Frontend Development (r)Evolution

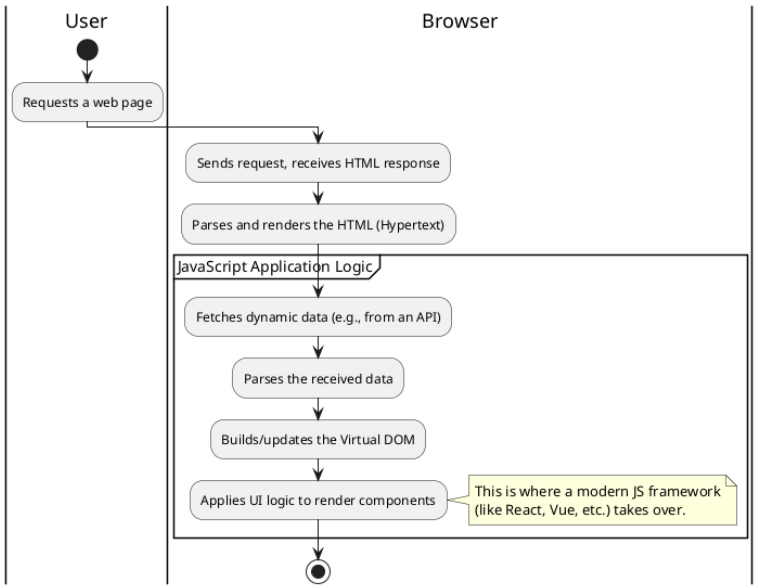
Oleksii Koval

August 2, 2025

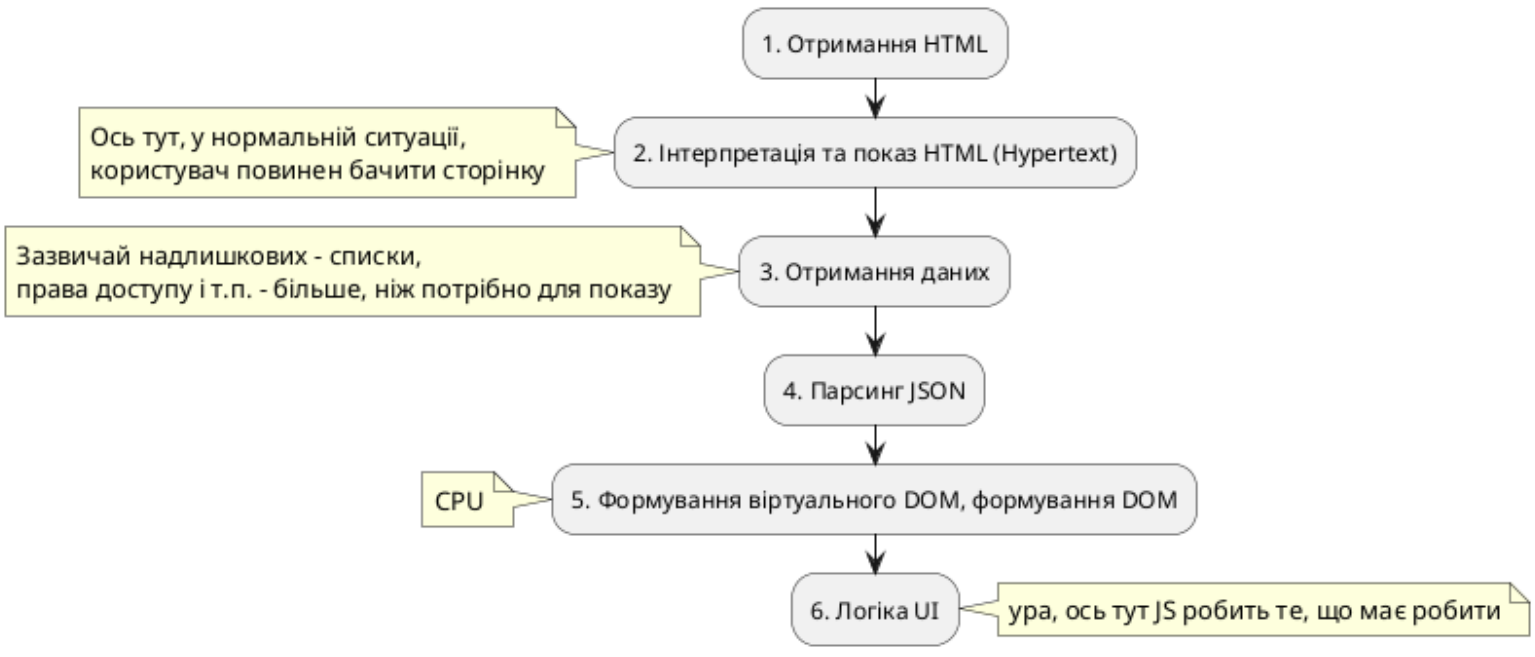
Принцип Оккама

Із множини сутностей слід обирати ті, що необхідні для пояснення явищ, уникаючи зайвих припущень.

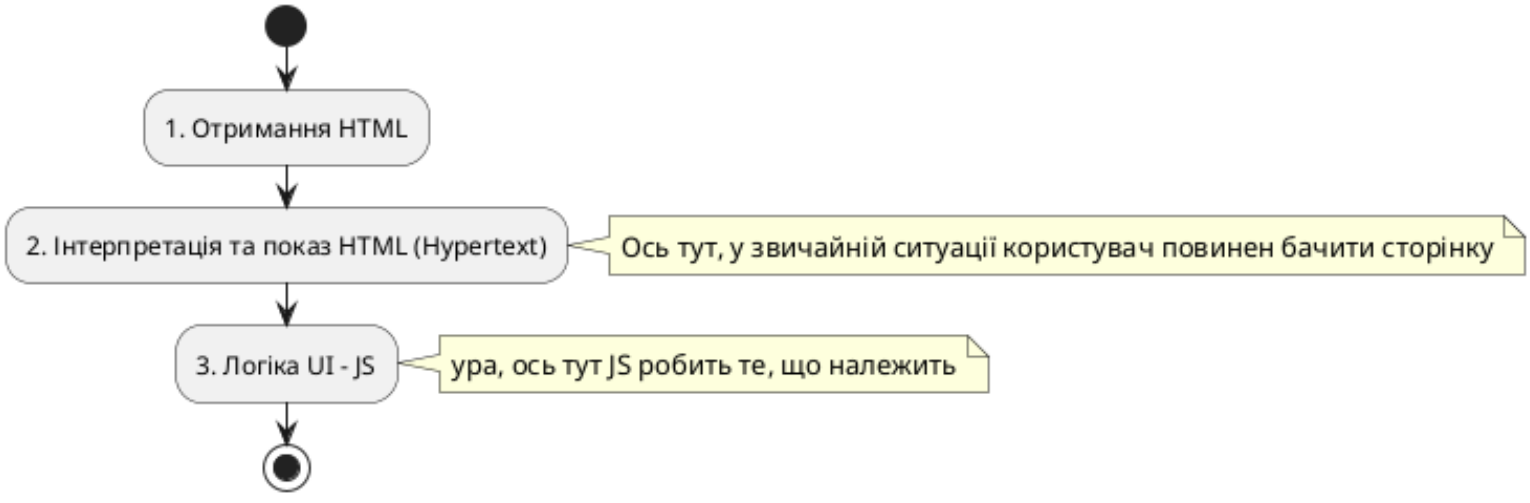
Надлишковість фронтенду у SPA



Що не так


1. Отримання HTML
Ось тут, у нормальній ситуації,
користувач повинен бачити сторінку
2. Інтерпретація та показ HTML (Hypertext)
Зазвичай надлишкових - списки,
права доступу і т.п. - більше, ніж потрібно для показу
3. Отримання даних
4. Парсинг JSON
CPU
5. Формування віртуального DOM, формування DOM
6. Логіка UI
ура, ось тут JS робить те, що має робити

Shorten it / how it should be



Що не так з погляду бізнесу

- Значні зусилля на розробку та підтримку SPA
- На відміну від Django, у NodeJS мало готових абстракцій:
  - tables (django-tables, sdh, sdh.v3)
  - paginator
  - django forms
  - тощо...
- Підтримка - $$$
  - проблеми з оновленнями та екосистемою npm
  - обсяг коду (boilerplate!)
- Швидкодія:
  - високе навантаження на CPU та RAM

Чому відразу не було оптимально

Чому відразу не здогадалися зробити так?

1. Відсутність frontend-reactivity у Hypertext
2. Відсутність backend-reactivity у Hypertext

Эволюция Hypertext

- Пакети типу AlpineJS розширюють HTML і вирішують питання frontend-реактивності

- Пакети типу HTMX розширюють HTML тегами для backend-реактивності

Що не так з точки зору програміста

- Надлишкова комунікація при розробці та підтримці (чия форма — FE? BE?)
  - необхідність синхронізації роботи
  - синхронізація даних — REST, структури даних тощо
- Усе це виливається в купу додаткової роботи та очікування
- REST був задуманий інакше, автор не очікував такого використання. Як задумано:
  - statelessness
  - client-server architecture
  - uniform interface
  - cacheability
  - layered system

Особиста історія, чому

Загалом так собі досвід. Приходили JS-розробники, писали код, потім йшли (FE-деви не могли сидіти на роботі більше 2 років, зазвичай рік), і приходили нові — переписували все з нуля.

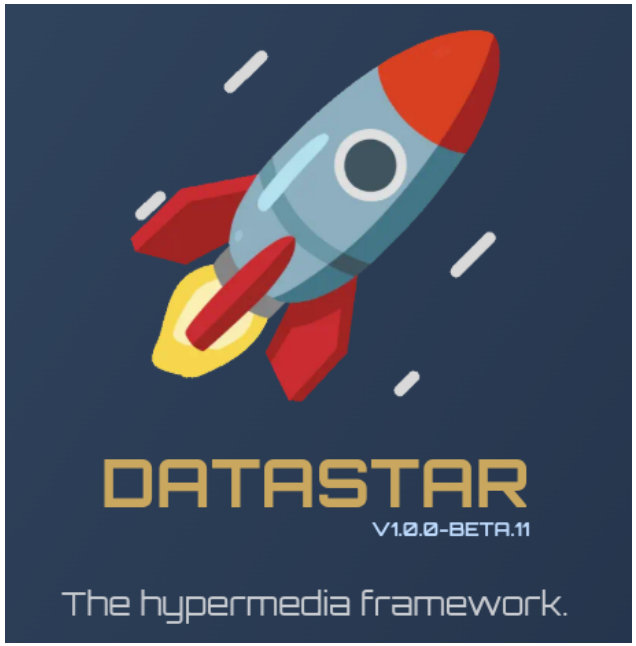
—

Дуже дорого, невигідно. Я працюю на Python і не розумію, чому кожен новий програміст — це проєкт з нуля. Адже у нас не так. Гівнокод? Гівнотехнології, що не живуть більше пари років?

Purpose of This Work

- In brief - make an experiment: lets take the most complex page in the project and rebuild it to demonstrate:
  - Simplicity of alternative approaches (code volume/effort)
  - Implementation speed
- Test these technologies in real-world scenarios
- Prove to myself that I can do this
- Show others that alternative paths exist

What is data-star dev?



Since late 2024, it's become possible to combine the advantages of frontend and backend applications.

Size & Performance

Core Concepts

The boundaries have blurred, at least for applications that don't require offline functionality. To understand how this works, we need to stop thinking of frontend as a set of components + API and instead embrace two simple concepts:

- Bidirectional signals (variables accessible from different parts of the system, including the backend)
- Fragments - any HTML element with an id attribute that can be replaced at any time

References

- https://data-star.dev
- Simple Made Easy
- https://www.reddit.com/r/Clojure/comments/1jtih16/comment/mm0xrih/?context=10000
- Real REST - how it supposed to be
- https://github.com/hyperfiddle/electric
- Testimonials:
  - https://chrismalek.me/posts/data-star-first-impressions/

Коли SPA все ще має сенс

1. Коли йде робота з великим артефактом — файлом, що є результатом роботи. Наприклад, Figma.
2. Якщо застосунок має працювати і в офлайн-режимі.
3. Якщо є потреба перекласти CPU-інтенсивні обчислення на бік клієнта.