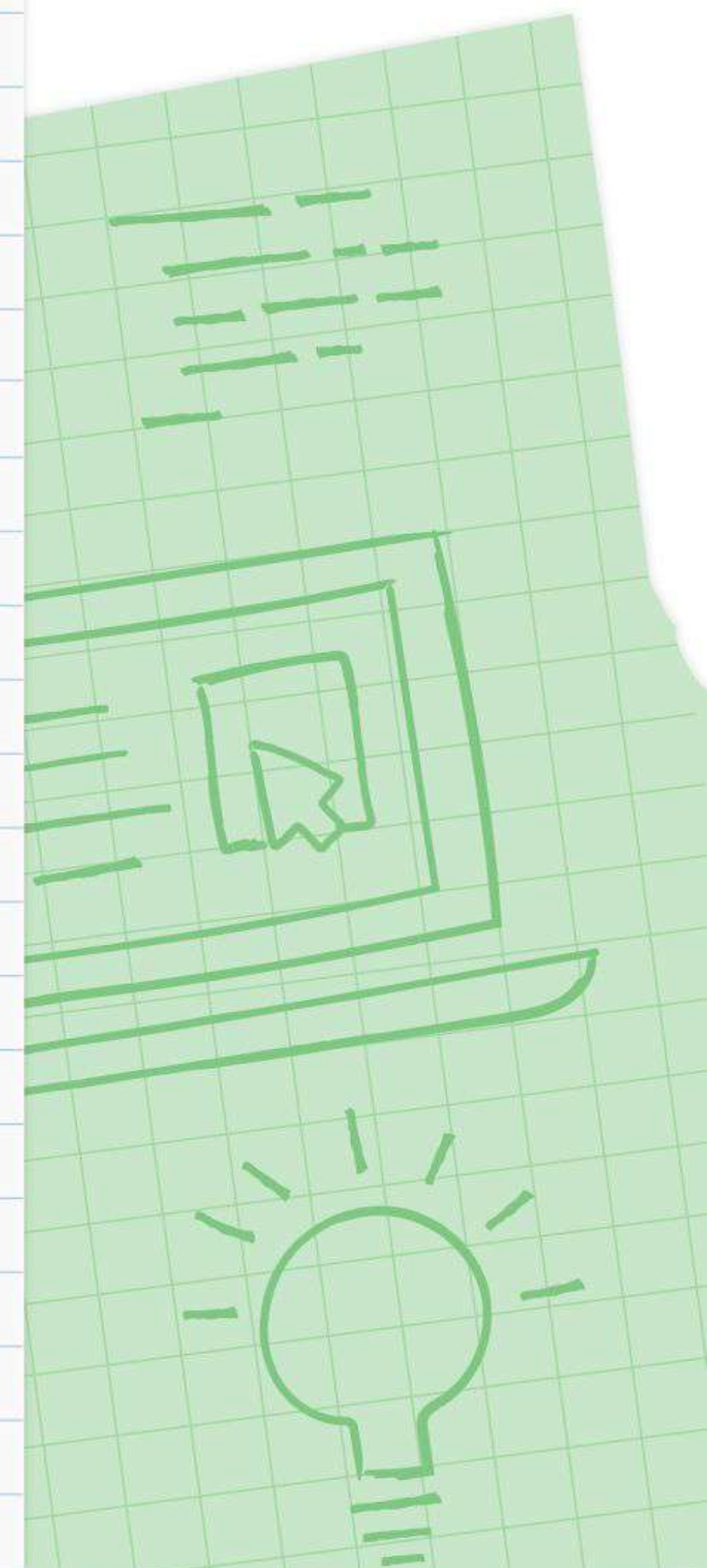




# Developer Experience

HR zaprasza na rozmowę, faktura z chmury przyszła

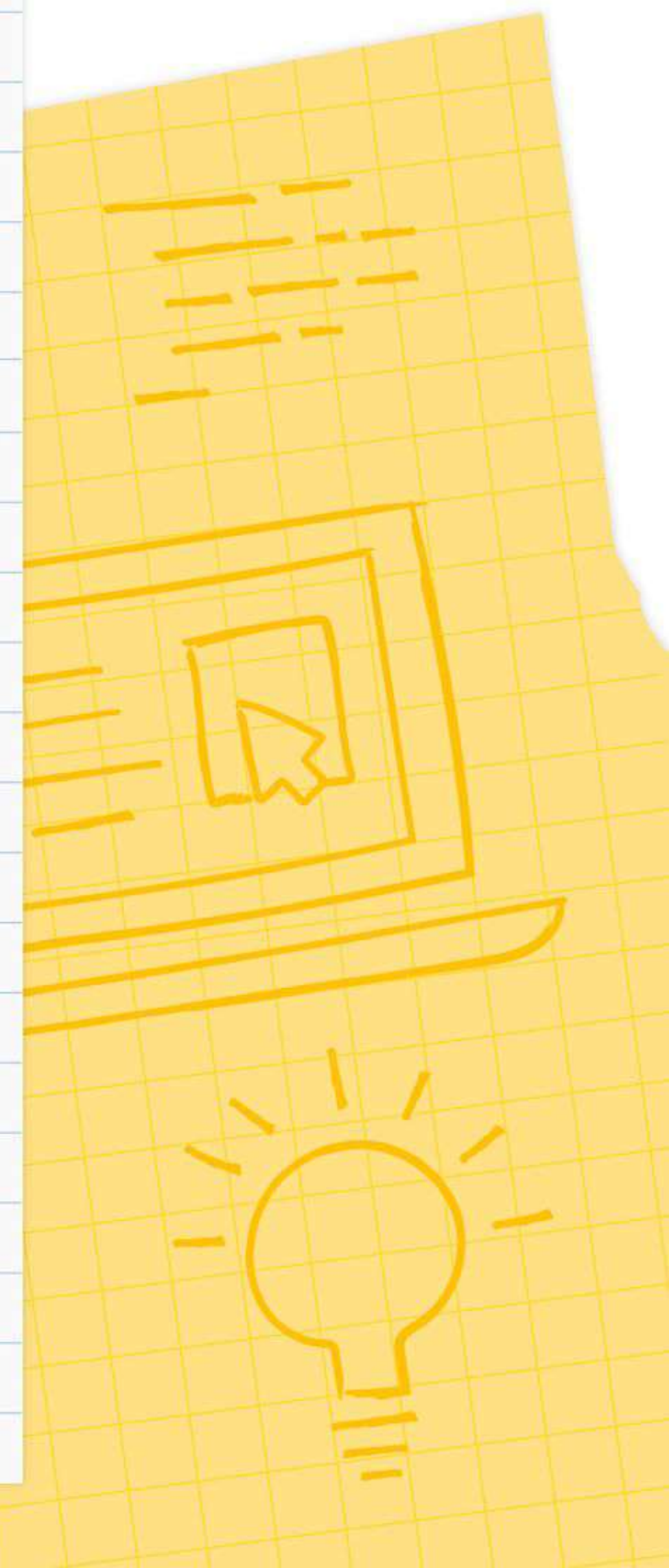
- czas potrzebny na wykonanie buildu
- koszty serwerów
- zawartość dynamiczna
- skalowanie
- QoS





# Pojęcie “hydration”

Pod pojęciem hydratacji (ang. hydration) kryję się “wzbogacenie” wyrenderowanego dokumentu HTML o możliwość interakcji i zachowania udostępniane przez JavaScript.





### **1) Renderowanie treści strony**

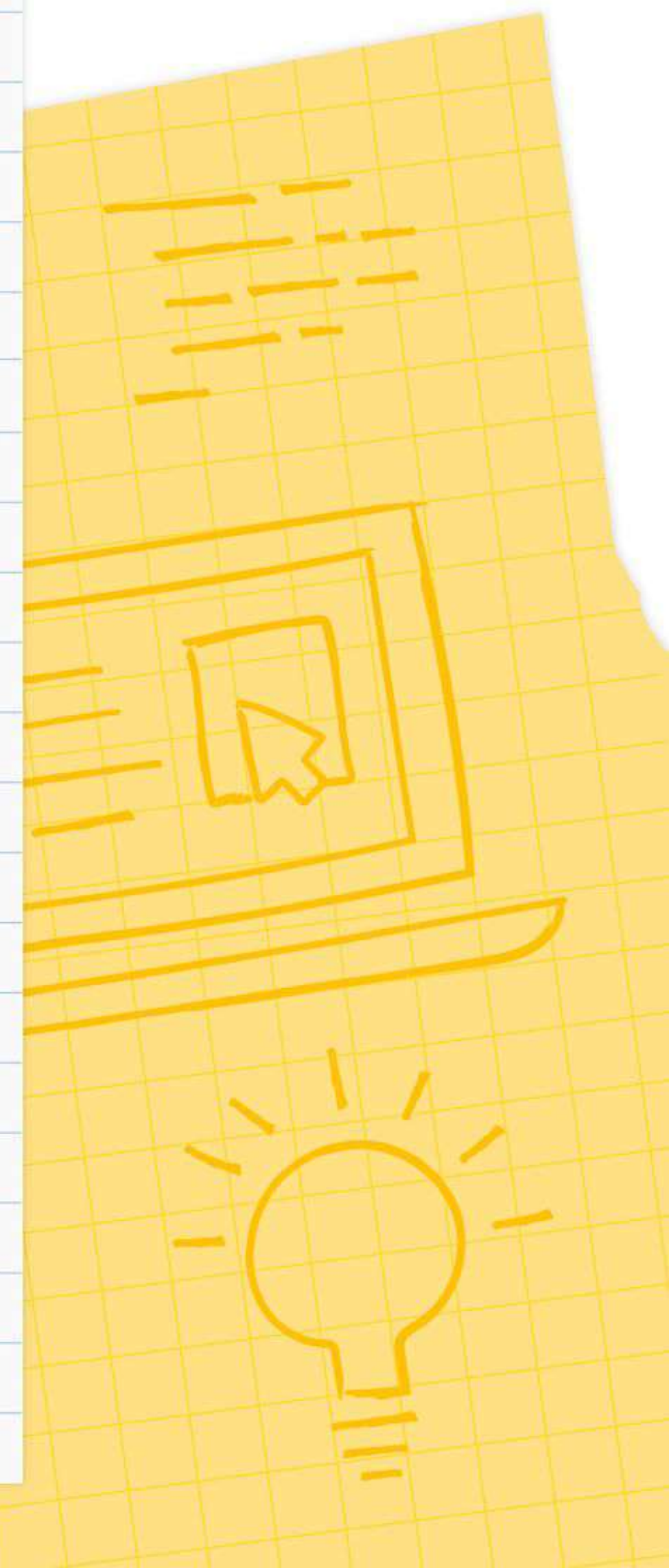
Pierwsze załadowanie strony, następuje wyrenderowanie statycznego HTML

### **2) Hydration**

Załadowanie kodu JavaScript związanego ze stroną, a następnie “hydration”, czyli dołączenie do statycznej strony np. event listenerów (naciśnięcie guzika), zarządzanie stanem i innych funkcji umożliwiających interakcję

### **3) Faza interakcji**

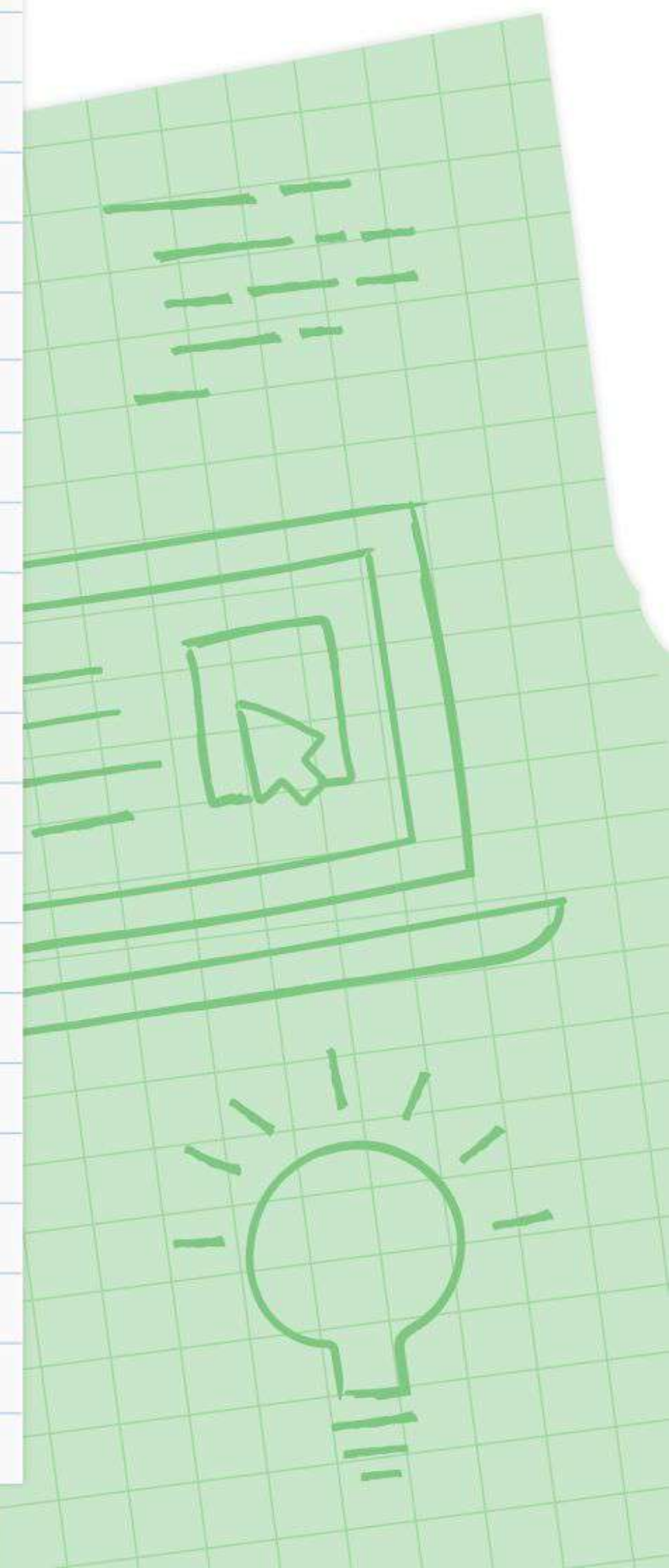
Po fazie hydration strona jest gotowa na interakcję z użytkownikiem - nasze działania wpływają na UI



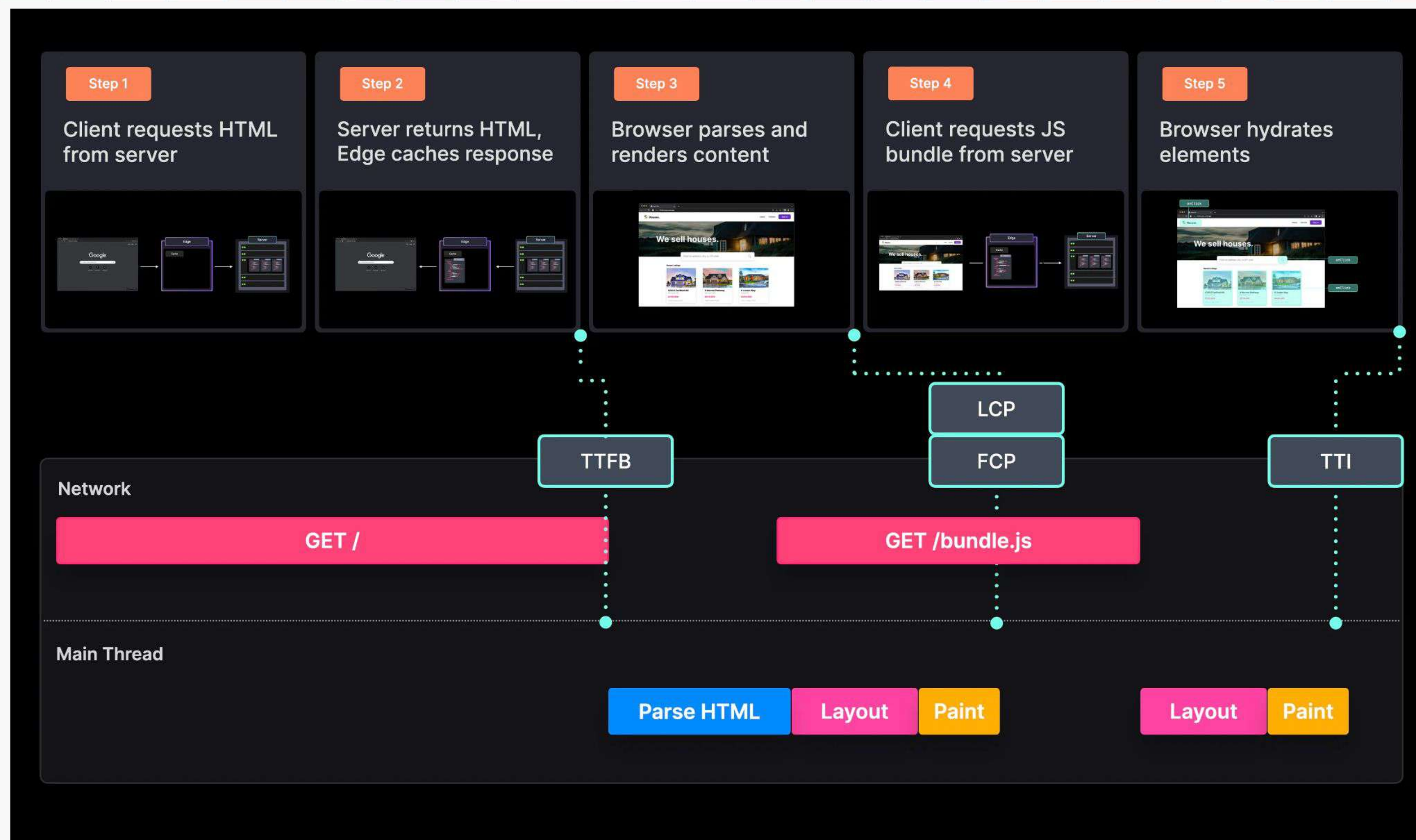


# Statyczne renderowanie

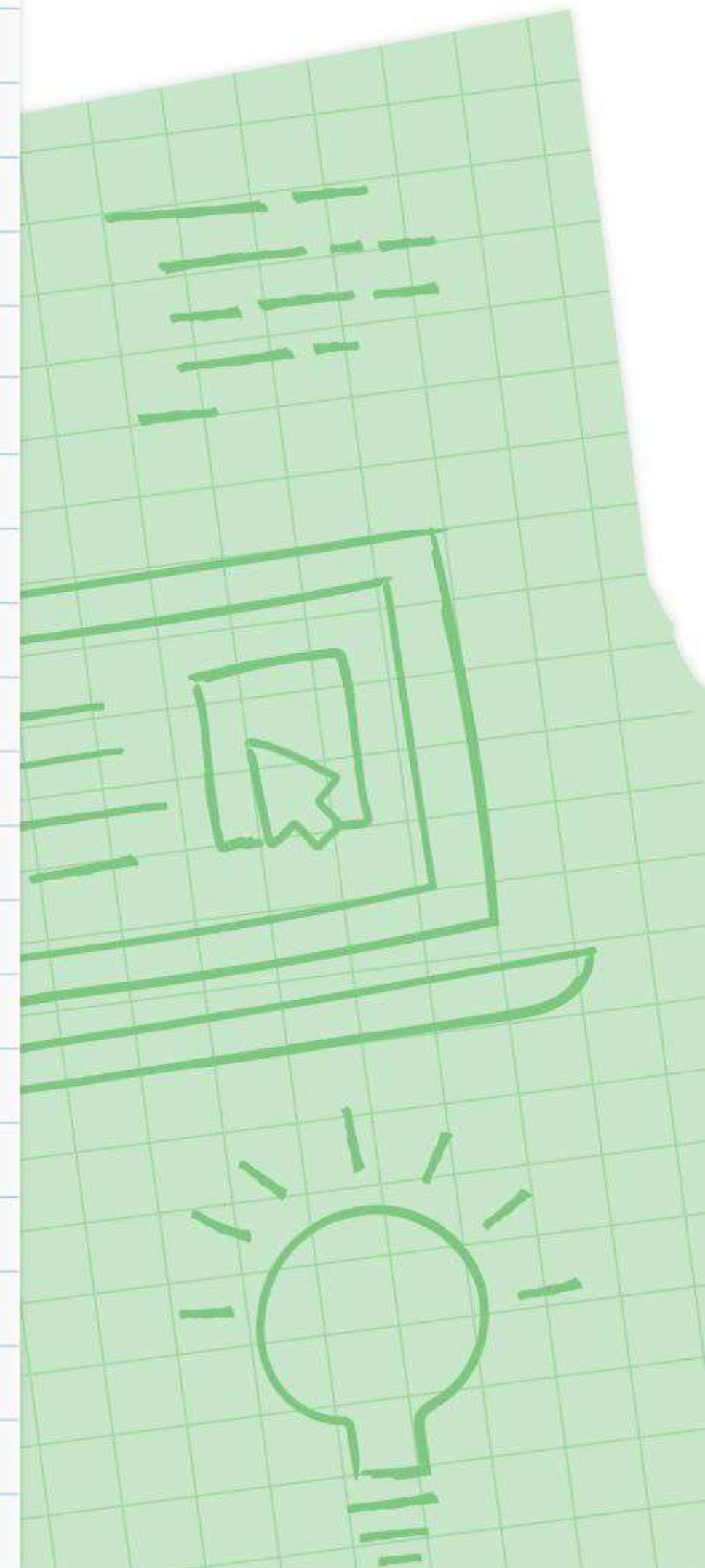
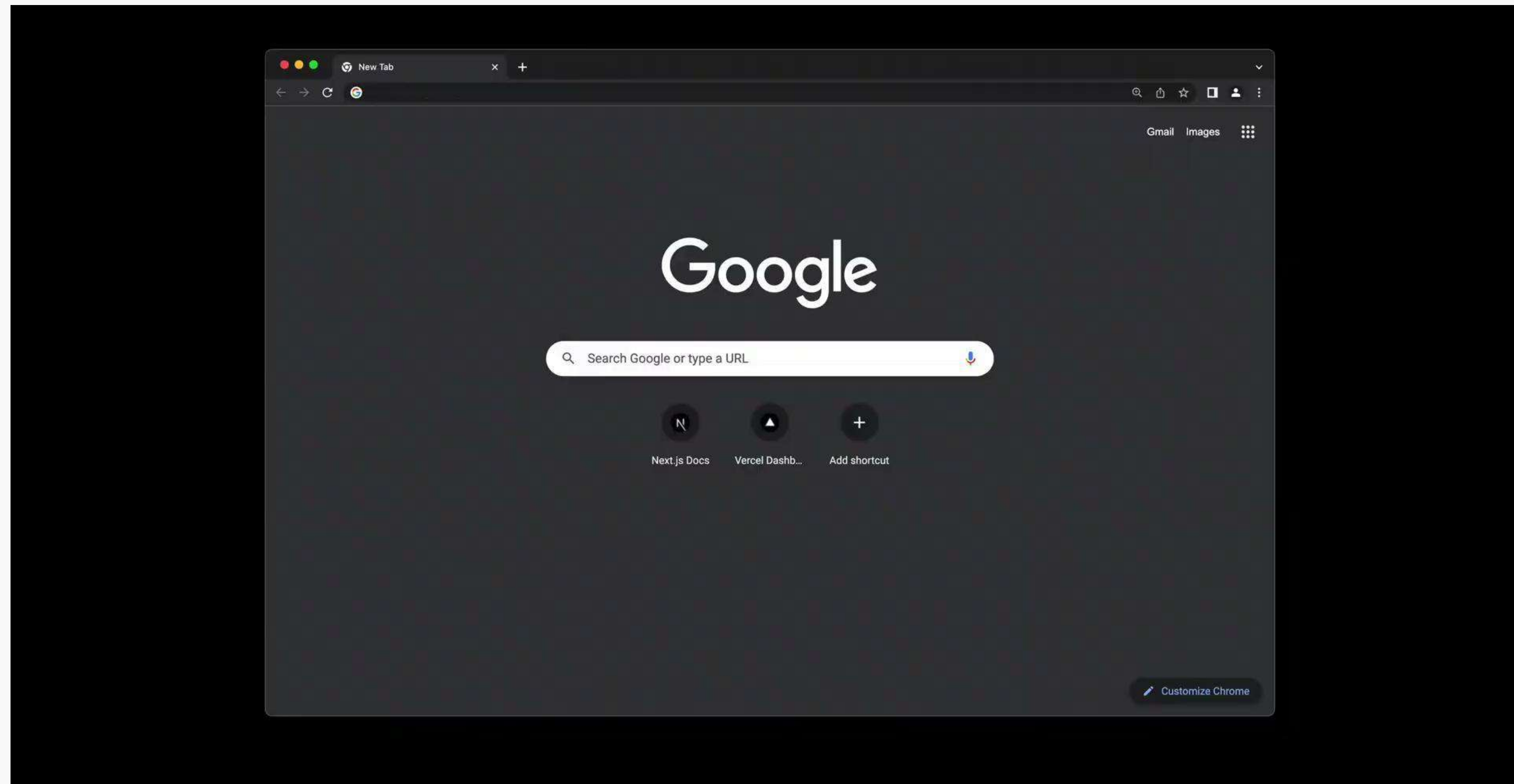
HTML całej strony jest dostępny w momencie wrzucenia na serwer, nie zmienia się. Zawartość jest statyczna, do tego łatwo można ją cachować. Dwa kluczowe słowa - prostota i szybkość.









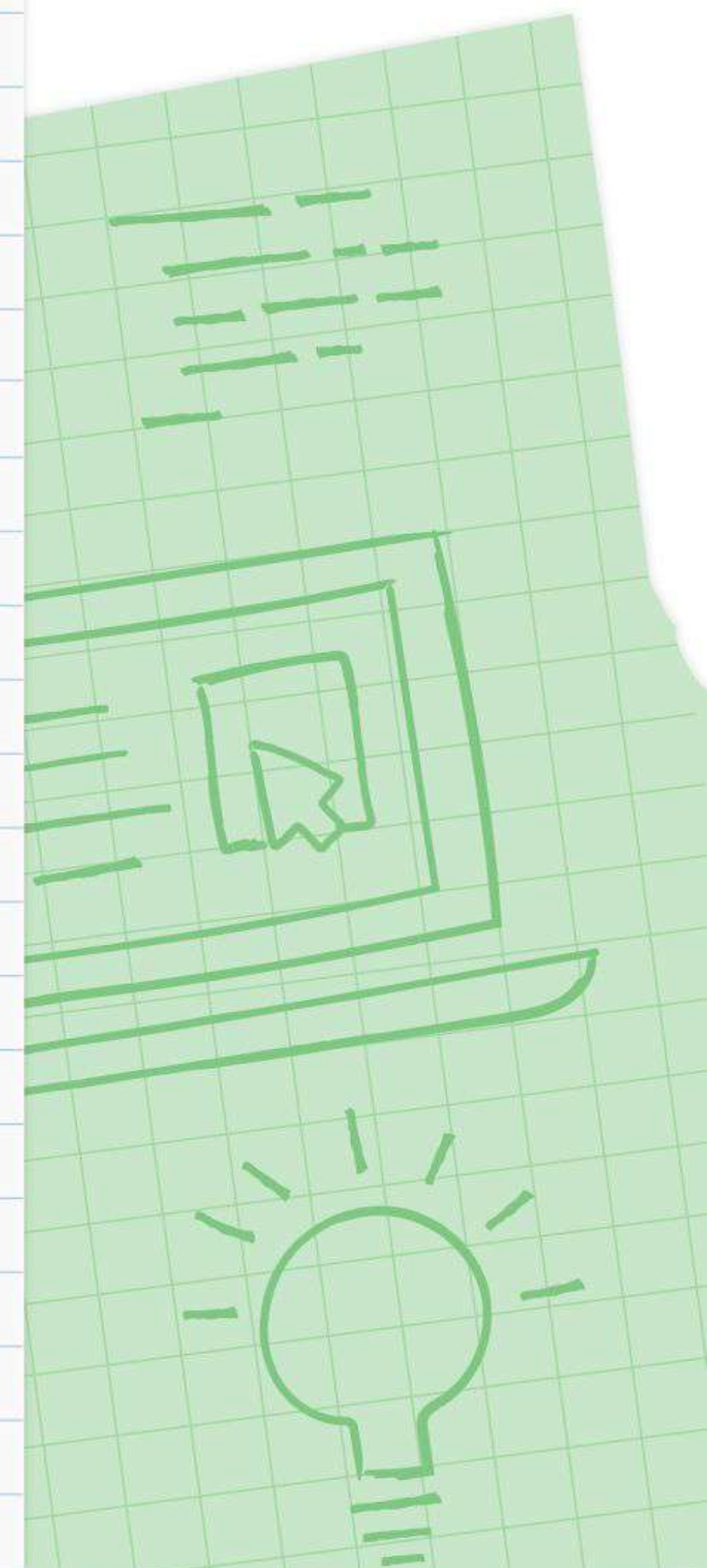




# Statyczne renderowanie

## Plusy i minusy

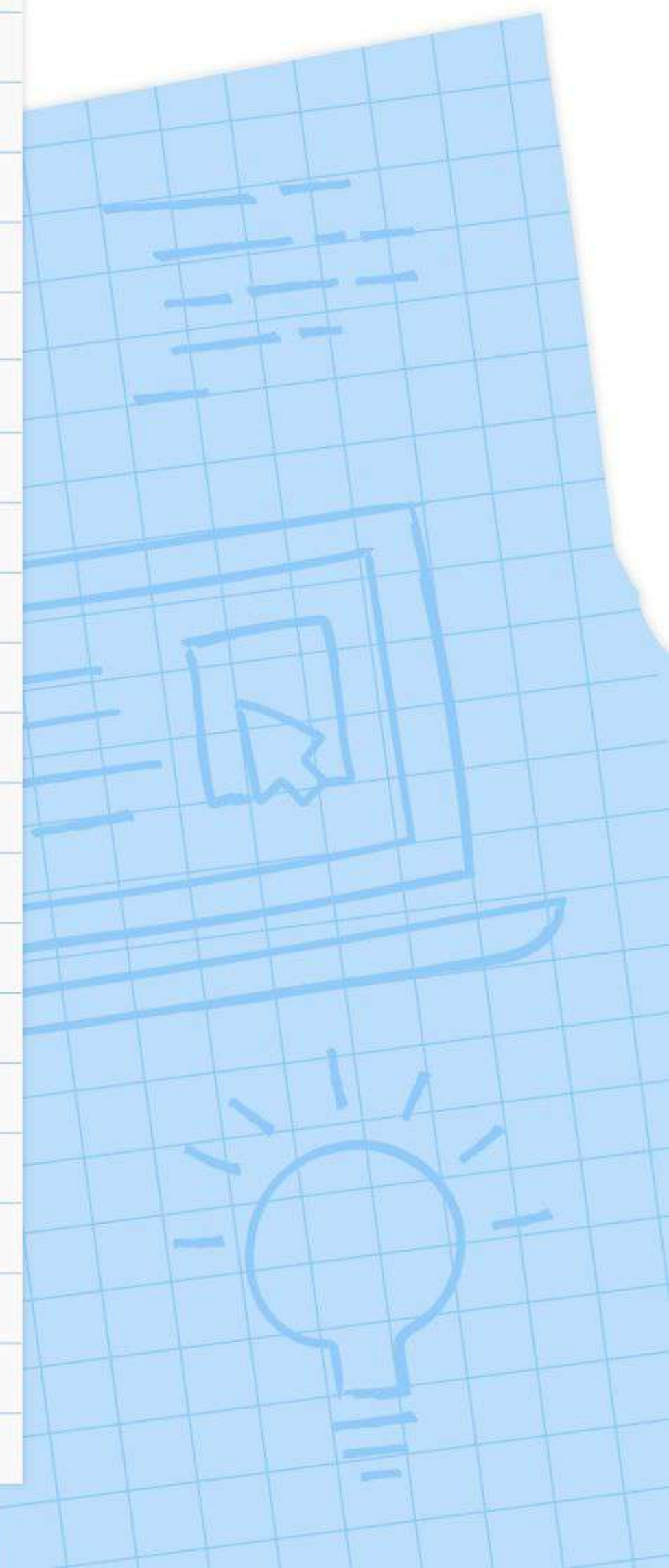
- + krótki czas ładowania, cała zawartość strony jest gotowa
- + idealna do prostych stron ze statyczną treścią
- + możliwość cachowania całej strony za pomocą CDN aby jeszcze bardziej zwiększyć wydajność
- nie nadaje się do projektów z dynamicznym, lub często zmieniającą się treścią
- wymaga ponownego wgrania na serwer w celu zaaplikowania zmian
- ograniczone możliwości w sferze interakcji



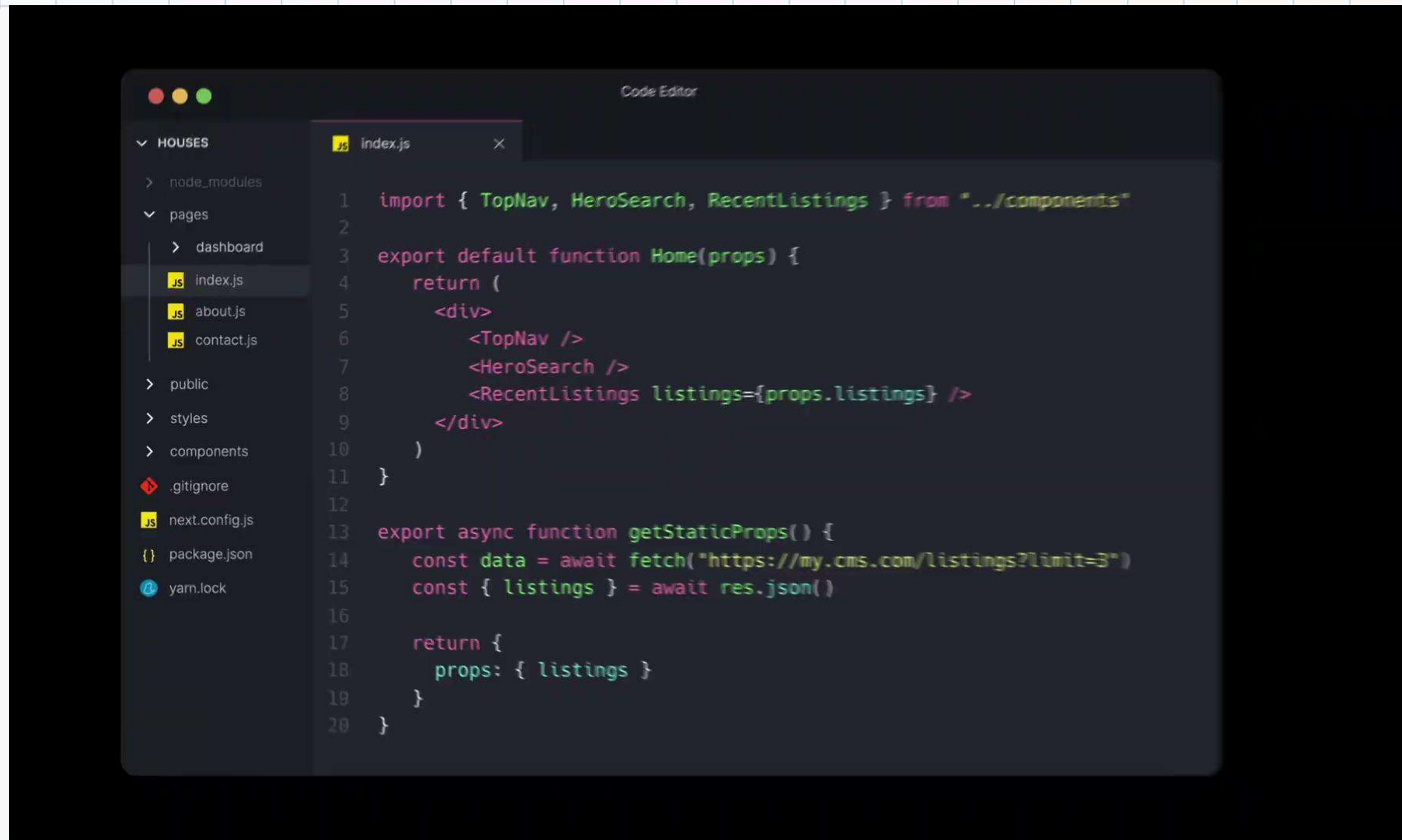


# Static Site Generation

Strony są tworzone w trakcie buildu, następuje pre-renderowanie. Content tworzony (umieszczany na stronie) jest raz i nie zmienia się do następnego buildu.



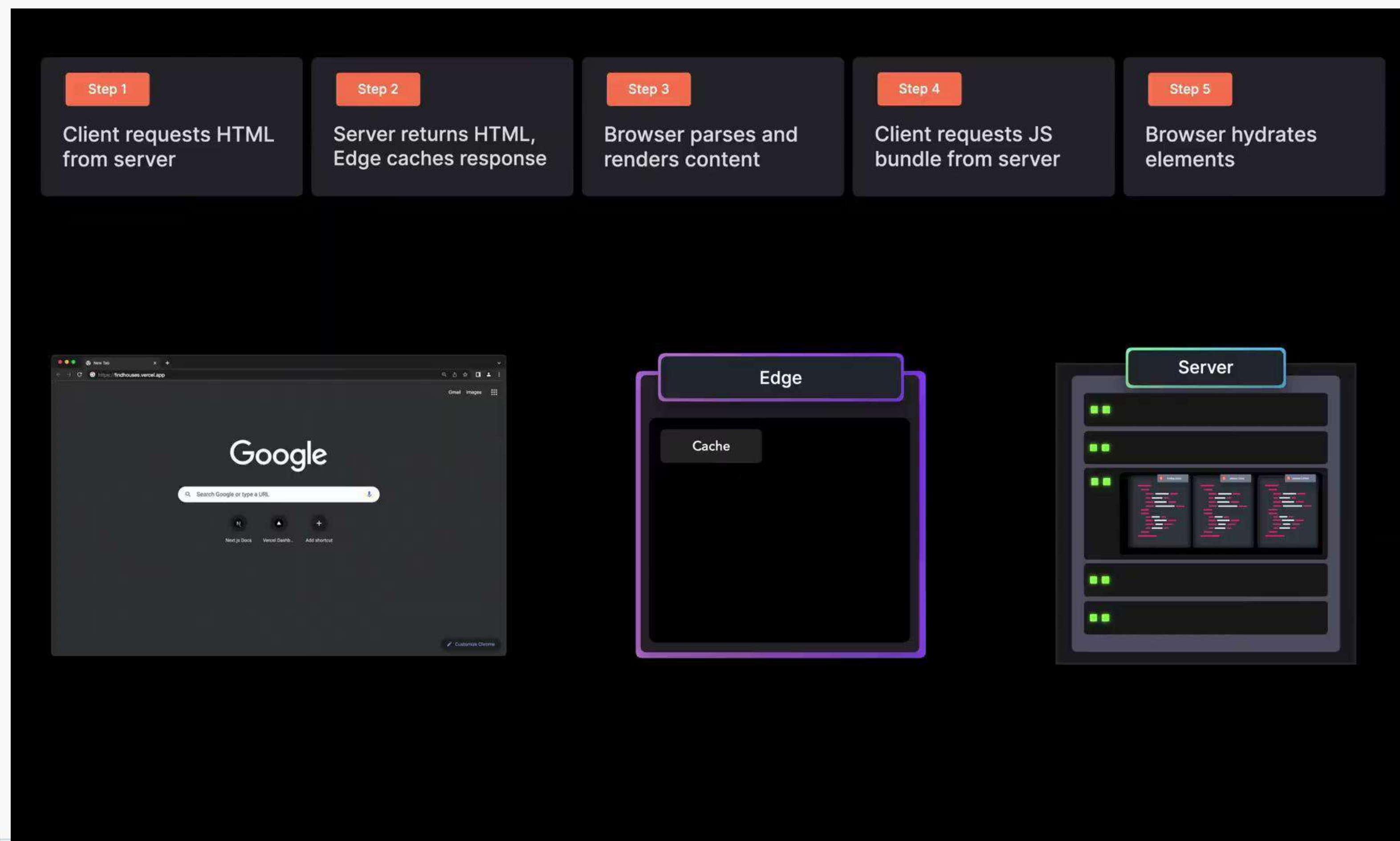




The screenshot shows a code editor window titled "Code Editor" with a file explorer on the left. The file explorer shows a project structure with folders like "node\_modules", "pages", "public", "styles", and "components". The "pages" folder is expanded, showing "index.js", "about.js", and "contact.js". The "index.js" file is selected and its content is displayed in the editor. The code is written in JavaScript and uses JSX to render a page layout. It imports components from a local directory and fetches data from an external API.

```
1 import { TopNav, HeroSearch, RecentListings } from "../components"
2
3 export default function Home(props) {
4   return (
5     <div>
6       <TopNav />
7       <HeroSearch />
8       <RecentListings listings={props.listings} />
9     </div>
10  )
11 }
12
13 export async function getStaticProps() {
14   const data = await fetch("https://my.cms.com/listings?limit=3")
15   const { listings } = await res.json()
16
17   return {
18     props: { listings }
19   }
20 }
```



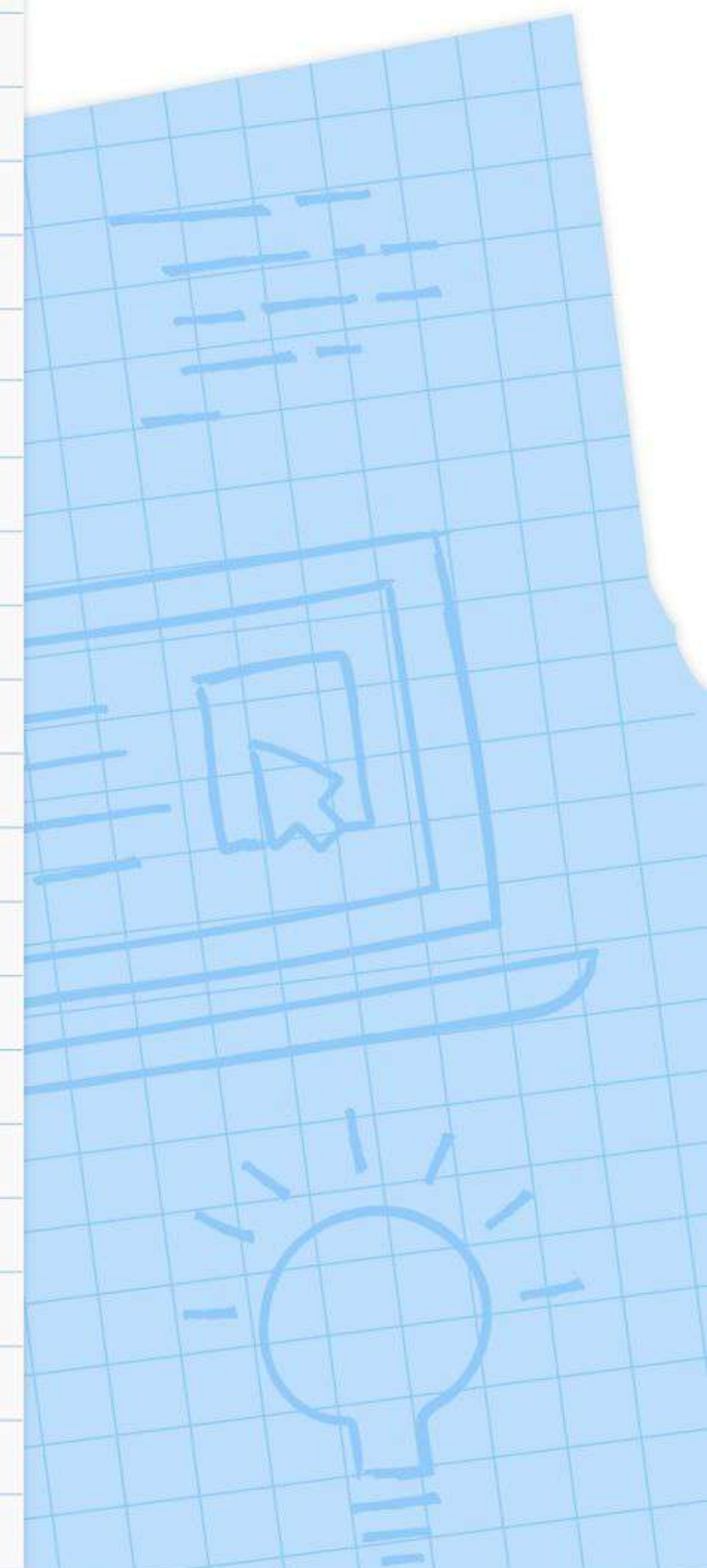




# Static Site Generation

## Plusy i minusy

- + Błyskawiczny czas ładowania stron
- + Minimalne obciążenie serwera
- + Możliwość efektywnego cache'owania
- + Niskie First Contentful Paint (FCP) i Time to Interactive (TTI)
- + Szybsze indeksowanie przez crawlery
- + Niższe koszty hostingu
- + Łatwe skalowanie poprzez CDN
- + Minimalne wymagania serwerowe
- Brak możliwości personalizacji w czasie rzeczywistym
- Dużo treści do obsłużenia, np. przez zaawansowaną strukturę strony (dużą liczbę podstron) może skutkować długim czasem buildu
- Ograniczona dynamiczność
- Zmiana danych na serwerze wymaga powtórnego przeprowadzenia buildu

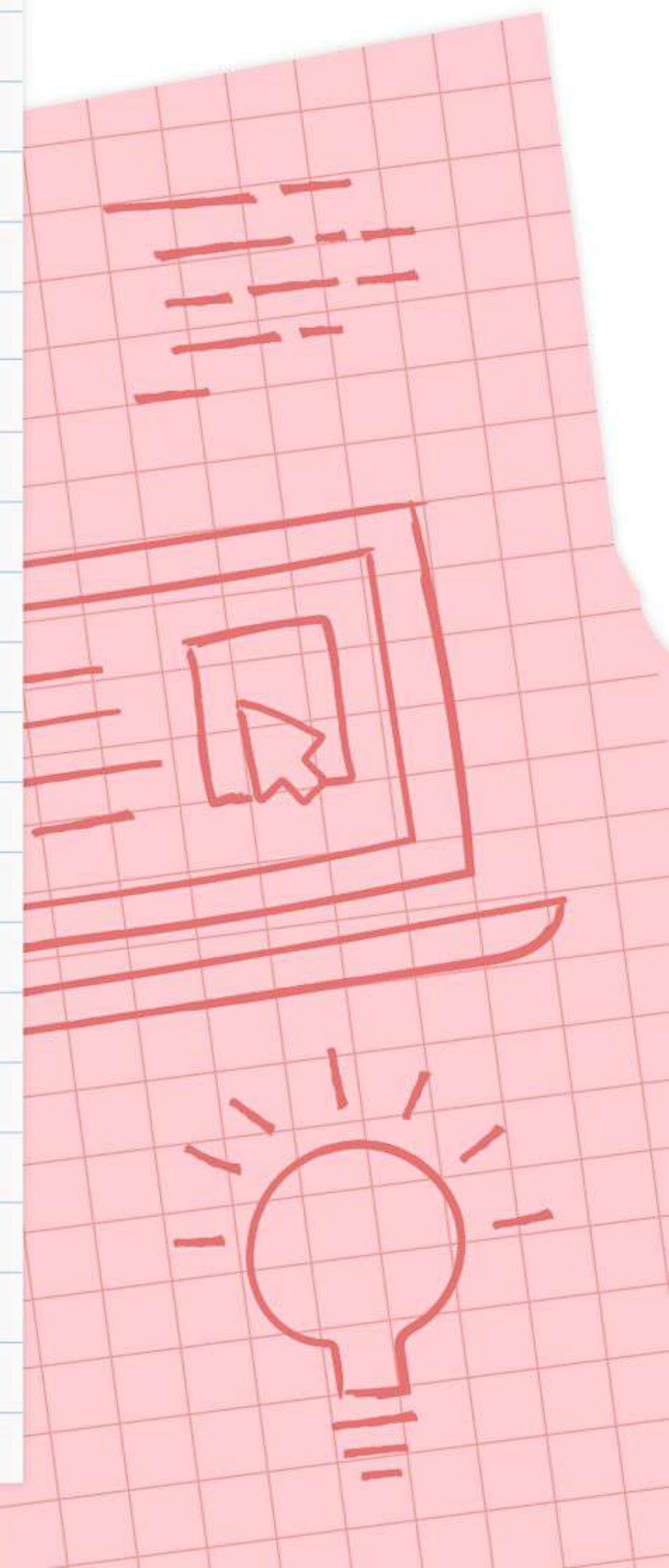




# Incremental Static Regeneration

## Podejście hybrydowe

Technika wprowadzona przez Next.js, która pozwala na stopniowe odświeżanie statycznie wygenerowanych stron bez konieczności przebudowywania całej aplikacji



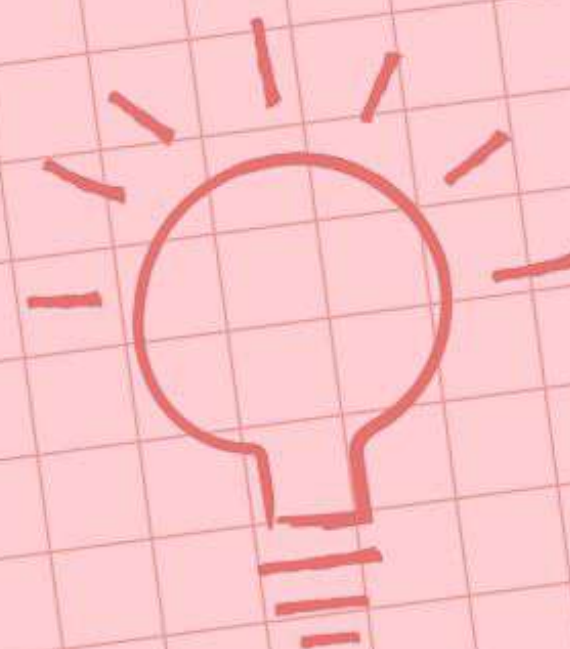


# Incremental Static Regeneration

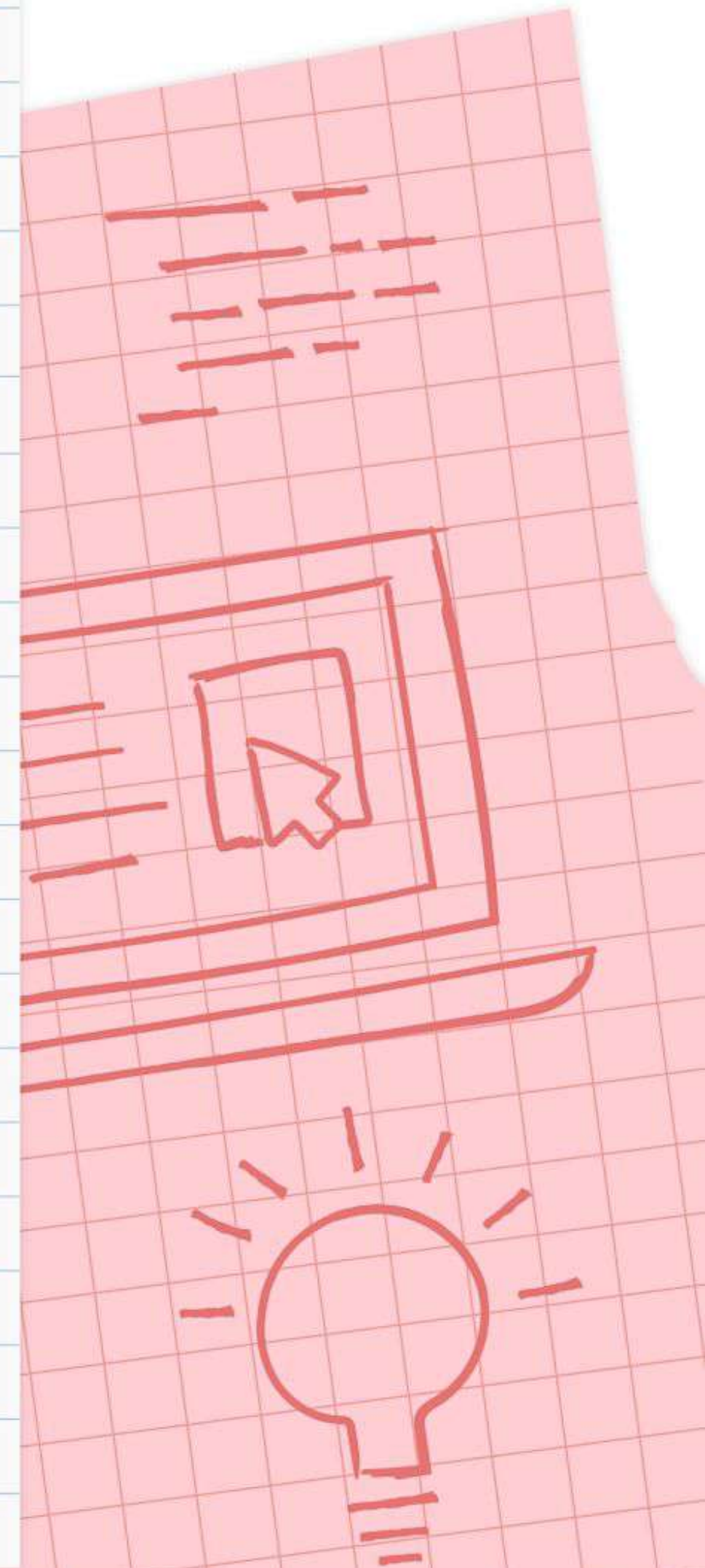
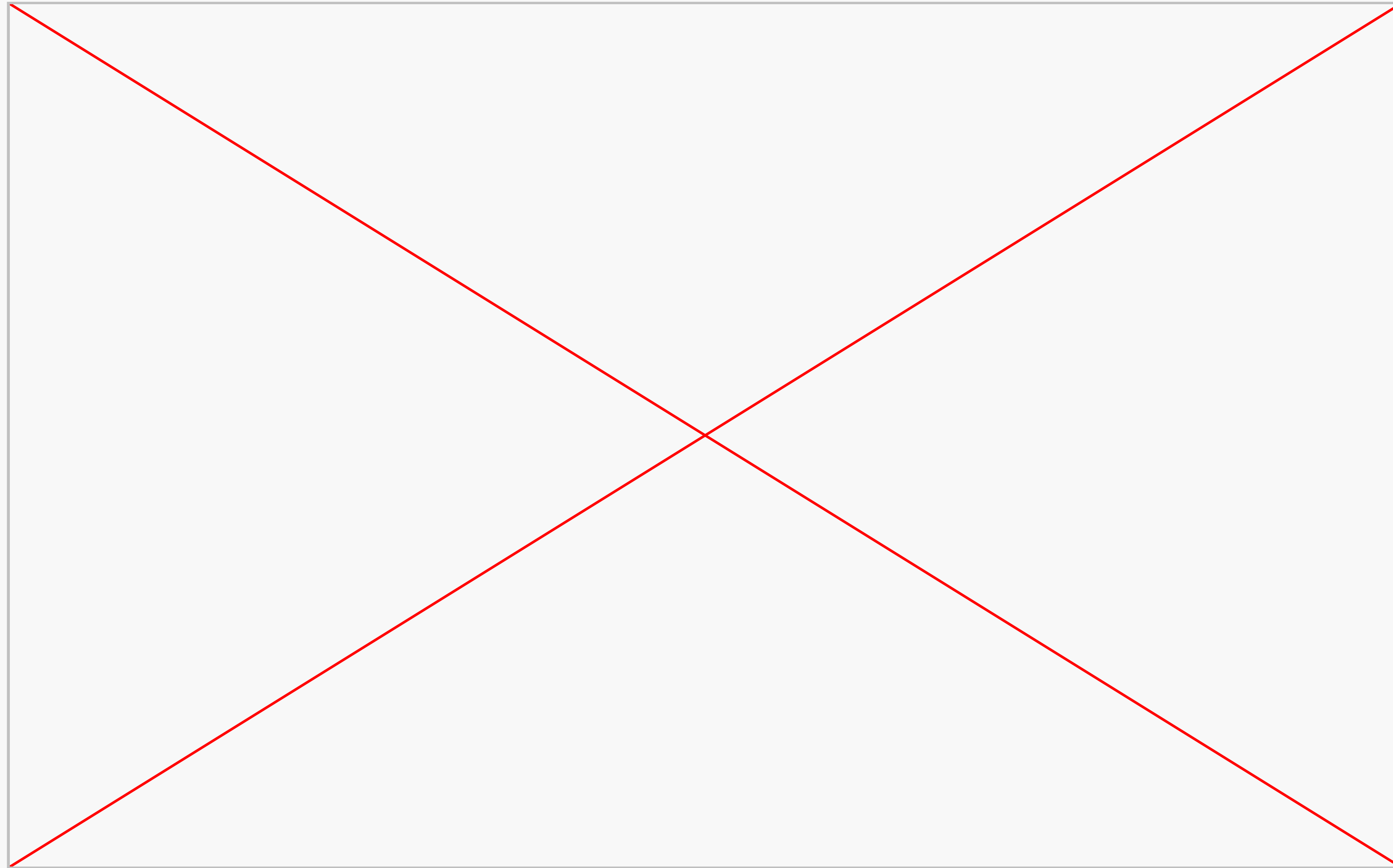
Kiedy użytkownik odwiedza stronę po upływie tego czasu, Next.js:

- Wyświetla najpierw starą wersję strony (z cache)
- W tle generuje nową wersję
- Kolejni użytkownicy zobaczą już zaktualizowaną wersję

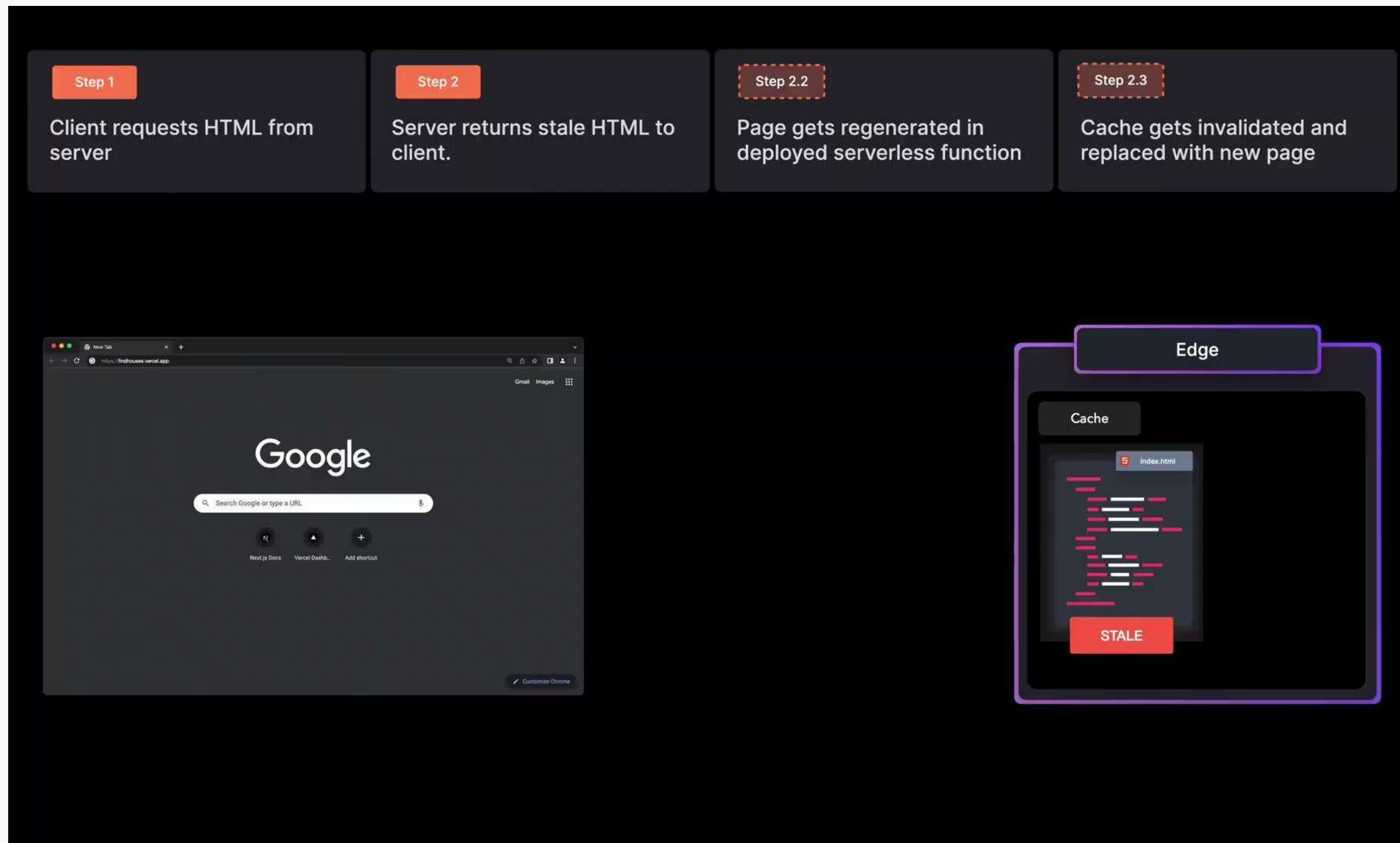
```
javascript Copy  
  
export async function getStaticProps() {  
  return {  
    props: {  
      data: await fetchData()  
    },  
    revalidate: 60 // Odświeżaj co 60 sekund  
  }  
}
```









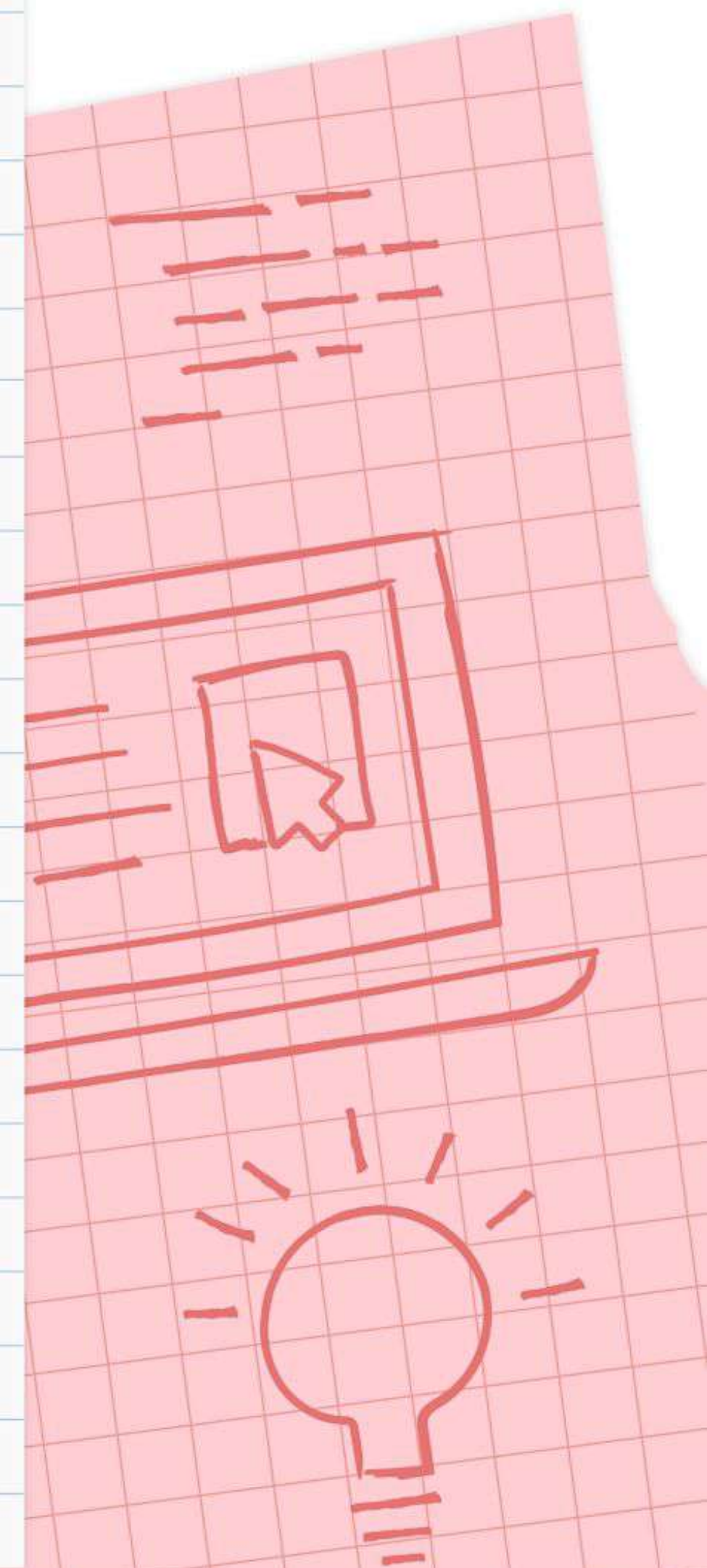




# Incremental Static Regeneration

## plusy i minusy

- + Zachowuje korzyści statycznej generacji (szybkość, SEO)
- + Pozwala na aktualizację treści bez pełnego rebuilda
- + Skaluje się dobrze nawet przy dużej ilości stron
- + Zmniejsza obciążenie serwera i bazy danych
- Nie nadaje się do treści wymagających natychmiastowej aktualizacji
- Wymaga odpowiedniej konfiguracji cache'owania
- Dostępne tylko w Next.js (choć podobne rozwiązania pojawiają się w innych frameworkach)

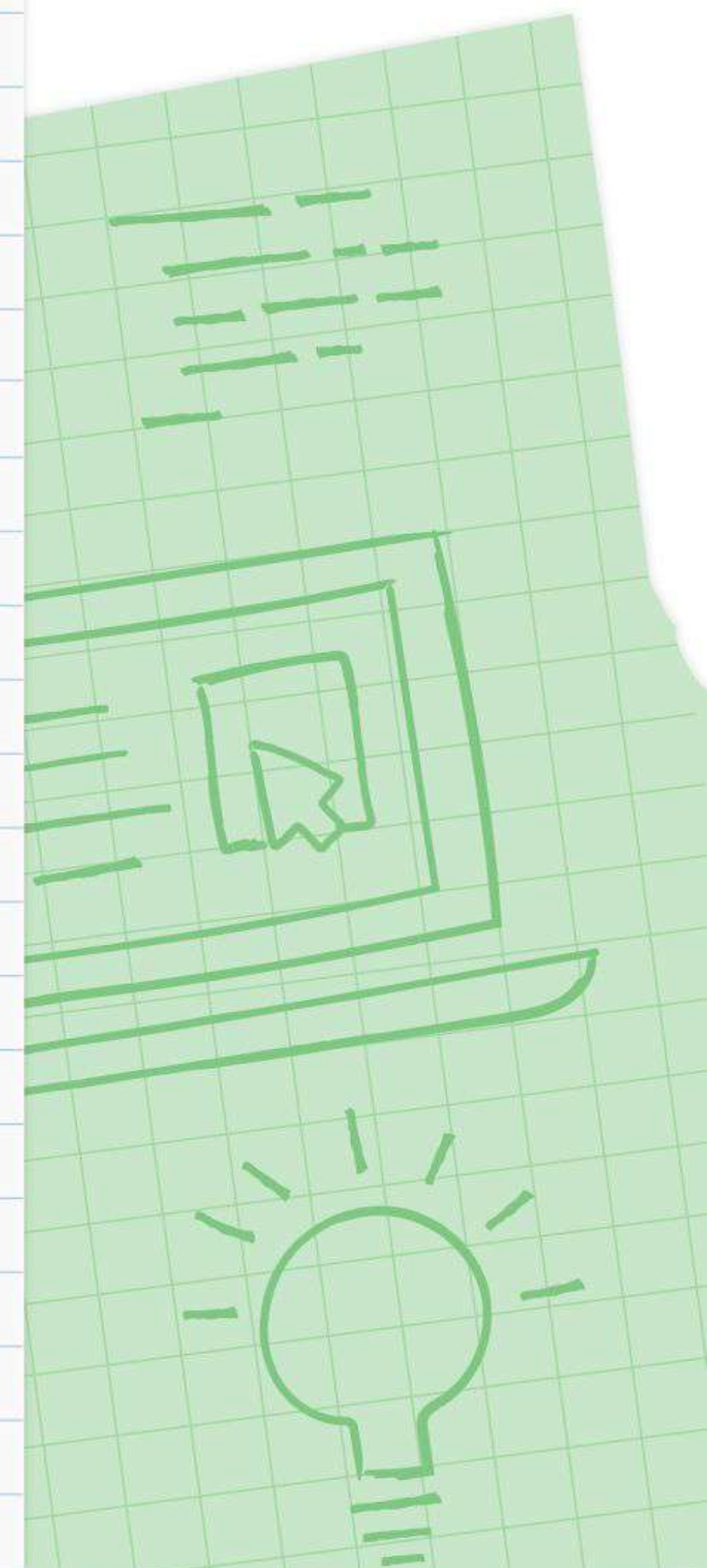




# Porównanie

## ISR vs SSG

- ISR to rozwiązanie hybrydowe, które rozwija koncept SSG
- SSG tworzy **wszystkie** strony, ISR dopuszcza możliwość aktualizacji ich za pomocą okresu rewalidacji lub on-demand
- ISR oferuje podejście inkrementacyjne w buildowaniu w przeciwieństwie do bazowego SSG
- ISR przez większą elastyczność oferuje nieco lepsze możliwości skalowania

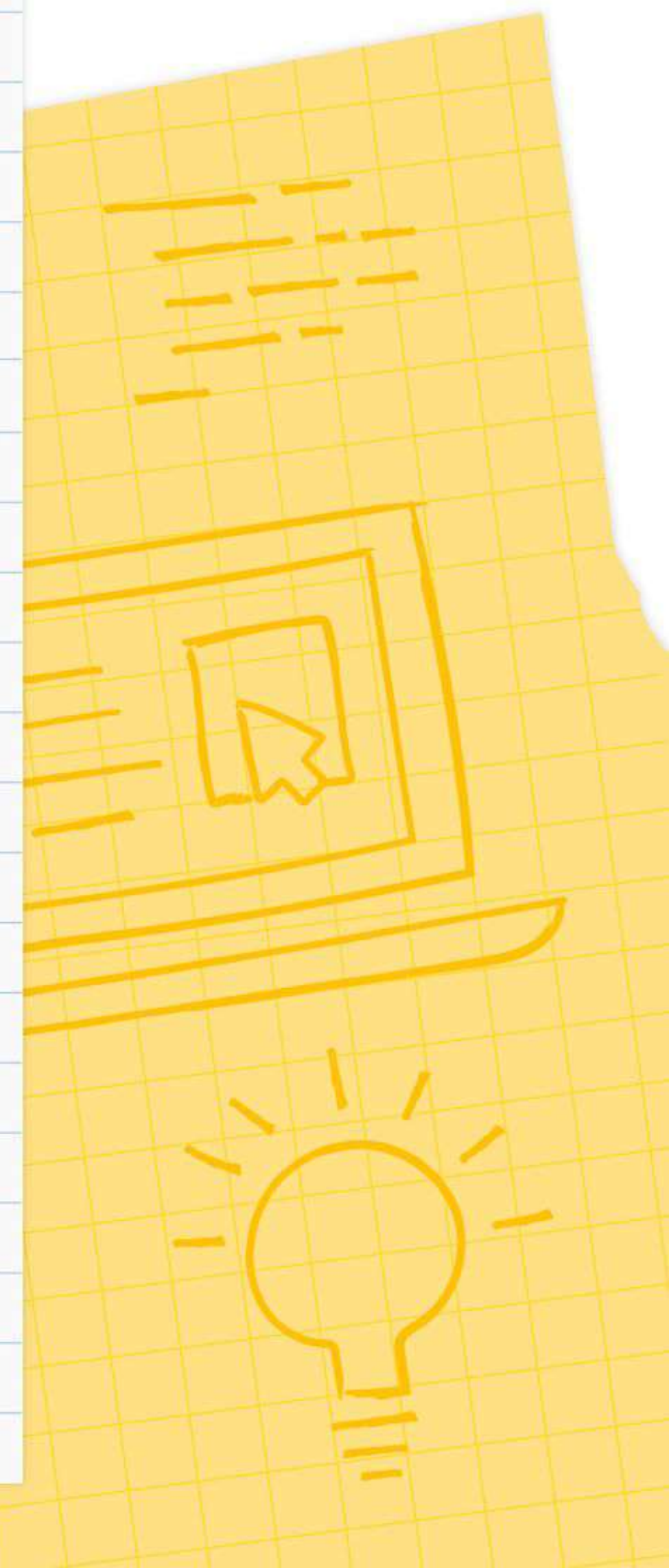




# Client Side Rendering

## Pora na wycieczkę do SPA

W przypadku CSR wyłącznie szkielet, minimum HTML jest renderowane przez serwer. Cała reszta jest obsługiwana przez kod JavaScript po stronie przeglądarki / klienta. Podejście jest związane z budowaniem aplikacji typu Single Page Application (SPA).







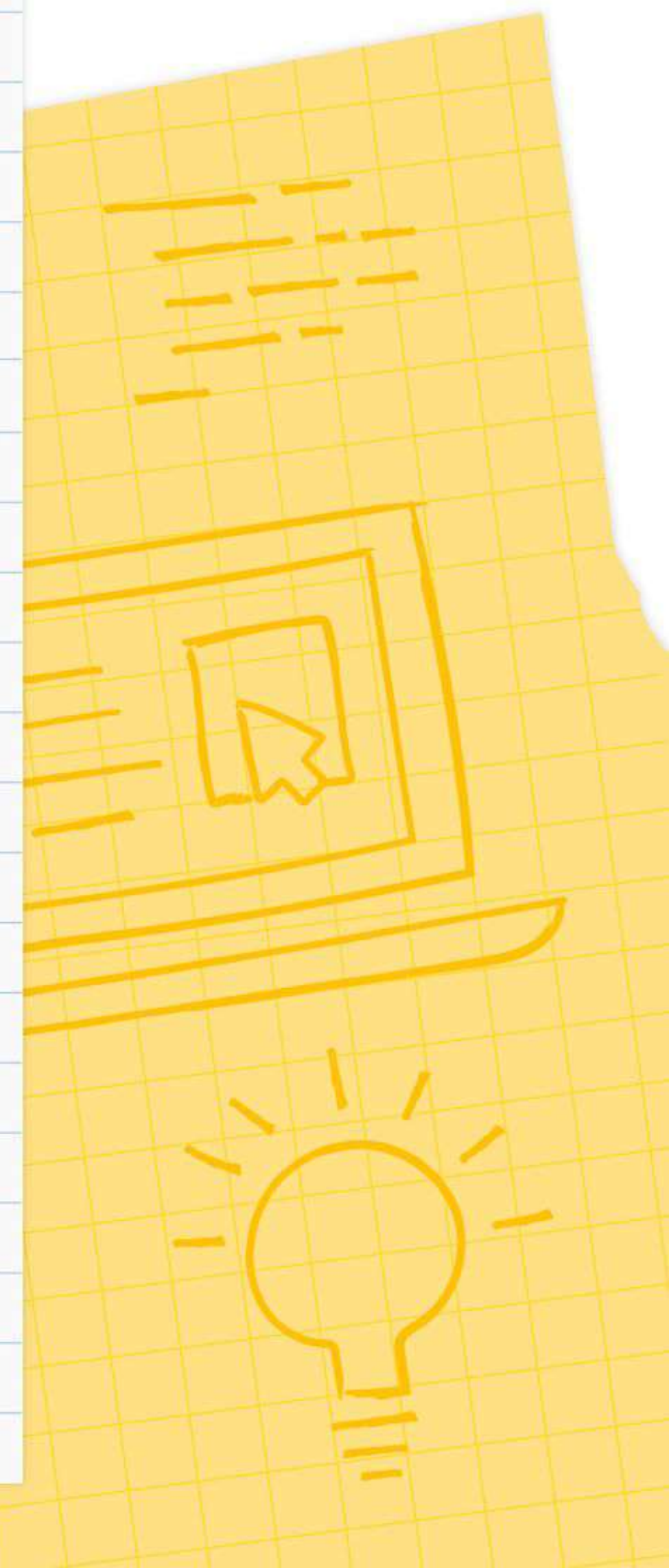
<https://www.youtube.com/watch?v=k-A2VfuUROg&feature=youtu.be>



# Client Side Rendering

## Plusy i minusy

- + nasze strony stają się potężnymi, interaktywnymi aplikacjami
- + nawigacja oferowana przez SPA, bez przeładowania strony
- ciężkie do optymalizowania pod kątem SEO (web crawlery)
- obciążenie klienta obowiązkiem ładowania danych

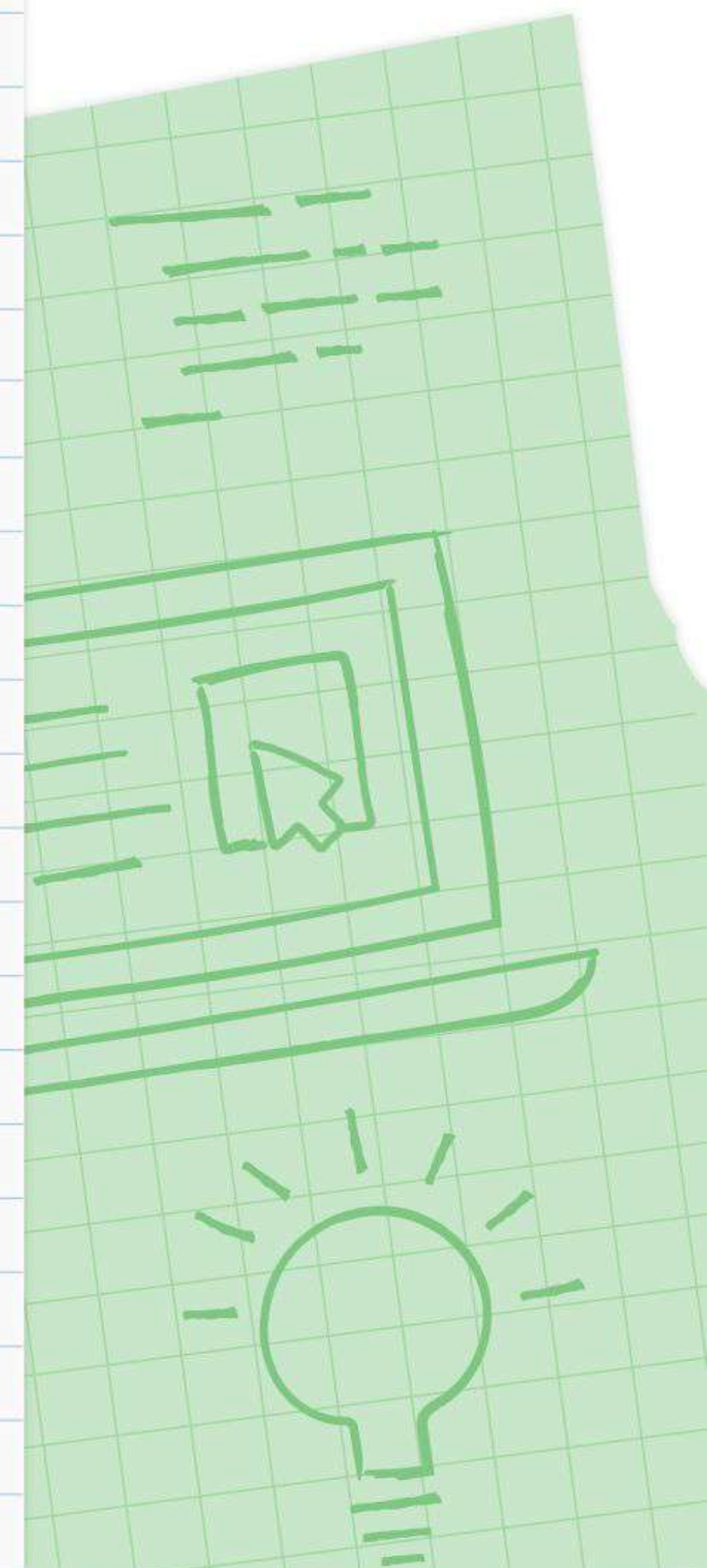




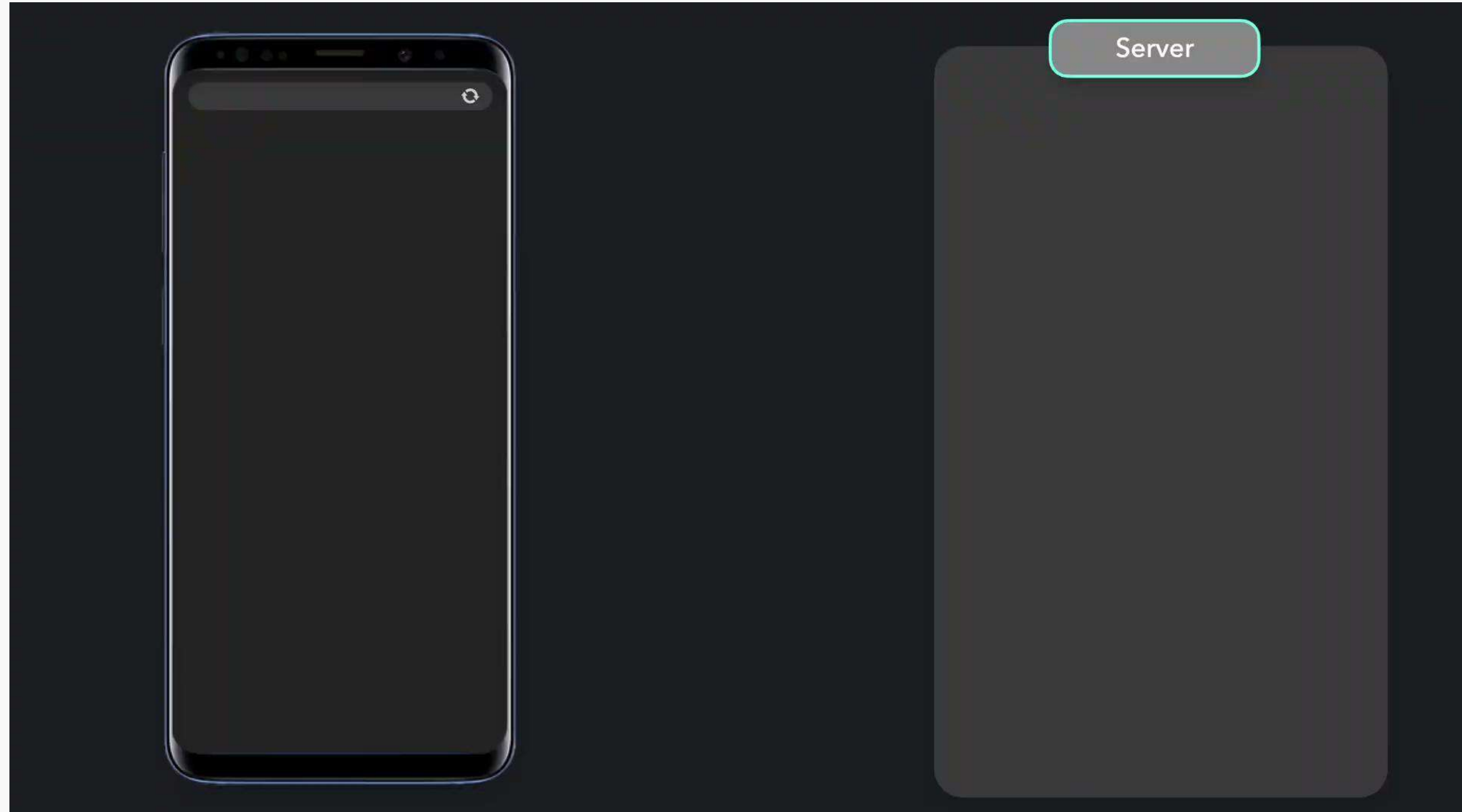
# Server Side Rendering

GET /lecture

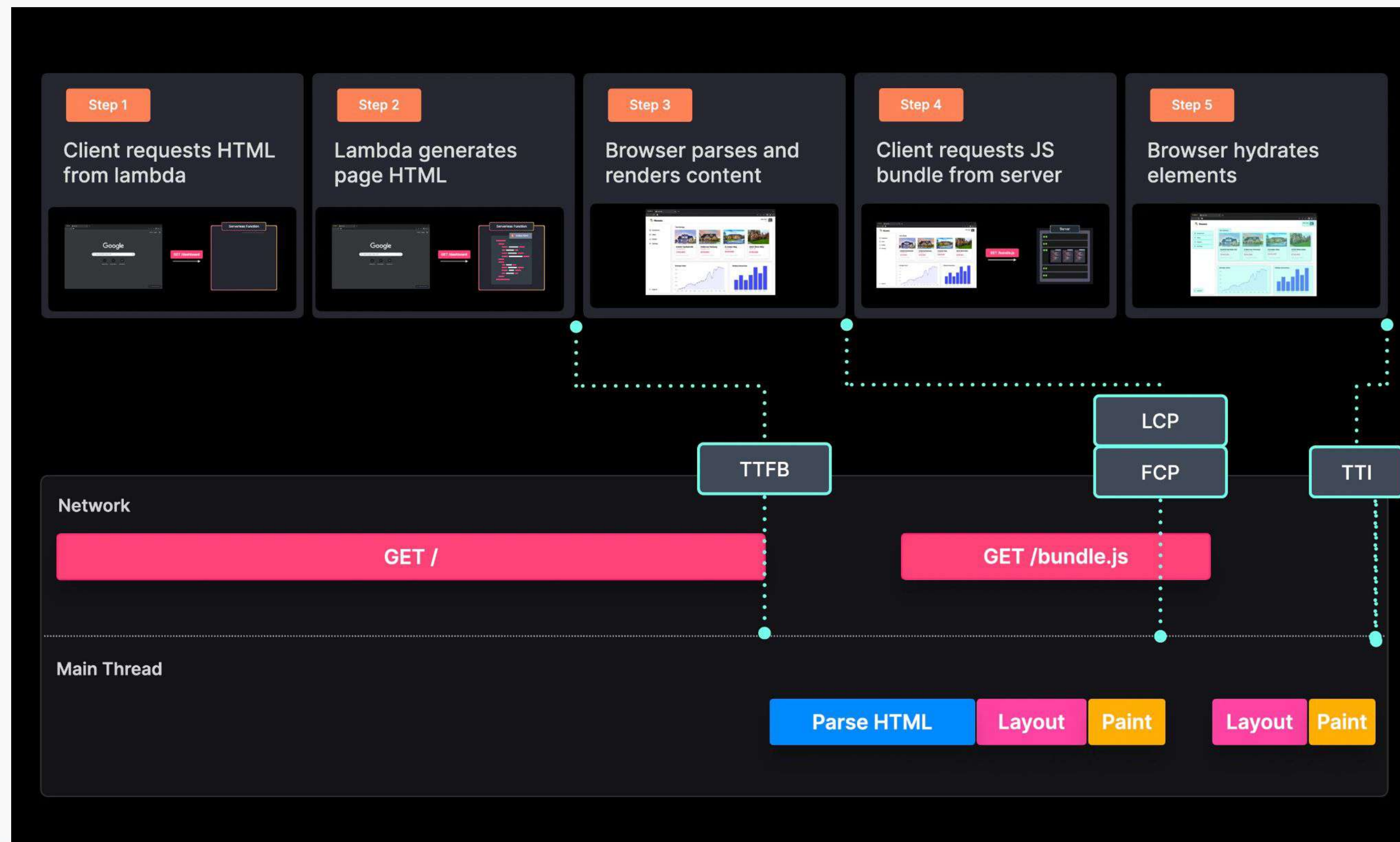
W przypadku SSR proces generowania HTML przenosi odpowiedzialność na serwer. Serwer renderuje naszą stronę, a w pełni wyrenderowany HTML trafia jako odpowiedź do żądania klienta.









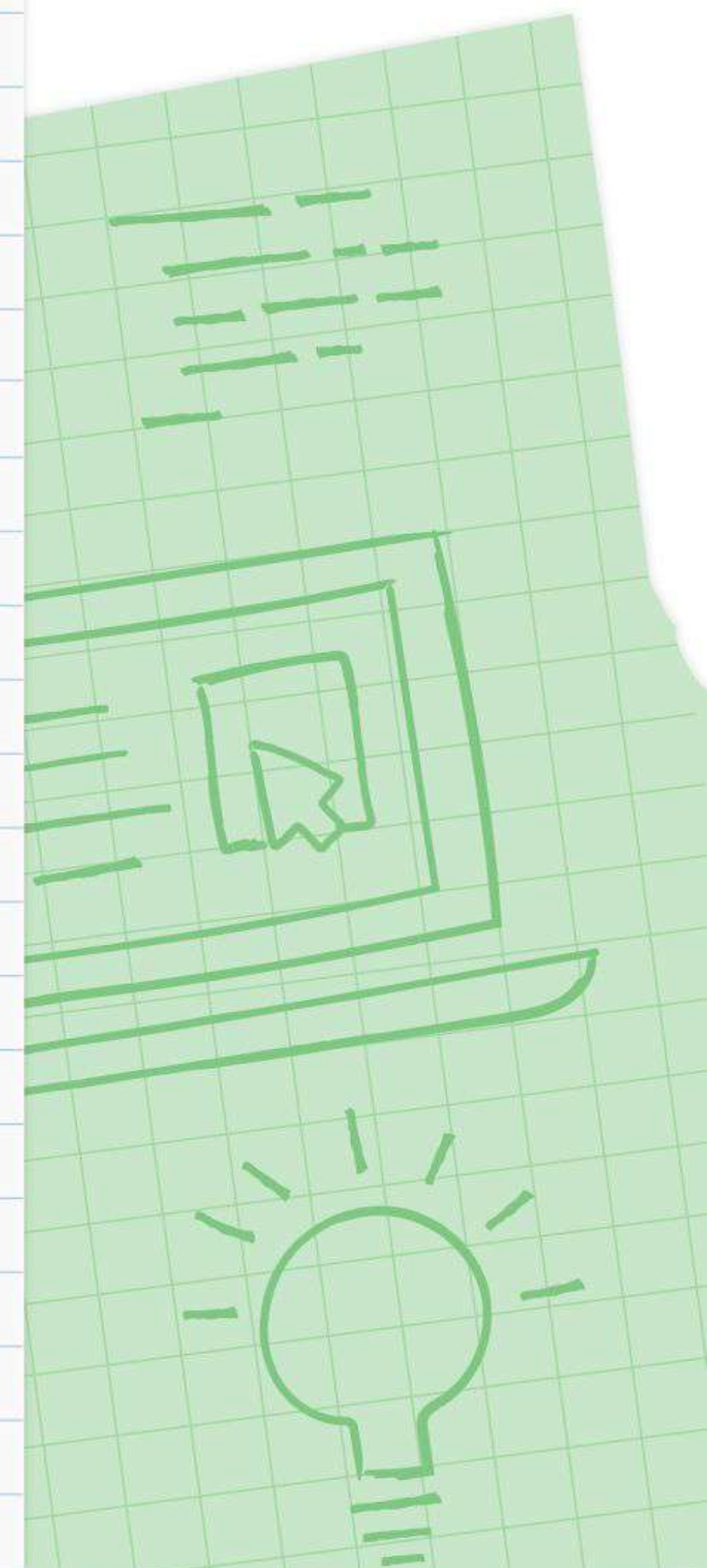




# SSR

## Plusy i minusy

- + Zawsze aktualne informacje
- + Mniej kodu JavaScript powoduje szybsze FCP oraz TTI
- + "SEO friendly"
- + Wrażliwe operacje wykonywane po stronie serwera
- Wolne TTFB - przetwarzanie po stronie serwera
- Możliwe opóźnienia przy dużym ruchu
- Ograniczone możliwości cache
- W klasycznym SSR ciężiej o interakcje realizowane przez kod po stronie klienta
- Wyższe koszty hostingu
- Potrzeba skalowalnej infrastruktury

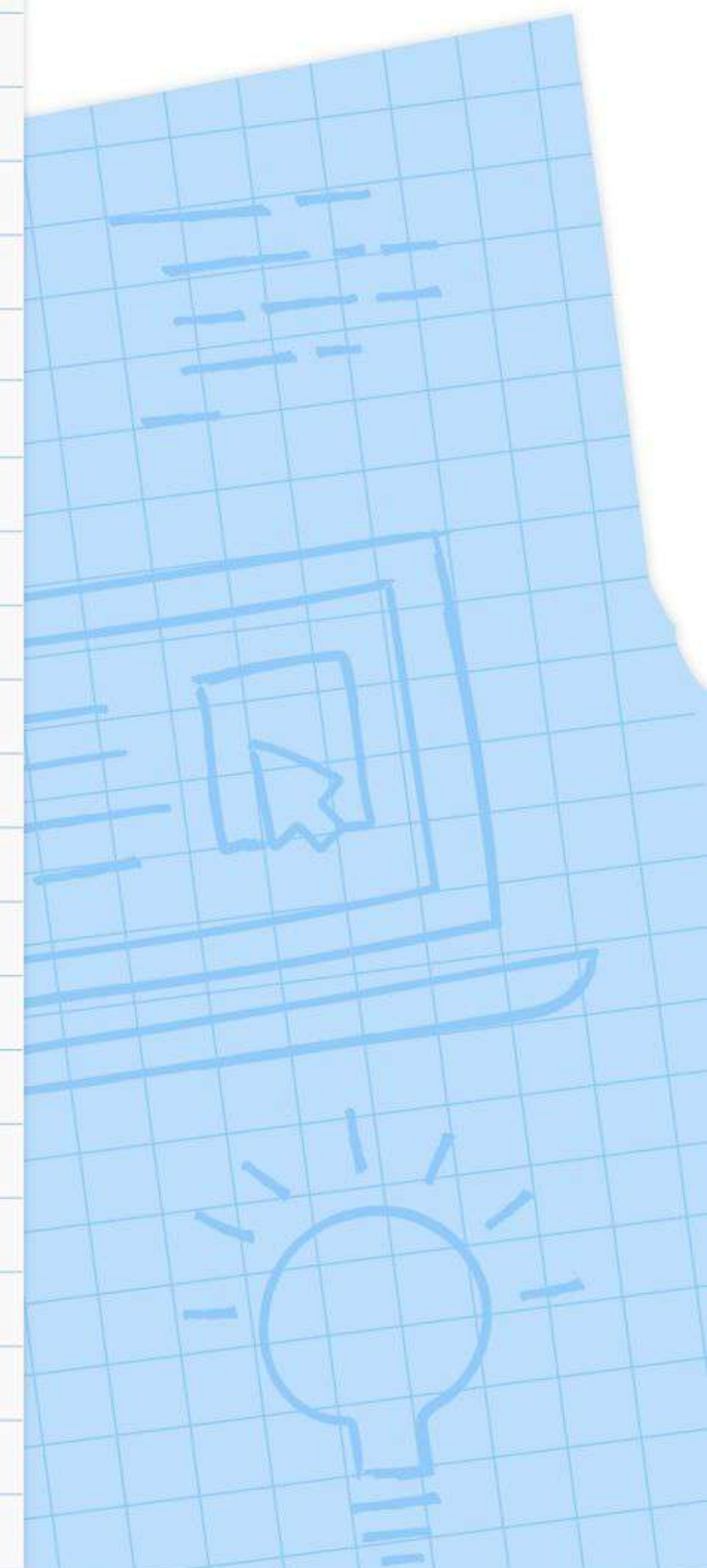




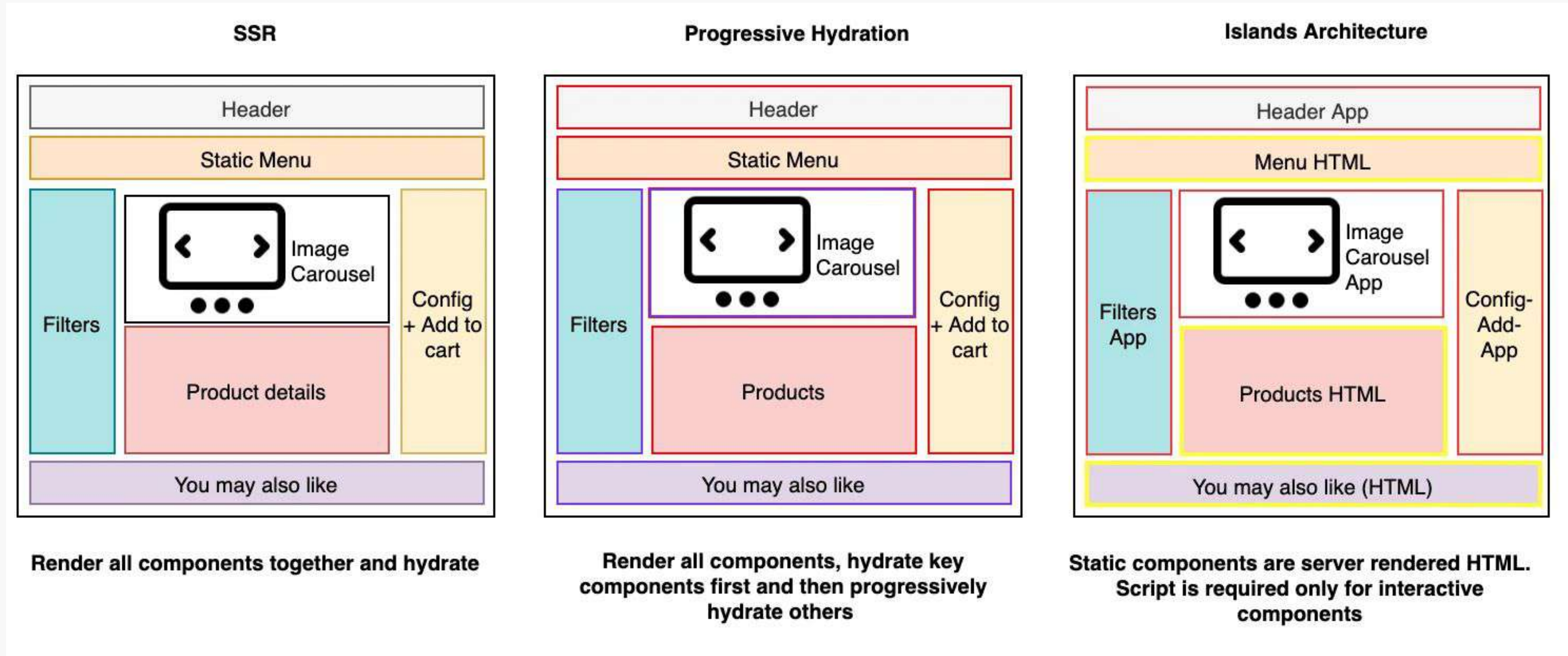
# Architektura “wysp”

## Island Architecture

- stosunkowo nowoczesne podejście, w którym większość strony to statyczny HTML z określonymi interaktywnymi "wyspami" komponentów dynamicznych, które są hydratowane przy użyciu JavaScript.
- ściśle związany z konkretnymi frameworkami (najpopularniejszym jest Astro)
- strategia optymalizacji, której celem jest zapewnienie jak najmniejszego “bundla” kodu JavaScriptu





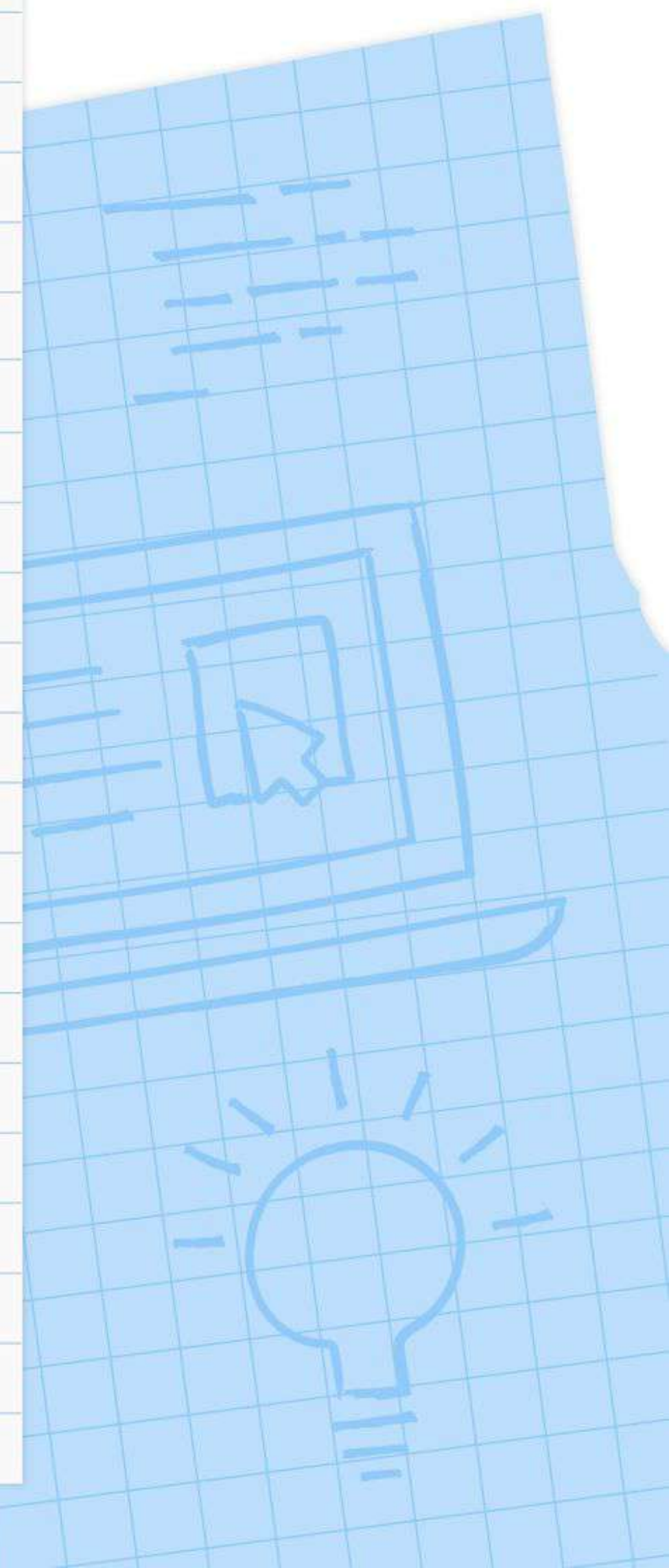




# Architektura “wysp”

## Plusy i minusy

- + “SEO friendly” przez renderowanie treści na serwerze
- + priorytetyzacja ważnych elementów strony
- + łatwa organizacja elementów budulcowych za pomocą architektury komponentów
- względnie nowy rendering pattern ograniczony konkretnymi narzędziami
- przez swoje kluczowe koncepty podejście nie sprawdzi się w przypadku wysoce interaktywnych aplikacji, których zbudowanie wymagałoby tysięcy “wysp”





# Podsumowanie

	SSR*	SSG	ISR	CSR
Generowanie HTML	Serwer	Build Serwer	Build Serwer	Klient
Zachowanie SPA	Niemożliwe / Ograniczone	Niemożliwe	Niemożliwe	Dostępne
Czytelność dla Crawlera	Pełna ⬇	Pełna	Pełna	Ograniczona
Możliwość cache	Minimum	Pełna	Pełna	Minimum
TTFB	Wysokie ⬇	Niskie	Niskie	Niskie ⬇
TTI:FCP	TTI = FCP / TTI > FCP	TTI = FCP	TTI = FCP	TTI >> FCP
Zastosowanie	W zależności od rodzaju SSR <small>* rozpatrywany przypadek klasyczny / z hydratacją</small>	Strony z przewagą statycznej zawartości i takie, które nie wymagają częstych aktualizacji np. portfolio, dokumentacja (hehe)	Strony wymagające częstego aktualizowania treści: e-commerce, ogłoszenia, kalendarze	Wysoce interaktywne aplikacje webowe (np. Instagram)





# Dziękuję za uwagę

