

# SVELTE

\$: Benja Vicente

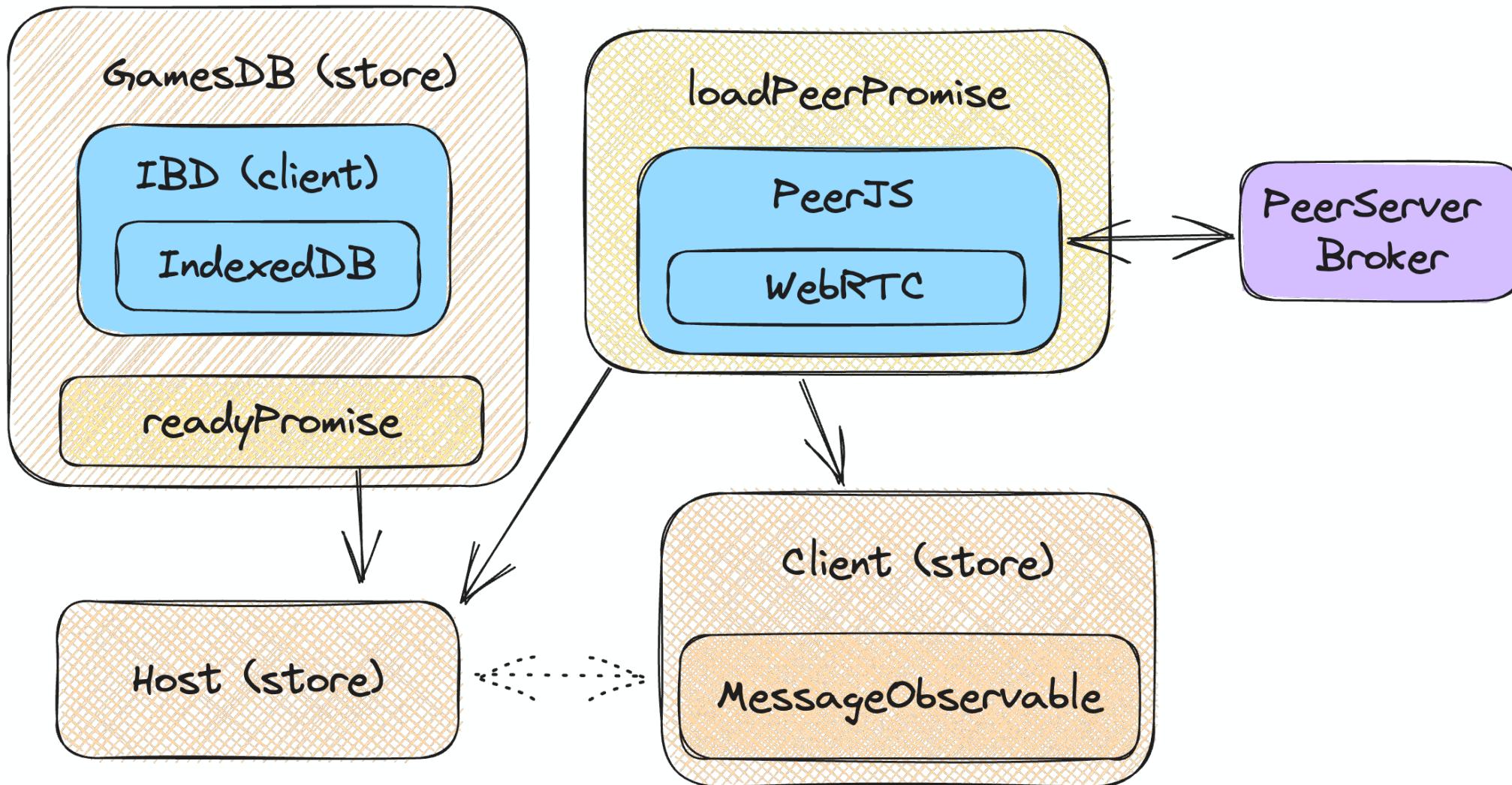
\$: Jose Antonio Castro

\$: José Madriaza



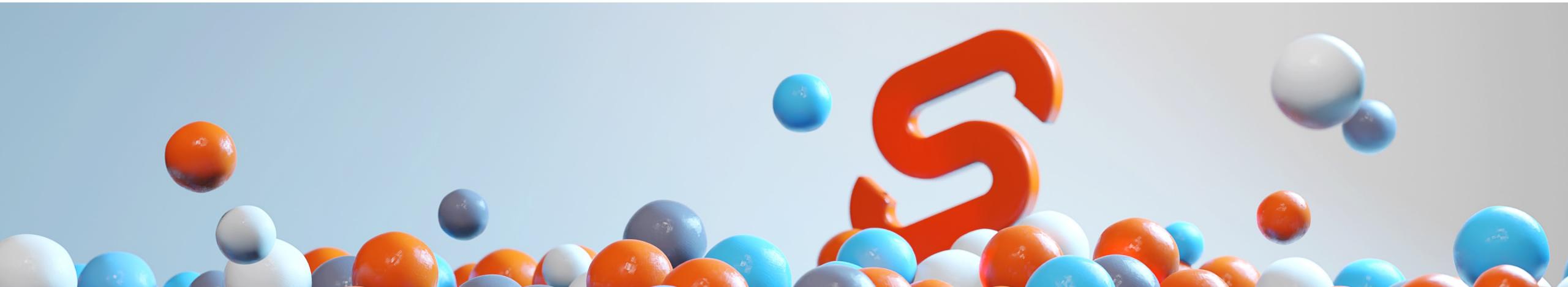
**Demo:  
Wordle Competitivo**

# Estado de la aplicación



# Svelte

- Domain Specific Language (DLS) para componentes
- Busca unir **performance** y **agilidad**
- Aprovecha fuertemente un proceso de **compilación**
- Pensado para **visualizaciones** (fácil de añadir animaciones)
- Aprovecha mucho **contratos framework-agnostic**



# Componentes

```
<script lang="ts">
  import { page } from '$app/stores';
  import ClientGameScreen from './ClientGameScreen.svelte';
  import { loadPeerPromise } from '$lib/peer';

  const queryParamId = $page.url.searchParams.get('game');
</script>

<main>
  #await loadPeerPromise
    <div>Loading</div>
  {:then peer}
    <ClientGameScreen {peer} {queryParamId} />
  {:else}
    <h1>Joining game</h1>
  {:if}
</main>
```

```
<script lang="ts">
  import type { Peer } from '$lib/peer';
  import GameDisplay from './GameDisplay.svelte';
  import { createGameClientStore } from './client';

  export let peer: Peer;
  export let queryParamId: string | null = null;

  const client = createGameClientStore(peer);
  if (queryParamId) client.connect(queryParamId);

  let connectionInputValue = '';
</script>

{:if $client.status === 'establishing-connection'}
  <div>Establishing connection</div>
{:else if $client.status === 'ready-to-connect'}
  <form on:submit|preventDefault={() => client.connect(connectionInputValue)}>
    <input type="text" name="game" bind:value={connectionInputValue} />
    <button type="submit">Join</button>
  </form>
{:else if $client.status === 'connected'}
  <GameDisplay
    gameState={$client.gameStateObservable}
    on:changeName={(event) => client.changeName(event.detail)}
    on:guess={(event) => client.guessWord(event.detail)}>
  </GameDisplay>
{:else}
  <h1>Joining game</h1>
{:if}</if>
```

# Partes de un componente

```
<script context="module">
    // Código disponible a nivel de módulo
</script>

<script lang="ts">
    // Código especial para el componente
</script>

<!-- Cuerpo del componente --&gt;

&lt;style lang="scss"&gt;
    /* Estilos locales */
&lt;/style&gt;</pre>
```

# Props

```
<script lang="ts">
  export let peer: Peer;
</script>

<GameDisplay
  gameState={$client.gameStateObservable}
/>
```

# Lógica en los templates

# Título : medio / Título

```
{#each $gameState.self.lastGuess.result as result, index}  
...  
{/each}
```

```
{#if $client.status === 'ready-to-connect'}  
...  
{:else if $client.status === 'connected'}  
...  
{/if}
```

```
<main>  
  {#await loadPeerPromise}  
    <div>Loading</div>  
  {:then peer}  
    <ClientGameScreen {peer} {queryParamId} />  
  {/await}  
</main>
```

# Bindings

```
<script lang="ts">
  let connectionInputValue = '';
</script>

<input bind:value={connectionInputValue} />
```

**Pasar valores de componente hijo a padre**

# Eventos

```
< on:eventname|modifiers={handler} />
```

```
<script lang="ts">
  import { createEventDispatcher } from 'svelte';

  const dispatch = createEventDispatcher<{ guess: string }>();
  let guess: string = '';
</script>

<form
  on:submit|preventDefault={() => {
    dispatch('guess', guess);
    guess = '';
  }}
>
  <input bind:value={guess} {disabled} type="text" />
</form>
```

```
<GameDisplay
  gameState={$client.gameStateObservable}
  on:changeName={(event) => client.changeName(event.detail)}
  on:guess={(event) => client.guessWord(event.detail)}
/>
```

# Animaciones

```
<script lang="ts">
  import { flip } from 'svelte/animate';
</script>

<ul>
  {#each players as player (id)}
    <li class="flex gap-2" animate:flip>
      ...
    </li>
  {/each}
</ul>
```

**Animaciones faciles con animate:flip**

```
{#key $gameState.self.lastGuess?.time}
<div>
  {#if $gameState.self.lastGuess}
    {#each lastGuess as result, index}
      <span in:receive={{ key: index }} />
    {/each}
  {/if}
</div>
{/key}

{#key $gameState.self.lastGuess?.time}
<div>
  {#each guess as _, index}
    <span in:fade out:send={{ key: index }} />
  {/each}
</div>
{/key}
```

# Estado

## Stores

### Reactividad Compilada

```
<script>
  let counts = new Array(5).fill(0).map(() => new Array(5).fill(0))
</script>

{-#each counts as row, rowIndex}
  <div>
    {-#each row as count, colIndex}
      ← Se actualiza todo lo que depende de counts →
      <button on:click={() => counts[rowIndex][colIndex] += 1}>
        {count}
      </button>
    {-/each}
  </div>
{/each}
```



Rich Harris  
@Rich\_Harris

...

I want to do something more like this, which we can do because WE'RE A COMPILER, MOFOS

12:17 p. m. · 30 oct. 2018

```
<script>
  import { writable } from 'svelte/store';

  function createCount() {
    const { subscribe, set, update } = writable(0);

    return {
      subscribe,
      increment: () => update(n => n + 1),
      decrement: () => update(n => n - 1),
      reset: () => set(0)
    };
  }

  const count = createCount();
</script>

<h1>The count is {$count}</h1>

<button on:click={count.increment}>+</button>
<button on:click={count.decrement}>-</button>
<button on:click={count.reset}>reset</button>
```

# Store contract

```
type UnsubscribeFn = (() => void) | { unsubscribe: () => void }

interface Store<T> = {
    // subscribe debe llamar a subscription inicialmente y por cada cambio
    subscribe: (subscription: (value: T) => void) => UnsubscribeFn,
    // para habilitar $store = ...
    set?: (value: T) => void
    // una store también puede tener atributos adicionales (actions)
}
```

**Las stores **no** dependen de un estado global ni de svelte**

# Store son compatibles con RxJS

```
function createMessageObservable(conn: Connection) {
  return (
    fromEventPattern(
      (handler) => conn.on('data', handler),
      (handler) => conn.off('data', handler),
    ) as Observable<MessageToPlayer>
  ).pipe(
    filter(({ type }) => type === 'game-state'),
    shareReplay(1),
  );
}
```

```
<div style:background-color={$gameState.self.representation.color}>
  {getEmoji($gameState.self.representation.emojiIndex)}
</div>
```

```
// Notificar que está listo en el primer mensaje
gameStateObservable.pipe(first()).subscribe(() => notifySubscribers());
```

# Actions contract

```
interface Action<Params> {  
    (node: HTMLElement, parameters: Params) => {  
        update?: (parameters: Params) => void,  
        destroy?: () => void  
    }  
}
```

Las actions **no** dependen de svelte

# Actions

```
<ul use:dndzone={{ items: wordsWithIdAttr, flipDurationMs }}>  
  ...  
</ul>
```

## isaacHagoel/**svelte-dnd-action**

An action based drag and drop container for Svelte



15  
Contributors

663  
Used by

1k  
Stars

64  
Forks

## isaacHagoel/**solid-dnd-directive**

A directive based drag and drop container for solid-js



28  
Used by

59  
Stars

2  
Forks



```
{  
  "peerDependencies": {  
    "solid-js": "^1.0.0"  
  },  
  "dependencies": {  
    "svelte-dnd-action": "0.9.11"  
  }  
}
```



# Conclusiones

- Magia del **compilador** ayuda a desarrollar muy ágilmente
- Los **contratos** permiten estar pensando JS y no en frameworks
- Tiene rough edges<sup>1</sup> pero va **bien encaminado**

<sup>1</sup> Por ejemplo, las stores solo se pueden usar con \$ si están definidas en el script del componente