Problemas comunes, fáciles de resolver a medias

Web Avanzado 2025-1 | Benjamín Vicente

Los sistemas webs son complejos

Partiendo que son sistemas distribuidos con envío de código…

Veamos algunos casos comunes

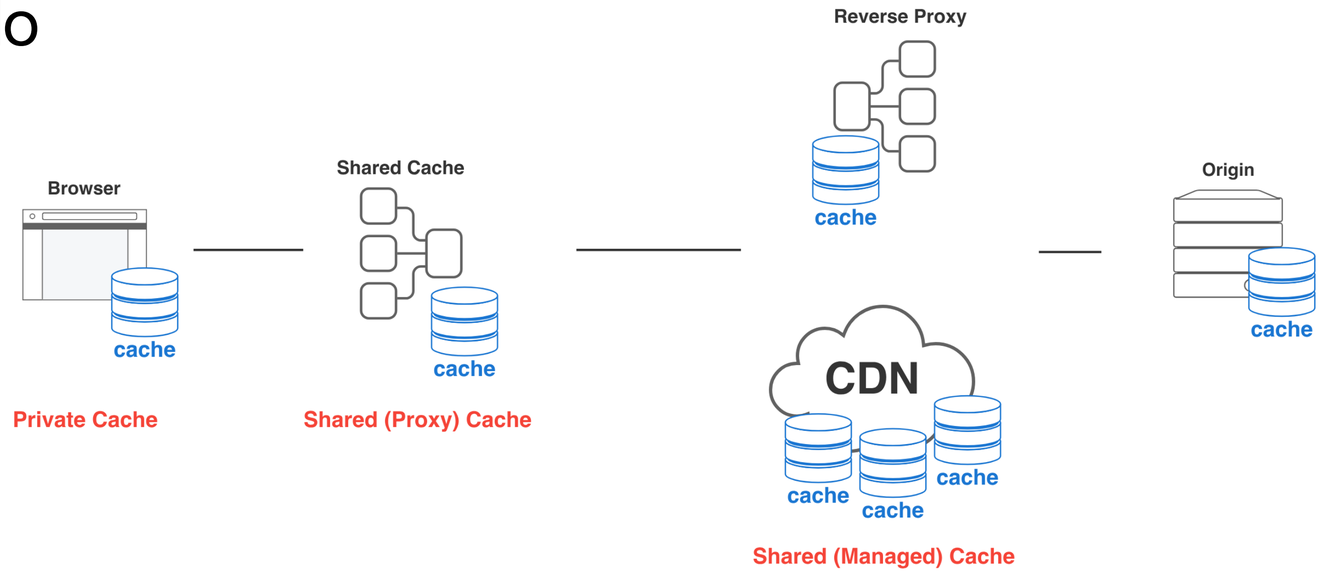
- HTTP Caching
- Carga de datos
- Estado complejo
- Componentes con estilos reutilizables
- [Extra] Devtools: performance pannel + teléfono

HTTP Caching

HTTP Caching

Consideraciones generales que hay que tener en cuenta:

- Sistemas emplean heurísticas para evitar consultas
- El cache puede ser compartido



https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Caching

HTTP Caching

Generalmente algo como `Cache-Control: max-age=3600` sirve y basta (1h)

¿Pero cuando no?

HTTP Caching

Imaginemos que el landing de una página tiene una foto de sobre 5MB

Queremos que la foto **siempre esté actualizada**.

¡Pero no queremos descargar la foto cada vez!

Podemos tener algo que identifique si el archivo cambió

HTTP Caching

Headers de respuesta `Last-Modified` (fecha) y `ETag` (hash)

Si el navegador tiene el recurso en el cache, puede enviar el valor en la request.

Si el recurso no cambio, se responde `304 Not Modified` **sin contenido**.

¿Podemos mejorar esto? ¿Y en qué casos?

HTTP Caching

Identificación de versión en el URL: Cache busting. Ej:`main.8f3b2c4a.js`

Solo se debe usar cuando el archivo no es accedido directamente.

Con esto, el navegador **no necesita hacer una request al servidor**.

Se usa el header `Cache-Control: public, max-age=31536000, immutable`

HTTP Caching

Otras opciones a tener en cuenta:

- `stale-while-revalidate`
- `public` vs `private`
- `no-cache` vs `no-store`

Ojo que el navegador tiene heurísticas que no requieren todos los “hints” de los headers, y es mejor debuguear para ver el comportamiento deseado.

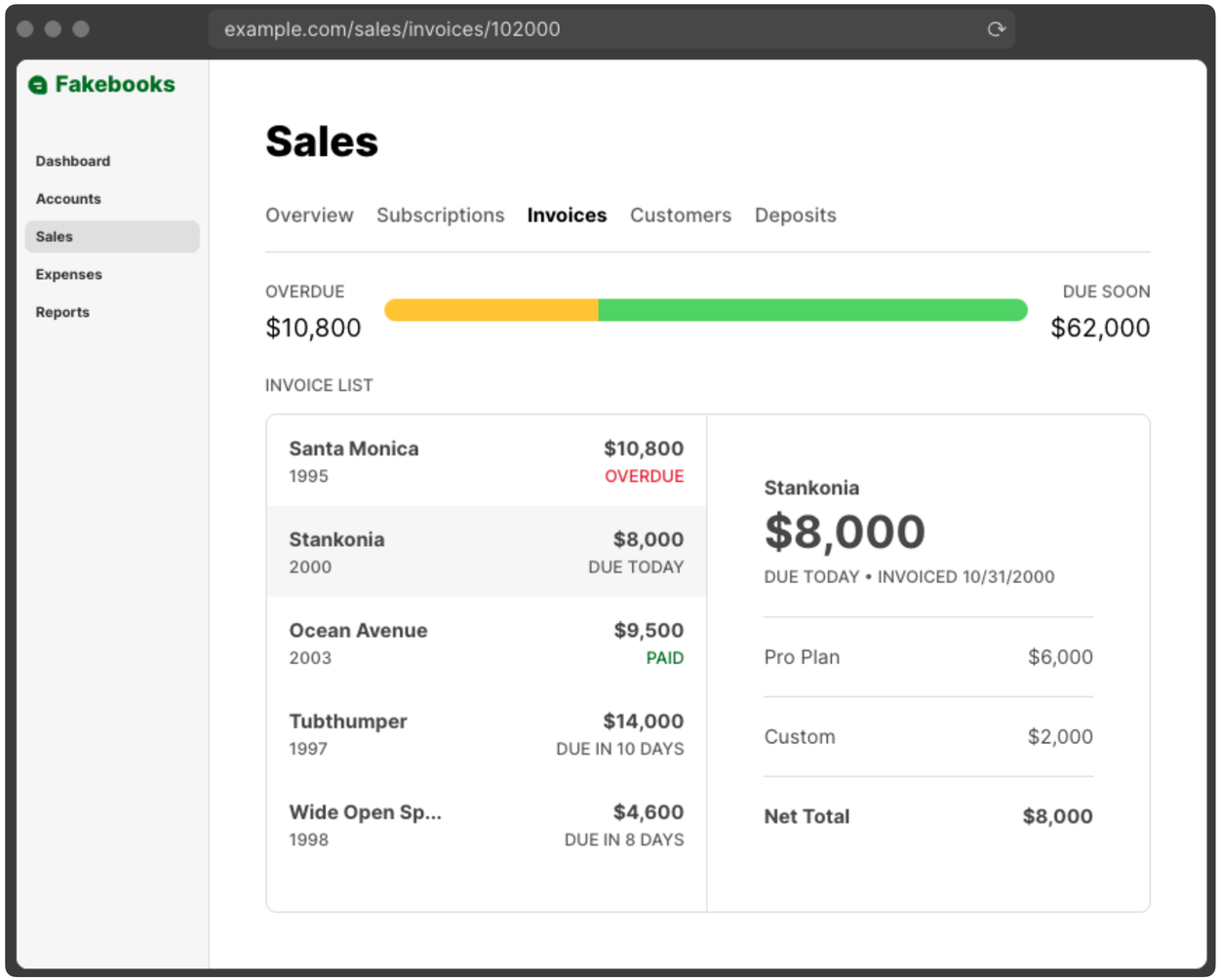
Carga de datos

Carga de datos

Por ejemplo, hay que cargar:

- El usuario
- La organización del usuario
- Ventas de la organización
- Detalle de una venta

¿Cómo cargamos eso rápido?



Carga de datos

```
function App() {
    const organization = useUserOrganization()
    if (!organization) {
        return  <div>Loading...</div>
    }
    return <OrganizationDashboard organization={organization} />
}
```

Se crea una cascada

Carga de datos

¿Cómo minimizamos el tiempo?

Carga de datos

Opción **acoplada al router**: Loaders de las páginas

- Next.JS Pages dir, SvelteKit, Remix, Tanstack Start
- Rutas (fragmentos) se cargan en paralelo

Desventaja: codigo de carga de datos puede estar muy lejos de donde se usa

Carga de datos

Opción **acoplada al router**:

- Next.JS Pages dir:

```
export async function getServerSideProps() {
  return { props: { name: "Web Avanzado" } }
}

export default function Home({ name }) {
  return (
    <div>
      Hola {name}
    </div>
  )
}
```

Carga de datos

Acoplada al router

Requests ocurren en paralelo, que pueden ser compactados en una sola consulta al servidor (single-flight)

Carga de datos

Opción **desacoplada**: Sistema de agregación de consultas

- Backend-for-Frontend
- GraphQL con Relay

Desventaja: capa adicional en la arquitectura, que puede se muy compleja

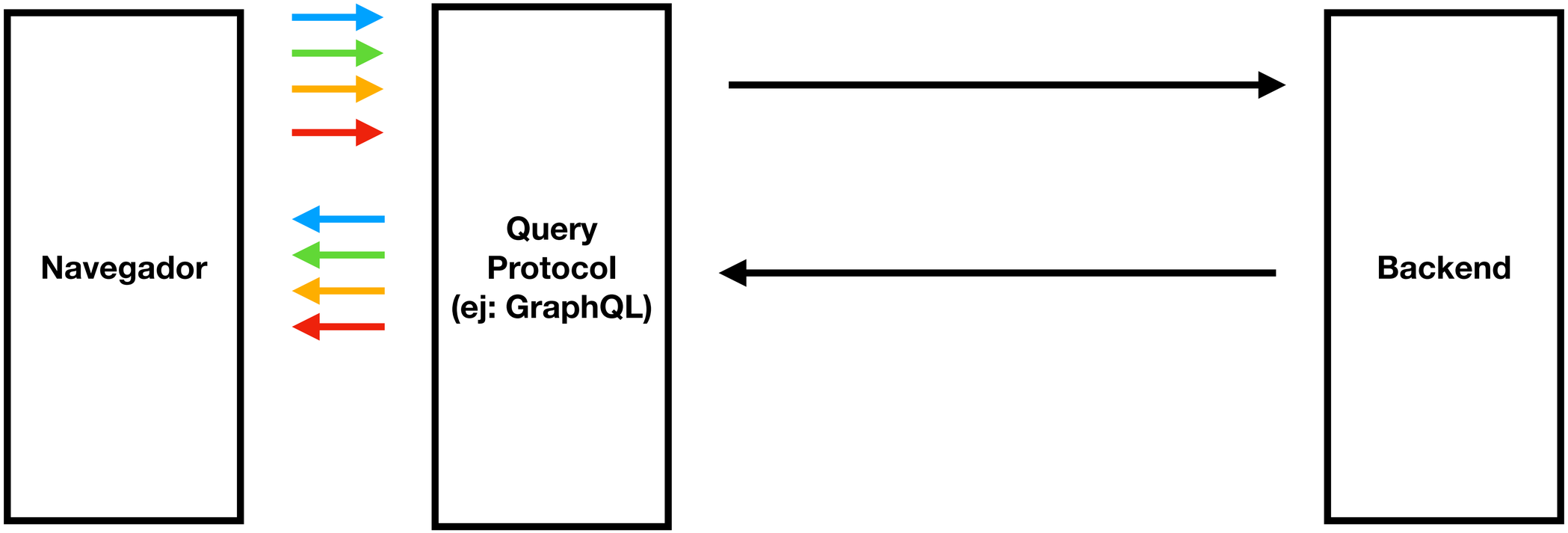
Carga de datos

Opción **desacoplada**

- GraphQL con Relay:

```
const data = useLazyLoadQuery<AppQuery>(
  graphql`
    query AppQuery {
      allFilms {
        films {
          id
          ...Film_item
        }
      }
    }
  `,
  {}
);
```

Carga de datos



Desacoplada

Protocolo intermediario optimiza consultas,
usualmente entendiendo relaciones entre sub-consultas

Carga de datos

Opción **acoplada a la UI**: Renderizado backend-first

- HTMX + Django, Livewire + Rails, Phoenix LiveView
- Marco.js, React Server Components

Desventaja: sin tener sync, la revalidación de datos suele ser más complicado

Carga de datos

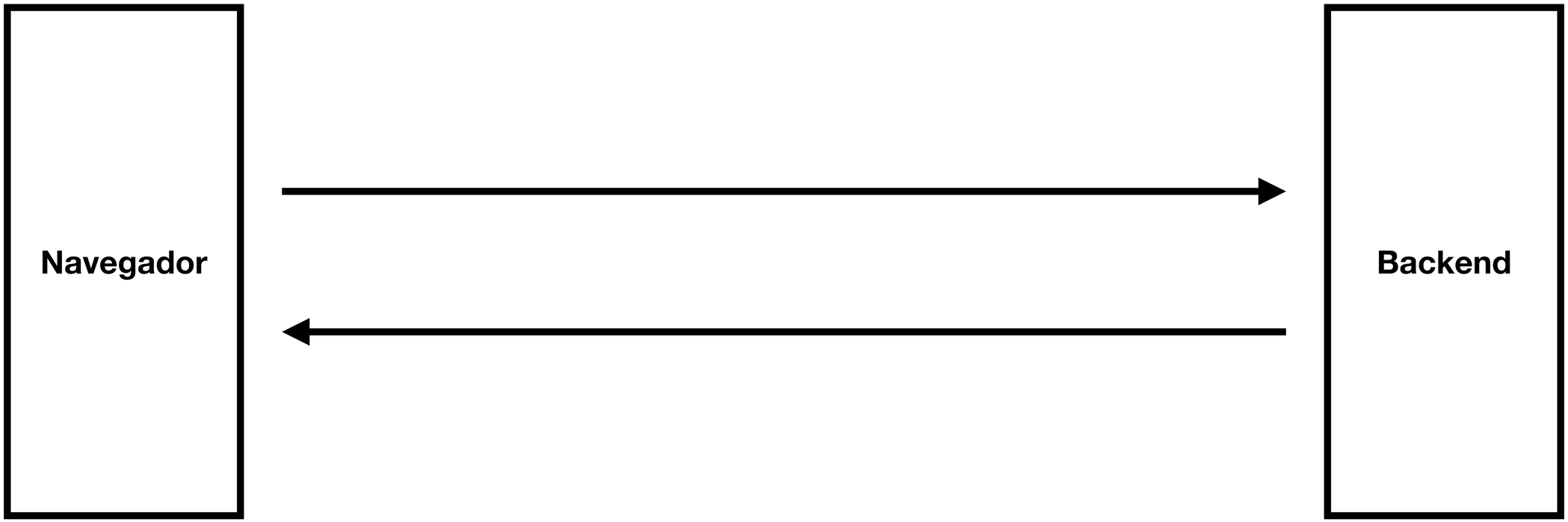
Opción **acoplada a la UI**:

- React Server Components:

```
async function Note({ id }) {
  const note = await db.notes.get(id);
  return (
    <div>
      <Author id={note.authorId} />
      <p>{note}</p>
    </div>
  );
}

async function Author({ id }) {
  const author = await db.authors.get(id);
  return <span>By: {author.name}</span>;
}
```

Carga de datos



Acoplada a la UI

Se ve que parte de la UI debe cambiar, y el servidor envía solo el HTML (o JSX) necesario

Carga de datos

Desafío comunes

- Seguridad: Un nodo es cargado independiente de su padre
  - Solución: Repetir autorización, middlewares en los nodos
- Performance real: Problema 1 + N
  - Solución: Batching, por ejemplo, con dataloader
- Performance percibido: No esperar que todo esté cargado
  - Solución: Streaming de la respuesta (demo realizado a mano)

Estado complejo

Estado complejo

¿Cómo modelo estado complejo?

Hay un punto donde:

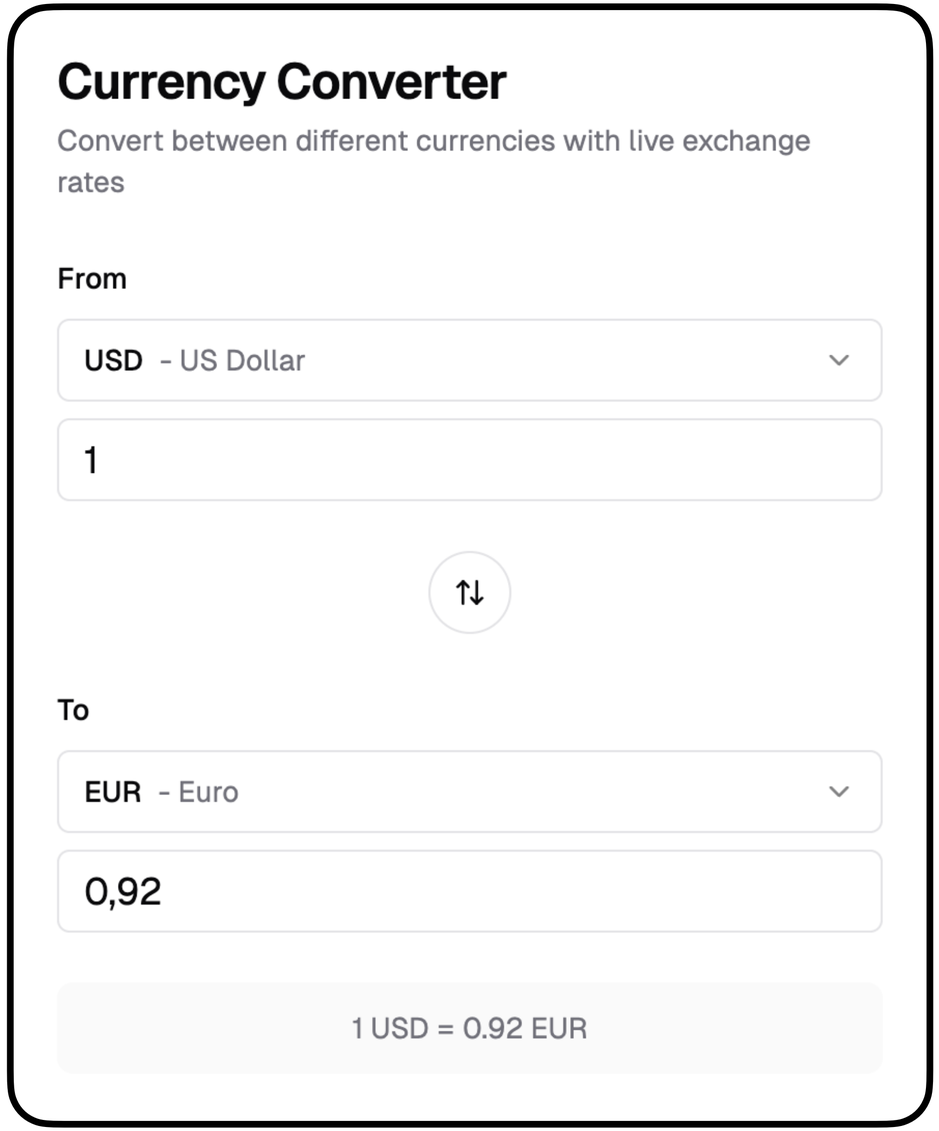
- Usar solo primitivos se vuelve un anti-patrón
- Es fácil perder referencia al valor actualizado
- Sin mucho conocimiento, se usan patrones que degradan el performance

Estado complejo

¿Qué hacemos con estado relacionado?

Ejemplo:

- https://v0-currency-converter-one.vercel.app/
- https://github.com/benjavicente/react-currency-converter-v0



Estado complejo

Lifecycle común de una libraría de UI como React:

1. Transición de estado (cambiamos algo)
2. Actualización (derivados, “correr componentes”)
3. Renderización (actualizar efectivamente el UI)
4. Sincronización (effects a la UI) ← Suele ser antipatrón actualizar estado aquí

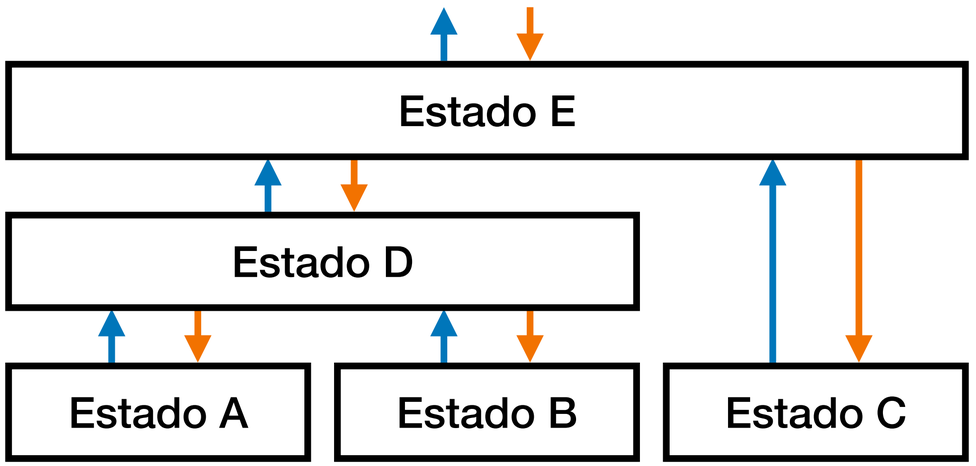
La actualización de estado debe ocurrir en el paso 1 o 2.

Estado complejo

Además, es recomendado siempre componer estado con:

- Setters (o actions)
- Getters de una “caja negra”

Librerías entregan diferentes interfaces, lo importante es **componer** estado.



Componentes con estilos reutilizables

Componentes con estilos reutilizables

Los componentes parten simples…

```
<DetailsCard
    name="Web Avanzado"
    topic="Componentes"
/>
```

Componentes con estilos reutilizables

Los componentes parten simples…

Pero al ser reutilizados salen
nuevas formas de personalizarlo…

¿Como evitamos este anti-patrón?

Composición

```
<DetailsCard
    color="green"
    semester="2025-1"
    name="Web Avanzado"
    topic="Componentes"
    photo="img/wa.png"
    notifications={1}
    marginTop={4}
    padding={2}
/>
```

Componentes con estilos reutilizables

Composición

Más código, pero se entiende más

¿Qué hacemos cuando
queremos personalizar estilos?

```
<DetailsCard.Root
    color="green"
    marginTop={4}
    padding={2}
>
  <DetailsCard.Banner photo="img/wa.png" />
  <DetailsCard.Header
    name="Web Avanzado"
    semester="2025-1"
  />
  <DetailsCard.Info
    topic="Componentes"
    notifications={1}
  />
/>
```

Componentes con estilos reutilizables

Composición

Más código, pero se entiende más

¿Qué hacemos cuando
queremos personalizar estilos?

¿Soluciones como
Tailwind son suficiente?

```
<DetailsCard.Root
    color="green"
    className="mt-4 p-2"
>
  <DetailsCard.Banner photo="img/wa.png" />
  <DetailsCard.Header
    name="Web Avanzado"
    semester="2025-1"
  />
  <DetailsCard.Info
    topic="Componentes"
    notifications={1}
  />
/>
```

Componentes con estilos reutilizables

- ¿Qué pasa si los estilos se reordenan (cargados en diferentes momentos)?
- ¿Cómo establecemos que estilos que se pueden sobre-escribir?
- ¿Soluciones como Tailwind son suficiente?

Caso de Tailwind: tailwind-merge como “solución parche mágica”

CSS tiene una solución nativa, que es usada comúnmente por los frameworks

Componentes con estilos reutilizables

Especificidad de CSS llega hasta cierto punto, se necesitan son capas:

CSS Cascade Layers

Especifican el orden en
la que se aplican los estilos

Así podemos establecer
estilos que pueden o
no deben ser sobrescritos.

```
@layer theme, base, components, utilities, fixed;
@layer theme { :root { --color-red: red } }
@layer base { * { min-width: 0 } }
@layer components {
  .details-card { padding-top: 2rem }
}
@layer fixed {
  .details-card { display: flex }
}
```

Componentes con estilos reutilizables

Propuestas de CSS a tener a consideración en el futuro:

- Container (style) queries y CSS Conditionals (`if`)
- Relative colors y `contrast-color`
- Masonry layout e Item Flow

Algunas ya están siendo implementadas en los navegadores

Devtools: performance pannel + teléfono

Devtools: performance pannel + teléfono

Es sitios más B2C, es clave probar en un teléfono real, tanto para probar la experiencia de usuario real e identificar errores no emularles en un PC.

- Android: Chrome + Teléfono con configuración de desarrollador
- iOS: Activar web inspector y modo de desarrollador en Safari (Guía Apple)

En Android, también se puede hacer port-forwading e Screencast, permitiendo inspeccionar elementos de una página servida desde localhost.

Devtools: performance pannel + teléfono

Performace pannel combina la información importante del rendimiento y estado de varios aspectos de una página web, en un timeline de eventos. Combina:

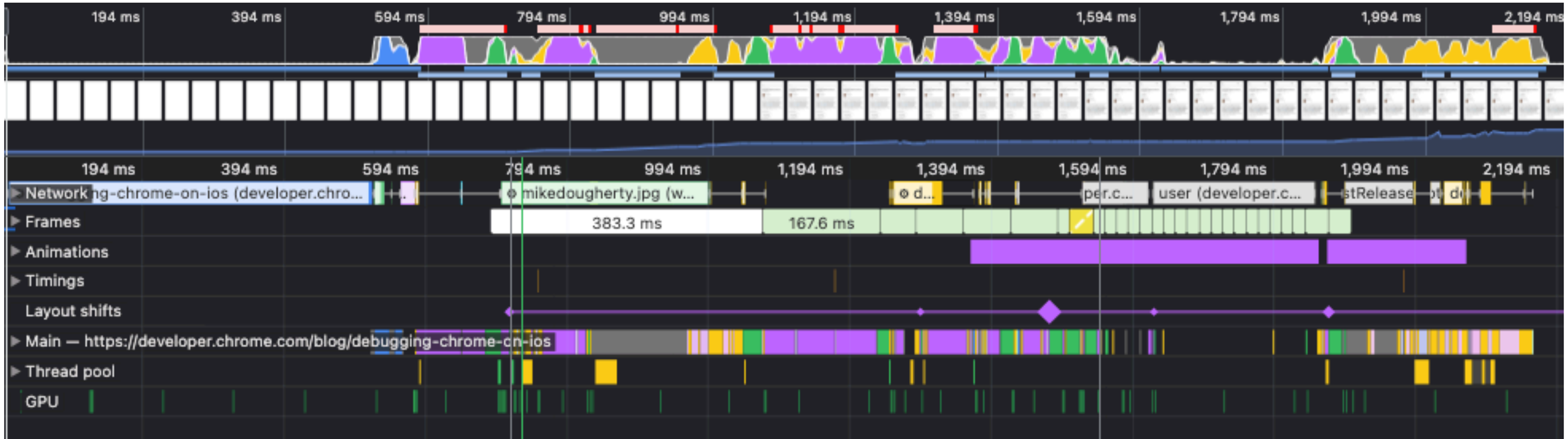
- FPS, uso de CPU, y uso de memoria
- Cada frame, permitiendo ver el estado de la página en cada momento
- Requests, que puede ser inspeccionadas en detalle en network pannel
- Animaciones, layout shifts e interacciones de usuario
- Flame chart del todo lo ejecutado (tasks del event loop)

¡Y ***mucho*** más! La página de referencia es bien larga

Devtools: performance pannel + teléfono

La UI es compleja, para navegarla hay que tener en cuenta que:

- Son varios acordeones, algunos muestran elementos 2D
- Drag and Drop para moverse dentro de un cuadro, scroll para mover timeline



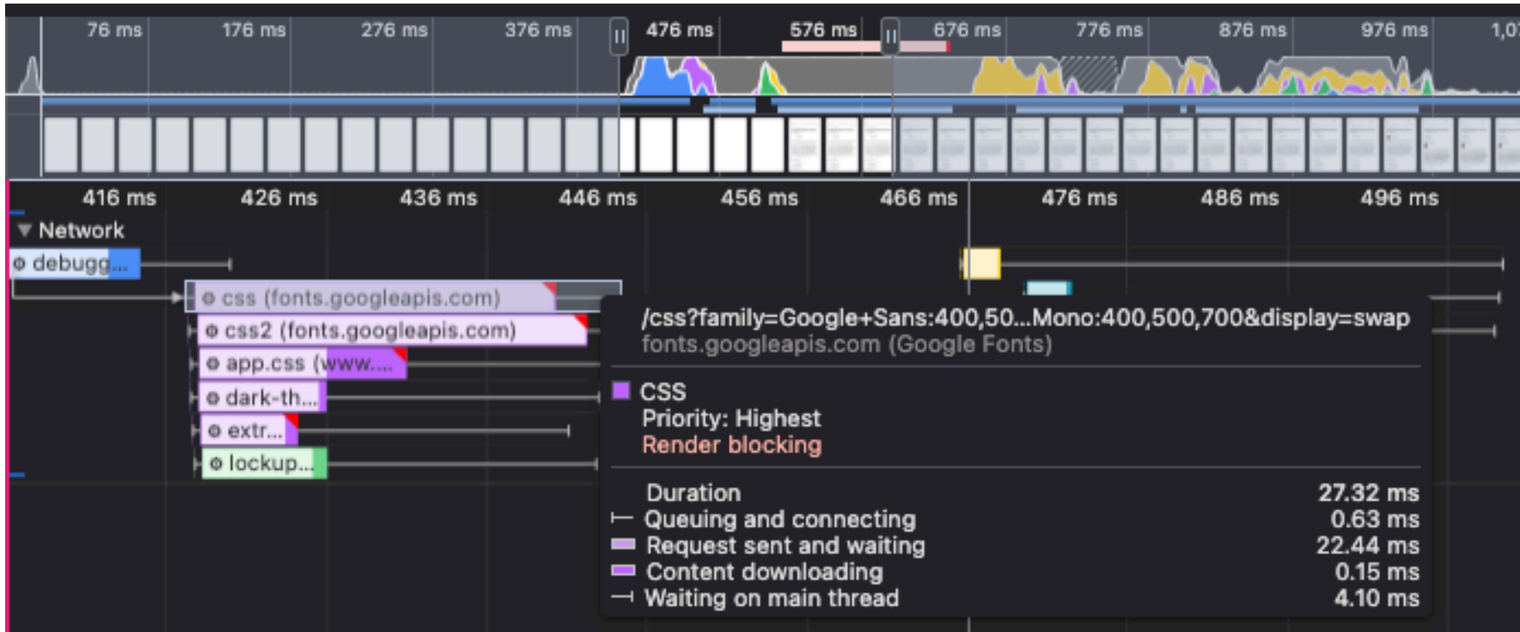
Devtools: performance pannel + teléfono

Network pannel: Qué requests realizó la página

Se marcan con rojo requests que son críticas para el renderizado

Para esas requests, es clave estar realizando ciertas optimizaciones!

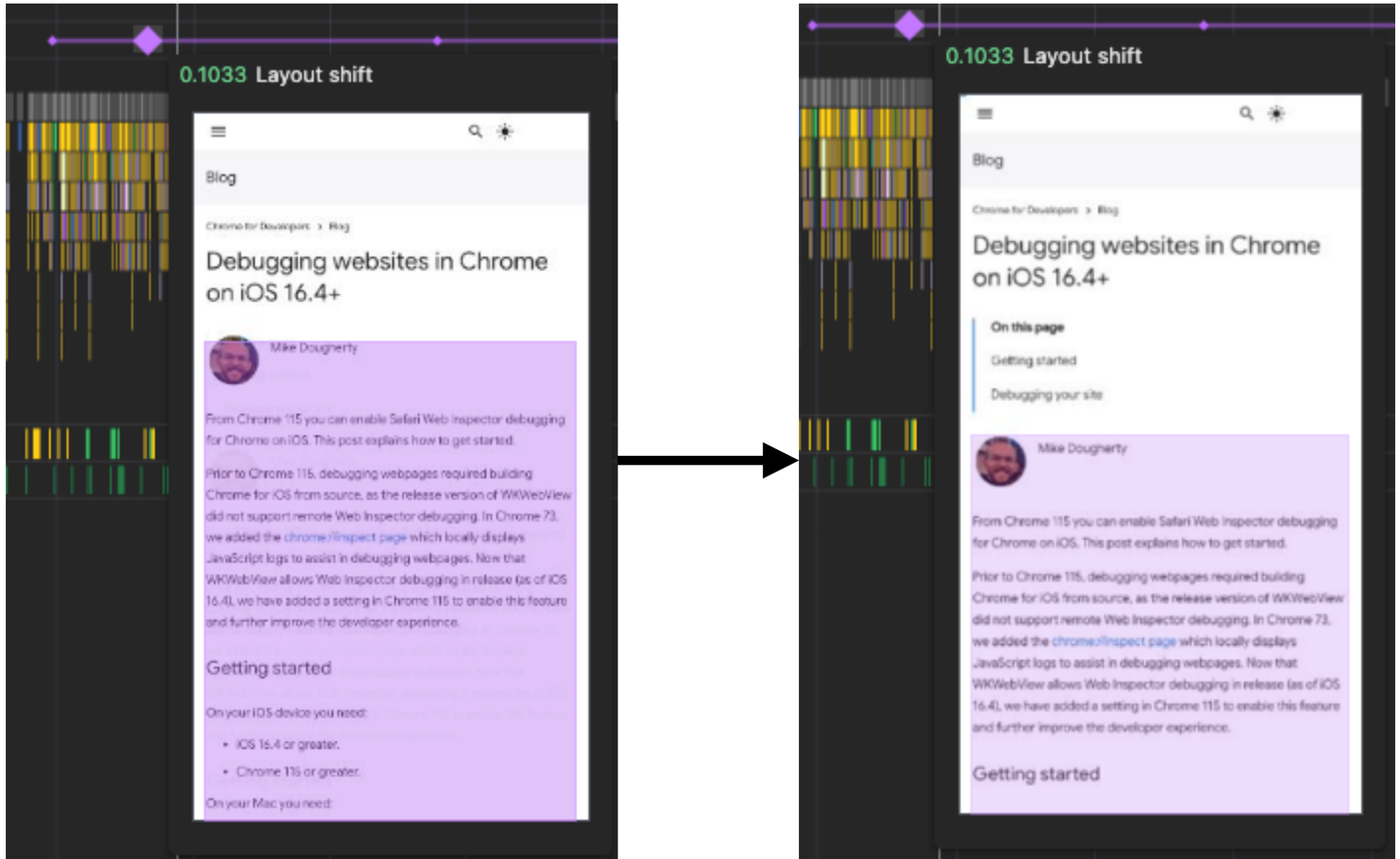
Muy útil para debuguear WebSockets



Devtools: performance pannel + teléfono

Layouts Shifts: Qué cambios de layout pasaron en la página

Bueno para entender cambios bruscos que empeoran la experiencia de usuario.



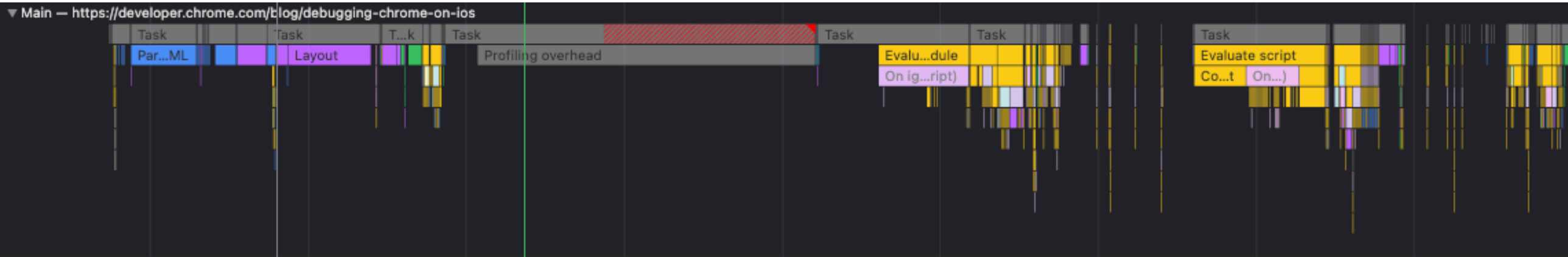
Devtools: performance pannel + teléfono

Flame graph (main): Qué código y tareas realizo el navegador

Cada barra horizontal es una tarea o llamado de función (usualmente amarillo)

La longitud horizontal es el tiempo, la aplicación vertical es el stack de llamadas

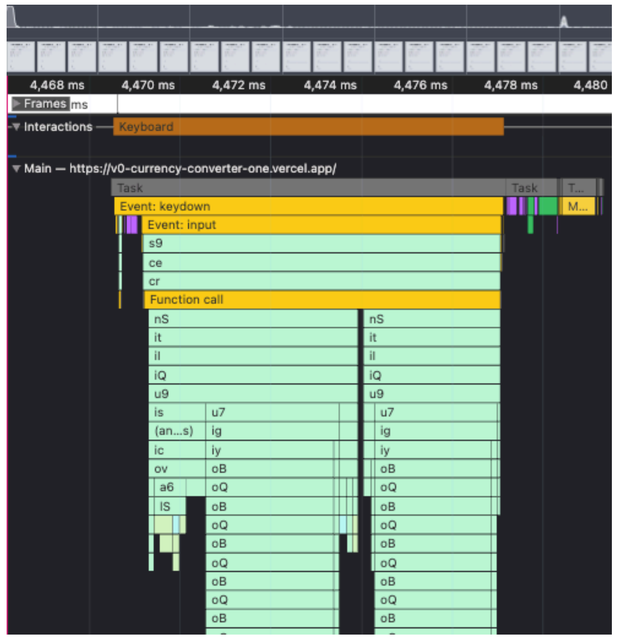
¡Ojo que cada task bloquea la interactividad del navegador hasta que termine!

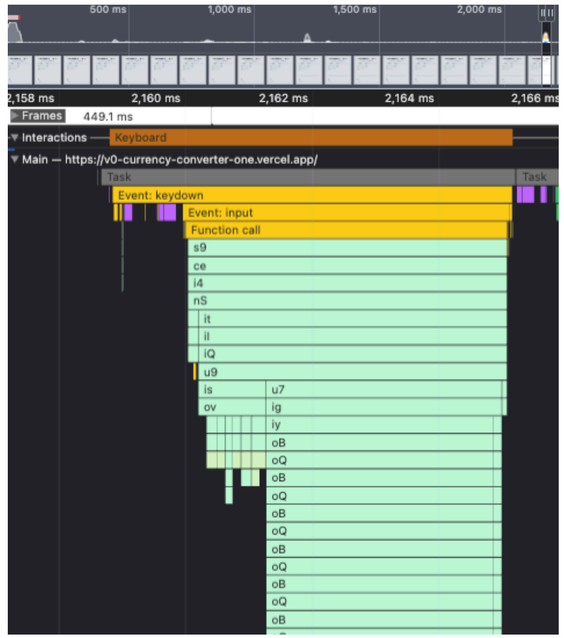


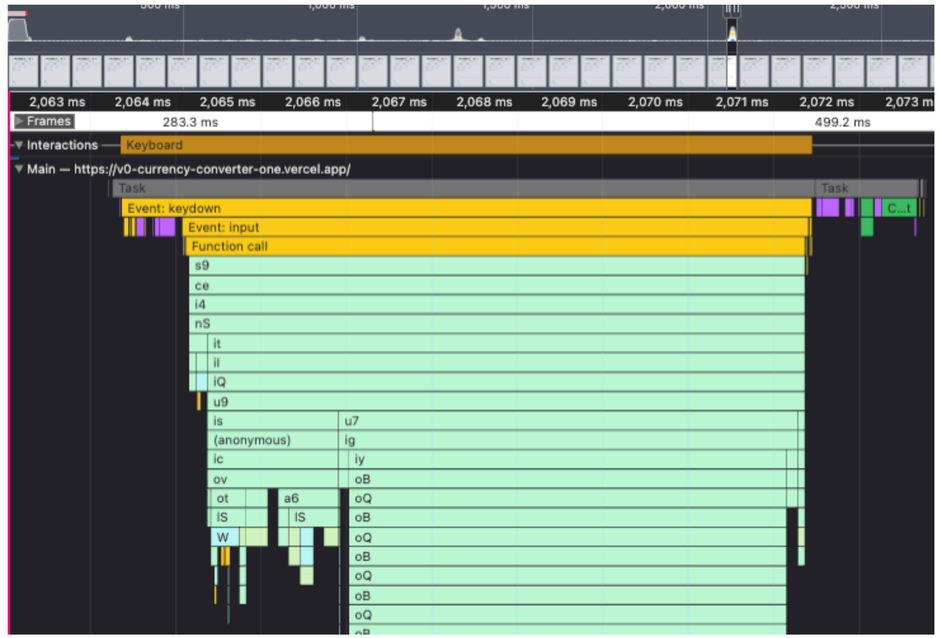
Devtools: performance pannel + teléfono

Caso practico: Slide Estado y demo de currency converter

¿Pueden ver cual es cada uno solo con ver el stacktrace?
¿Qué son las funciones que ocupan más tiempo?







No son más de 10ms, pero este es el caso más simple

Devtools: performance pannel + teléfono

Otras herramientas interesantes fuera de performance pannel:

- Recorder: grabar interacciones y obtener el código en puppeteer
- Layers: ver le página en 3D
- Sensors: cambiar ubicación y orientación
- Animations: jugar con timeline de animaciones