Signals & Streams

THINKING DECLARATIVELY

Presented by Mischa Koischwitz

akquinet tech@spree

Why are Signals taking over?

Many modern frontend frameworks (Angular, Solid.js, Preact, Svelte) are adopting signals because they offer a cleaner, more predictable way to handle UI state.

PREDICTABILITY

They offer more predictable state updates.

DECLARATIVENESS

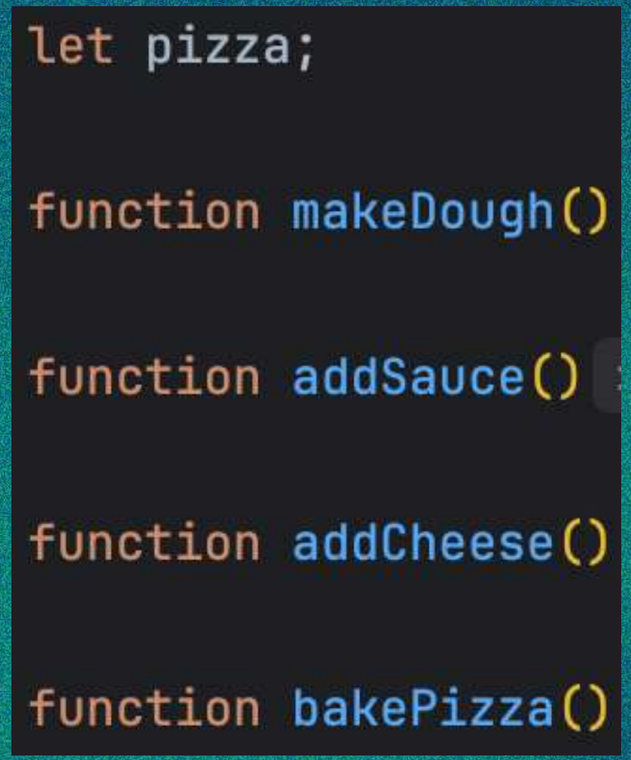
They remove imperative complexity from event-driven UIs.

Imperative Code

Describes actions.

State relationships are hidden (implicit).

Requires keeping track of variables.



Declarative Code

Describes states & relationships explicitly.

Defines full behavior of variables at time of their declaration

```
const dough = 'pizza-dough';
const sauce = 'tomato';
const cheese = 'mozzarella';
const pizza : string  = dough + sauce + cheese;

const oven = 'stone oven';
const bakedPizza : string  = pizza + oven;
```

Introducing Signals

THINKING DECLARATIVELY

What are Signals?

A signal is a reactive value that automatically updates when its dependencies change.

	A	B	C	D
1	item	price	quantity	total
2	apples	0.69	4	2.76
3	bananas	0.39	6	2.34
4	cantaloupes	2.49	2	4.98
5				10.08

Source: Rich Harris - Rethinking Reactivity

Signals in Code

Writable Signals

```
const variable : WritableSignal<number> = signal(1);
variable.set(3);
console.log('variable', variable());
```

Computed Signals

```
const variablesTimesTwo : Signal<number> = computed(() : number => {
  return 2 * variable();
});
```

Side effects

```
effect(() : void => {
  console.log('variable', variable());
});
```

Examples

DIVING INTO CODE

Example 1

Keeping track of the current window width and concluding if the user is using a mobile phone.

Non-Signal Approach

Initial Value

```
windowWidth : number  = window.innerWidth;
isMobile : boolean  = window.innerWidth <= 480;
```

Later Adjustments

```
document.addEventListener('resize', () : void  => {
  this.windowWidth = window.innerWidth;
  this.isMobile = window.innerWidth <= 480;
});
```

Signal Approach

isMobile's behavior is fully defined
during initial declaration

```
windowWidth : WritableSignal<number>  = signal(window.innerWidth);
isMobile : Signal<boolean>  = computed(() : boolean  =>
  this.windowWidth() <= 480
);
```

windowWidth is still set imperatively

```
document.addEventListener('resize', () : void  => {
  this.windowWidth.set(window.innerWidth)
});
```

Introducing Streams

Streams let us react to events over time, rather than tracking a single value like signals do.

USER INTERACTIONS

clicks, key presses, mouse movements

SERVER EVENTS

WebSockets, async data fetching

TIME-BASED ACTIONS

delayed events, throttling, debouncing

RxJS Streams in Code

Streams in RxJS are called **Observables**.

Events from the DOM or HTTP endpoints

```
const clickStream$ : Observable<Event>  = fromEvent(document, 'click');
const dataFromEndpoint$ : Observable<Object>  = this.httpClient.get(url);
```

Deriving Streams from Streams

```
const clickCount$ : Observable<number>  = clickStream$.pipe(
  scan(value : number  => value + 1, 0)
);
```

Subscribing to Streams

```
clickStream$.subscribe(event : Event  => {
  console.log('click', event);
});
```

Converting Streams & Signals

Provided out of the box in **Angular**

toObservable()

Converts a Signal into an Observable (Stream).
Lets us react to Signal value changes over time.

toSignal()

Converts an Observable into a Signal.
Lets us track the values of the Stream.

Benefits of this approach

Straightforward Http Handling

```
readonly tabs : Signal<string[]> = toSignal(
  this.httpRequest(), {initialValue: []}
);
```

Less manual subscribing

```
const clickCount : Signal<number> = toSignal(
  clickCount$,
  {initialValue: 0}
);

console.log('click count', clickCount());
```

Deriving values

```
const warningMessage : Signal<'Enough clicking!' | ''> = computed(
  () : 'Enough clicking!' | '' => clickCount() > 10 ? 'Enough clicking!' : ''
);
```

Example 1 - Part 2

Keeping track of the current window width and concluding if the user is using a mobile phone.

Fully Declarative Code

```
readonly windowWidth : Signal<number>  = toSignal(
  fromEvent(window, 'resize').pipe(
    map(() : number  => window.innerWidth)
  ),
  {initialValue: window.innerWidth}
);

readonly isMobile : Signal<boolean>  = computed(() : boolean  => this.windowWidth() <= 480);
```

Behavior of **windowWidth** and **isMobile**
is fully defined during initial declaration

Example 2

A tab-bar component that lists tabs from left to right.

Fully responsive, including:

- **Responsive tab width**
- **Hiding tab titles if necessary**
- **Handling overflowing tabs**

Imperative Approach

Initial Value

```
tabWidth : number  = window.innerWidth / this.tabs.length;
visibleTabsLimit : number  = Math.floor(window.innerWidth / this.tabWidth);
```

Later Adjustments

```
document.addEventListener('resize', () : void  => {
  let responsiveWidth : number  = window.innerWidth / this.tabs.length;
  if (responsiveWidth < 80) responsiveWidth = 44;
  if (responsiveWidth > 280) responsiveWidth = 280;
  this.tabsWidth = responsiveWidth;
  this.visibleTabsLimit = Math.floor(window.innerWidth / this.tabs.length);
});
```

Declarative Approach

Calculating tab width

```
readonly tabWidth : Signal<number> = computed(() : number => {
  const responsiveWidth : number = this.windowWidth() / this.tabs().length;
  if (responsiveWidth < 80) return 44;
  if (responsiveWidth > 280) return 280;
  return responsiveWidth;
});
```

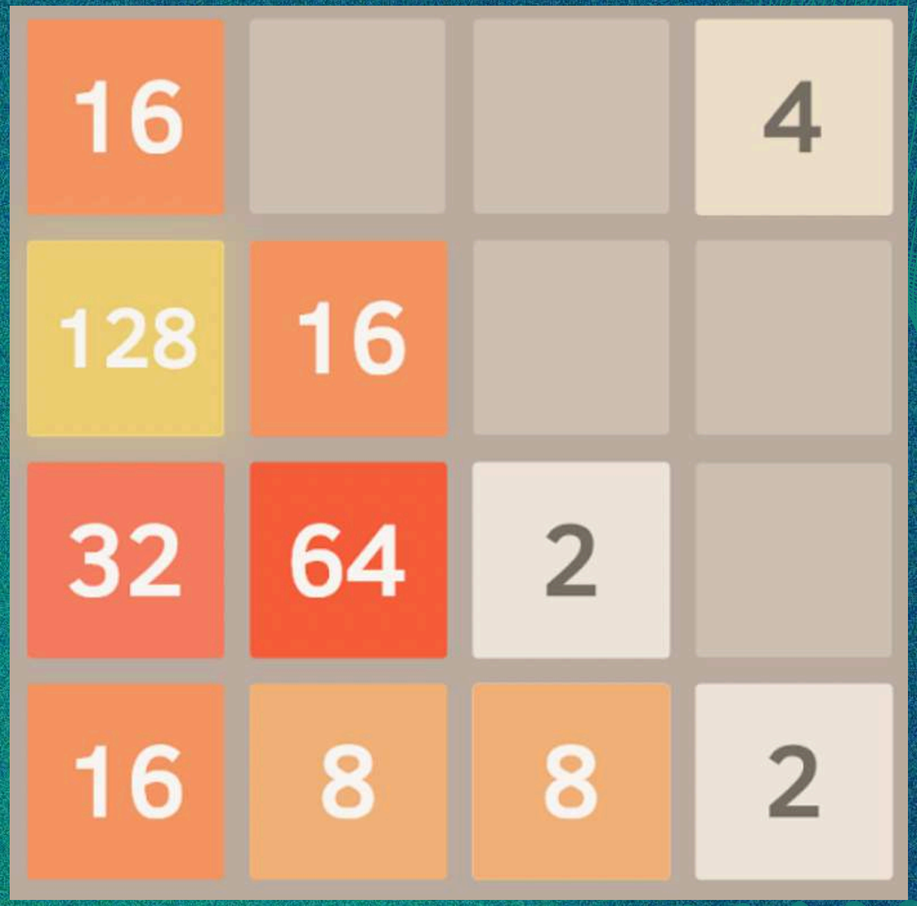
Maximum tabs that can fit

```
readonly visibleTabsLimit : Signal<number> = computed(() : number =>
  Math.floor(this.windowWidth() / this.tabWidth())
);
```

Example 3

The game 2048 - a sliding puzzle game where players merge numbered tiles by powers of two to reach 2048.

- Allowing multiple input types
- Assuring the game state stays up to date



Imperative Code

Initial Value

```
grid : WritableSignal<number[][]> = signal(initialGrid);
```

Later Adjustments

```
document.addEventListener('keypress', (event: KeyboardEvent) : void => {
  if (!['w', 'a', 's', 'd'].includes(event.key.toLowerCase())) {
    return;
  }
  const key = event.key.toLowerCase() as 'w'|'a'|'s'|'d';
  if (['w', 'a', 's', 'd'].includes(key)) {
    const updatedGrid : number[][] = moves[key](this.grid());
    this.grid.set(updatedGrid);
  }
});
```

```
export const moves = {
  w: slideGridUp,
  a: slideGridLeft,
  s: slideGridDown,
  d: slideGridRight
};
```

Declarative Code

Defining valid move inputs

```
readonly moveInputs : Observable<'w' | 'a' | 's' | 'd'> = fromEvent<KeyboardEvent>(document, 'keyup').pipe(
  map(event : KeyboardEvent => event.key.toLowerCase()),
  filter((key : string): key is 'w'|'a'|'s'|'d' => ['w', 'a', 's', 'd'].includes(key))
);
```

Initial grid & grid behavior

```
readonly grid : Signal<number[][]> = toSignal(
  this.moveInputs.pipe(
    scan(
      (currentGrid : number[][] , direction : 'w' | 'a' | 's' | 'd' ) : number[][] => moves[direction](currentGrid),
      initialGrid
    )
  ), {initialValue: initialGrid}
);
```

Common Pitfalls

WHEN WORKING WITH SIGNALS

Treating Signals as mutable state

```
clickCount : WritableSignal<number> = signal(0);

someClickEvent() : void { Show usages new *
  this.clickCount.set(this.clickCount() + 1);
}
```

Assigning state in effect()

```
const clickCount : WritableSignal<number> = signal(0);

const clickCountTimesTwo : WritableSignal<number> = signal(0);

effect(() : void => {
  clickCountTimesTwo.set(2 * clickCount());
});
```

Instead use computed()

```
const clickCount : WritableSignal<number> = signal(0);

const clickCountTimesTwo : Signal<number> = computed(() : number =>
  2 * clickCount()
);
```

Advanced Example

EXPANDING OUR TAB BAR WITH “NEW TAB” FUNCTIONALITY

Waiting for an HTML element to load

Without relying on lifecycle hooks

```
readonly newTabButton : Signal<ElementRef<any> | undefined> = viewChild<ElementRef>('newTabButton');

readonly newTabButtonAvailable : Observable<any> = toObservable(this.newTabButton).pipe(
  filter(elementRef : ElementRef<any> | undefined => !!elementRef),
  map(elementRef : ElementRef<any> => elementRef.nativeElement)
);
```

Let’s use this approach for our tab-bar!

Adding an action to the HTML element

In this case: adding an item to a string array

```
readonly newTabButtonClick : Observable<(tabs: string[]) => string[]> = this.newTabButtonAvailable.pipe(
  switchMap(element : HTMLButtonElement => fromEvent(element, 'click')),
  map(() : (tabs: string[]) => string[] => (tabs: string[]) : string[] => [...tabs, 'new tab'])
);
```

Defining our tab logic

Combining initial HTTP request & adding of new tabs

```
readonly tabs : Signal<string[]> = toSignal(
  this.httpGetTabs().pipe(
    switchMap(initialTabs : string[] =>
      this.addNewTabEvent.pipe(
        startWith(() : string[] => initialTabs),
        scan(
          (tabs : string[] , action : ((tabs: string[]) => string[]) | (() => s... ) : string[] => action(tabs),
          initialTabs
        ),
      )
    )
  ), {initialValue: []}
);
```

This now resembles a reducer-pattern!

Code along

CREATING A DRAG & DROP FEATURE

github.com/VengsHub/bob-konf-2025

Get In Touch With Me

Email Address

mischa@koischwitz.de

Github

VengsHub