

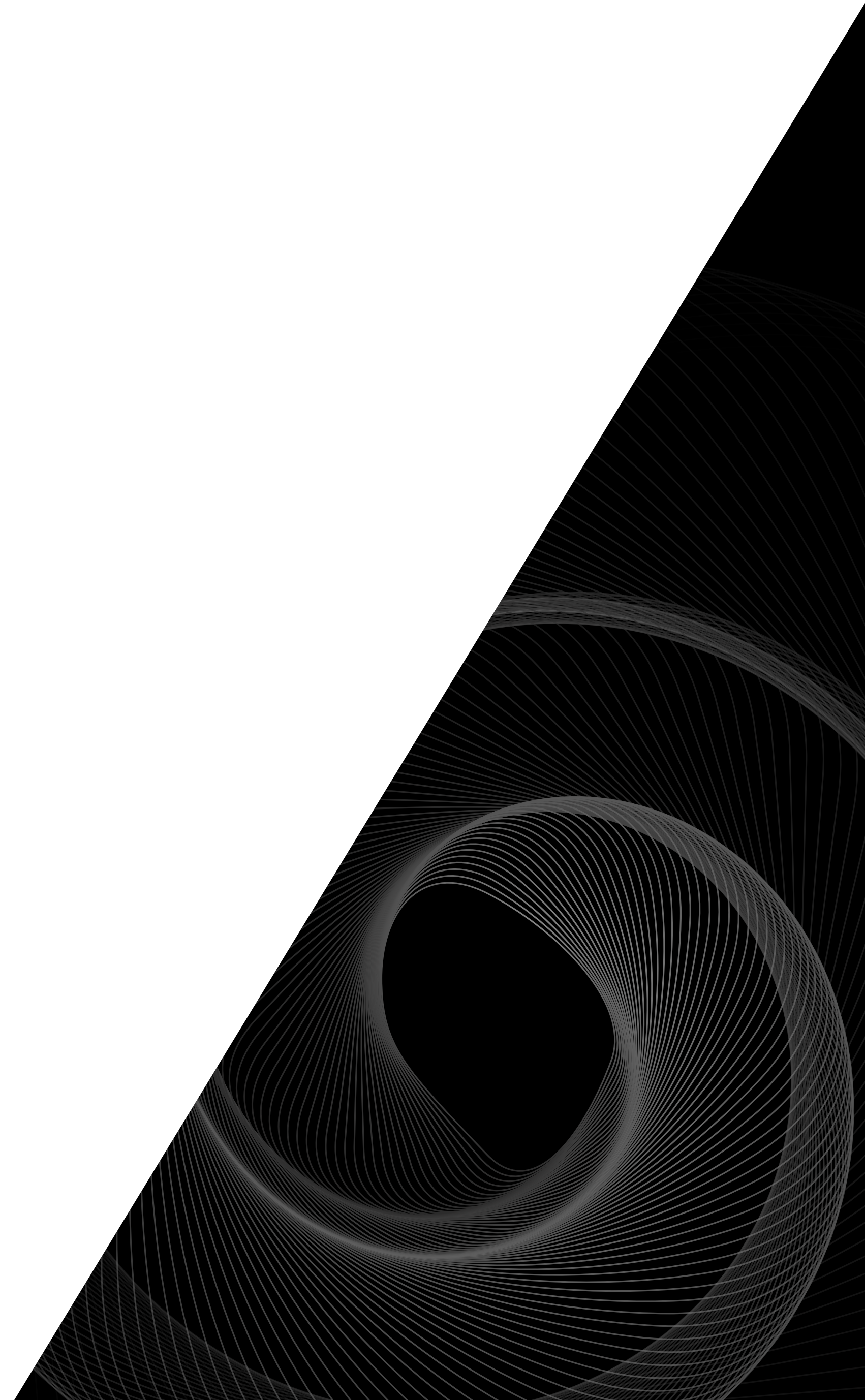
THE POWER OF COMPONENTS*

*** AND DESIGN SYSTEMS**

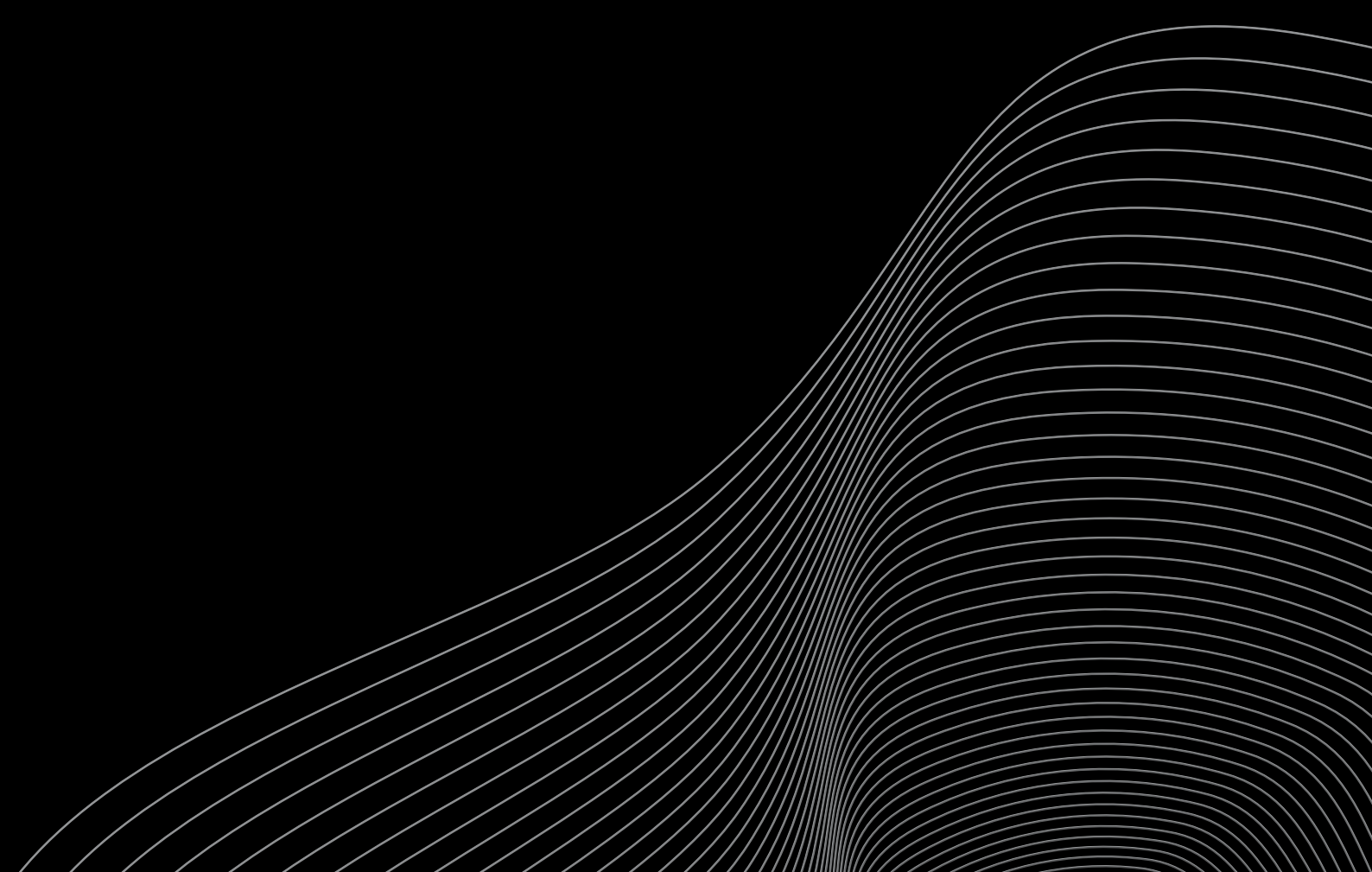
Lode Vanhove

Solution Architect

React Native & iOS Developer



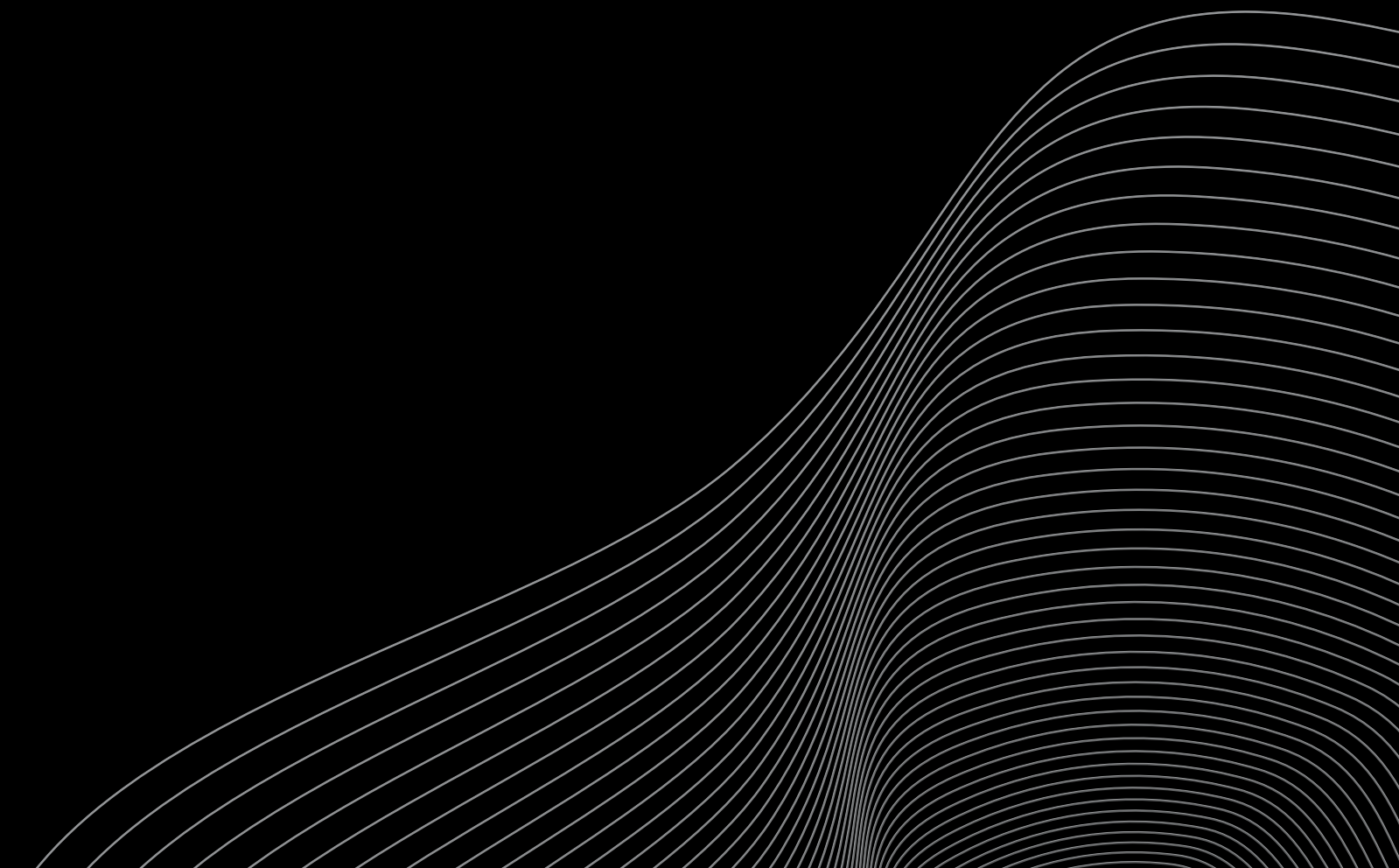
This talk is about how a **component-based approach** can close the gap between design & development (and positively impact both at the same time).



The Power of Components / Overview

Topics

- Cross-platform Development
- Design Systems
- Components
- Tooling



CROSS-PLATFORM DEVELOPMENT

Now & then



Cross-platform Development / Now & Then

The "Dark Ages"

- Phonegap/Cordova, Ionic, ...
 - Hybrid (using webviews)
 - Limited functionality
 - Bad UX/performance

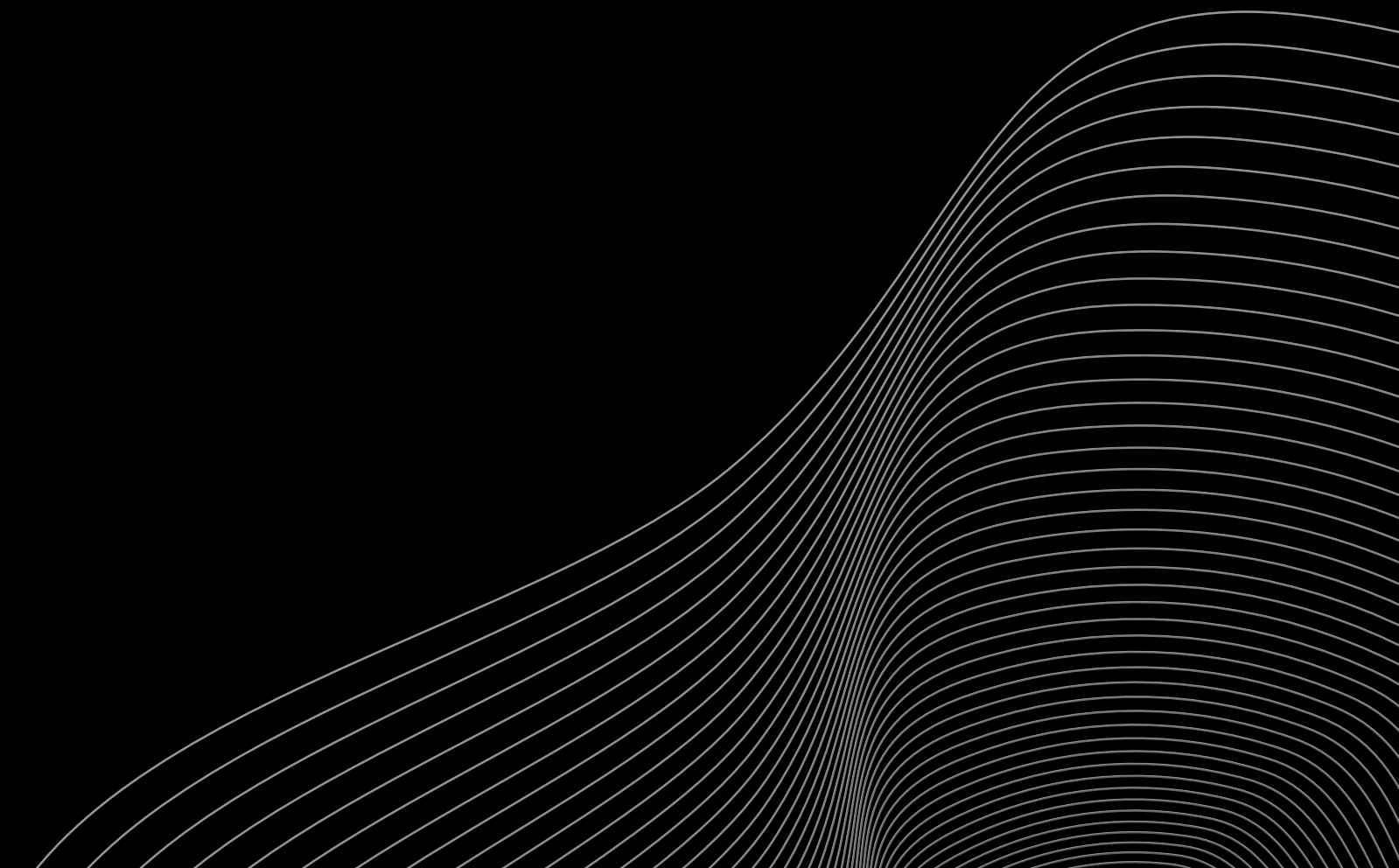
You end up sacrificing a lot for the sake of a common codebase.

- Xamarin
 - Forms = limited
 - Native = no sharing of UI code
- Design process not as streamlined
 - Screens often designed separately
 - Cluttered mess of information
 - Hard to maintain set of assets & guidelines

Cross-platform Development / Now & Then

The "Modern Days"

- The advent of component-based frameworks
 - React (Native)
 - Vue.js
 - Angular 2 (or was it 4?)
- A "bottom-up" design approach
 - Focus on creating a consistent set of assets that can be easily combined & reused
- The rise of new tools like
 - Sketch
 - Zeplin



DESIGN SYSTEMS

A shared vocabulary



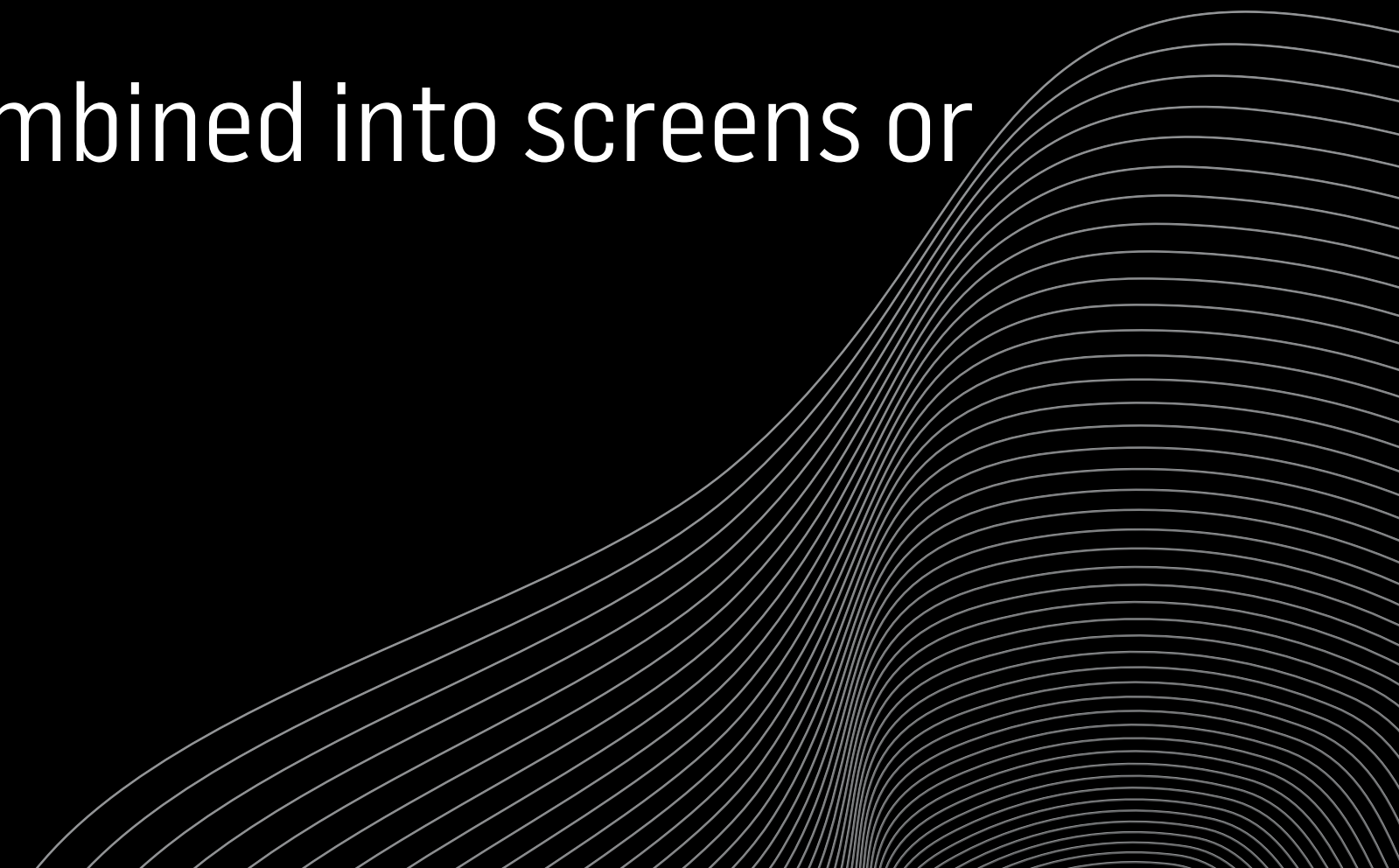
Design Systems / A shared vocabulary

What is it?

- Centralised **source of "design truth"**
 - Contains colours, typography, icons, layout, grids, interface elements, ...
 - Often accompanied by a set of rules and guidelines
 - Acts as a **shared vocabulary**
- The **consistency** it brings enables:
 - **less user confusion**
 - **a faster design process**
 - **faster development**
 - **easier onboarding**
 - **higher-level thinking** & allows to focus on more challenging things

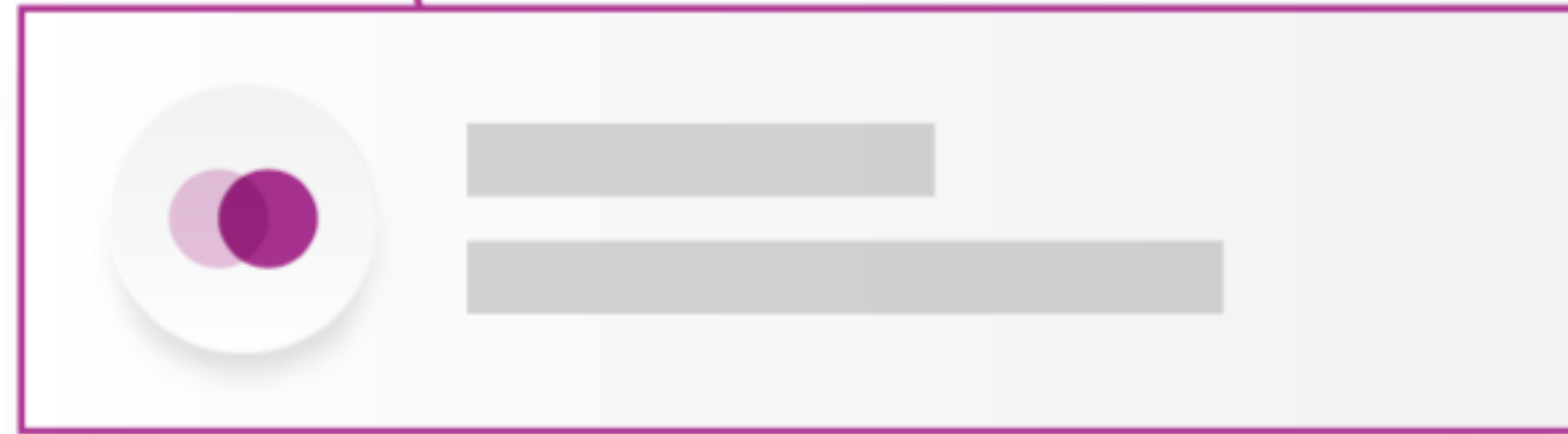
Atomic Design

- A **framework** or methodology for helping to create design systems
 - Helps us reason about UI as a **cohesive whole** and a **collection of parts** at the same time
 - Goes beyond the basics such as colours, typography, grids, etc...
- Defines a set of hierarchical layers in which our UI components will reside:
 - **Atoms** (avatar, button, spinner, ...)
 - **Molecules** (search bar, list item, ...)
 - **Organisms** (header, form, list view, ...)
 - These are combined into screens or pages



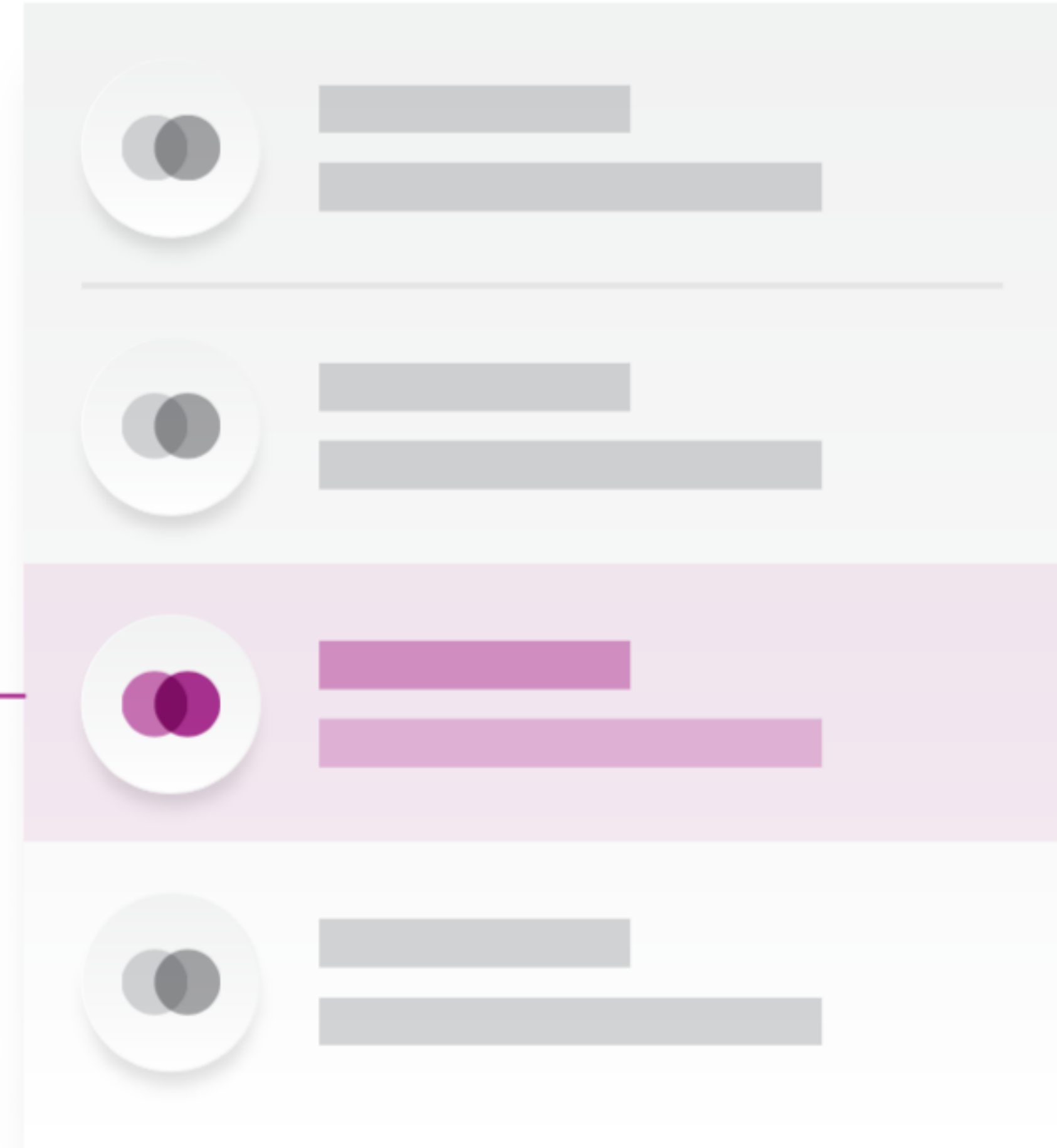


ATOMS



MOLECULES

ORGANISMS



Color

This collection of colors and tints is used across Teamle to make the visual design more consistent, provide meaning and hierarchy to the UI.

Primary colors

Lightest	#E1ECFA rgb(225, 236, 250)	Lightest
Light	#C1CEDE rgb(193, 206, 222)	Light
Teal	#64788F rgb(100, 120, 143)	Mint
Dark	#344B63 rgb(52, 75, 99)	Dark
Darkest	#2A3B4D rgb(42, 59, 77)	Darkest

Typography

Styles

A collection of typographic styles that make it possible to create hierarchy in Teamleader.

Name	Weight	Size	Line height
Heading 1	Medium	24 pt	3
Heading 2	Medium	18 pt	2
Heading 3	Medium	16 pt	2
HEADING 4	Bold	12 pt	1
Text Display	Regular	16 pt	2
Text Body	Regular	14 pt	2
Text Small	Regular	12 pt	1

Avatars

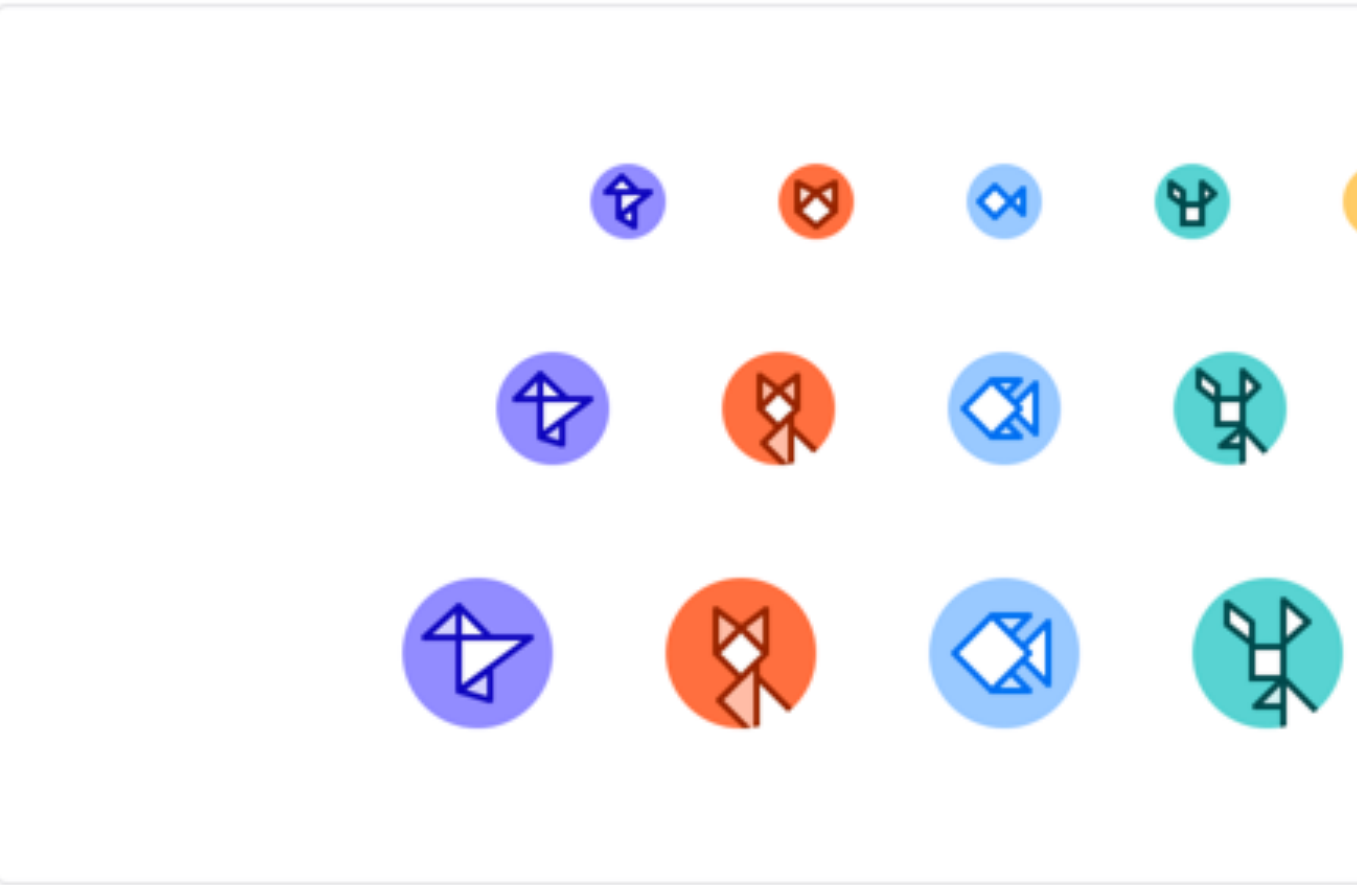
Types

Avatars are used to show a thumbnail representation of an user or company in the app.

Contacts	  
Companies	  

Sizes

Each avatar comes in three sizes: small, medium and large





Atoms/Tints/Primary/Teal

43w



Title

Body

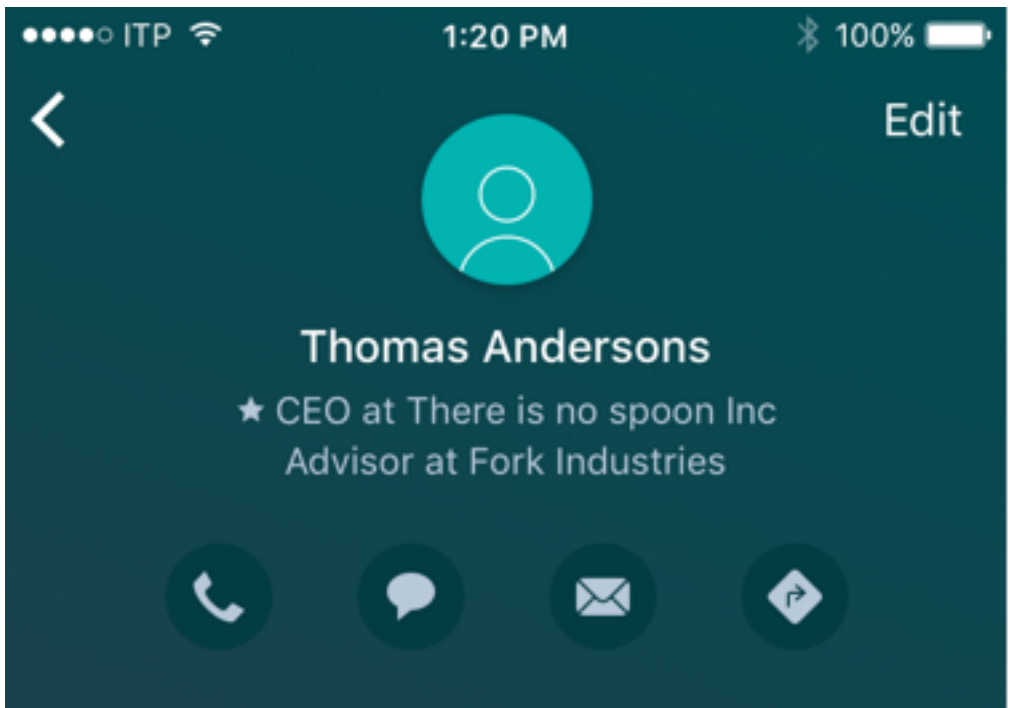
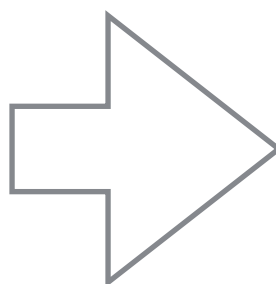
Small

Small/medium

Small/bold

Xtra small

XXtra small



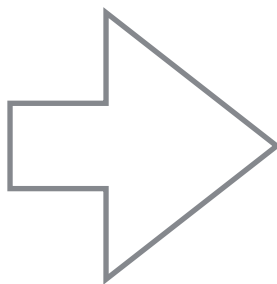
Button text ×



Label



Label



Tags

Client

Lead

Mobile

(615)661-8868



Phone

(615)661-8877



Email

thomasanderson@thereisnospoon.co



Adress

214 Overlook Circle
Suite 220
Brentwood TN 37027
United states



COMPONENTS

A great fit



Components / A great fit

React (Native)

- A **view-centric** technology
 - focused on **rendering**
 - focused on **composition** ($\neq$ inheritance)
- **Declarative** user interfaces (markup) for the win (compared to layout in code, XIBs, Storyboard, ...)
- Contain related (user interface) **logic** and **styling** for maximum **reusability**
- **Easy** to reason about:
 - props (external data)
 - state (internal data)
 - both trigger rendering
- One-way, **top-down** data flow

**PRESENTATIONAL
COMPONENTS**

**CONTAINER
COMPONENTS**

Presentational Components

- Concerned with how things look
 - Contain styling & markup
 - No dependencies on the rest of the app architecture
 - Don't specify how/where data is loaded/mutated
- Receive data & callbacks exclusively via props
 - Rarely have state (except for UI state)
 - e.g. isSelected, isActive, ...
 - Are written as functional components unless they need state, lifecycle hooks, or performance optimisations

Components / A great fit

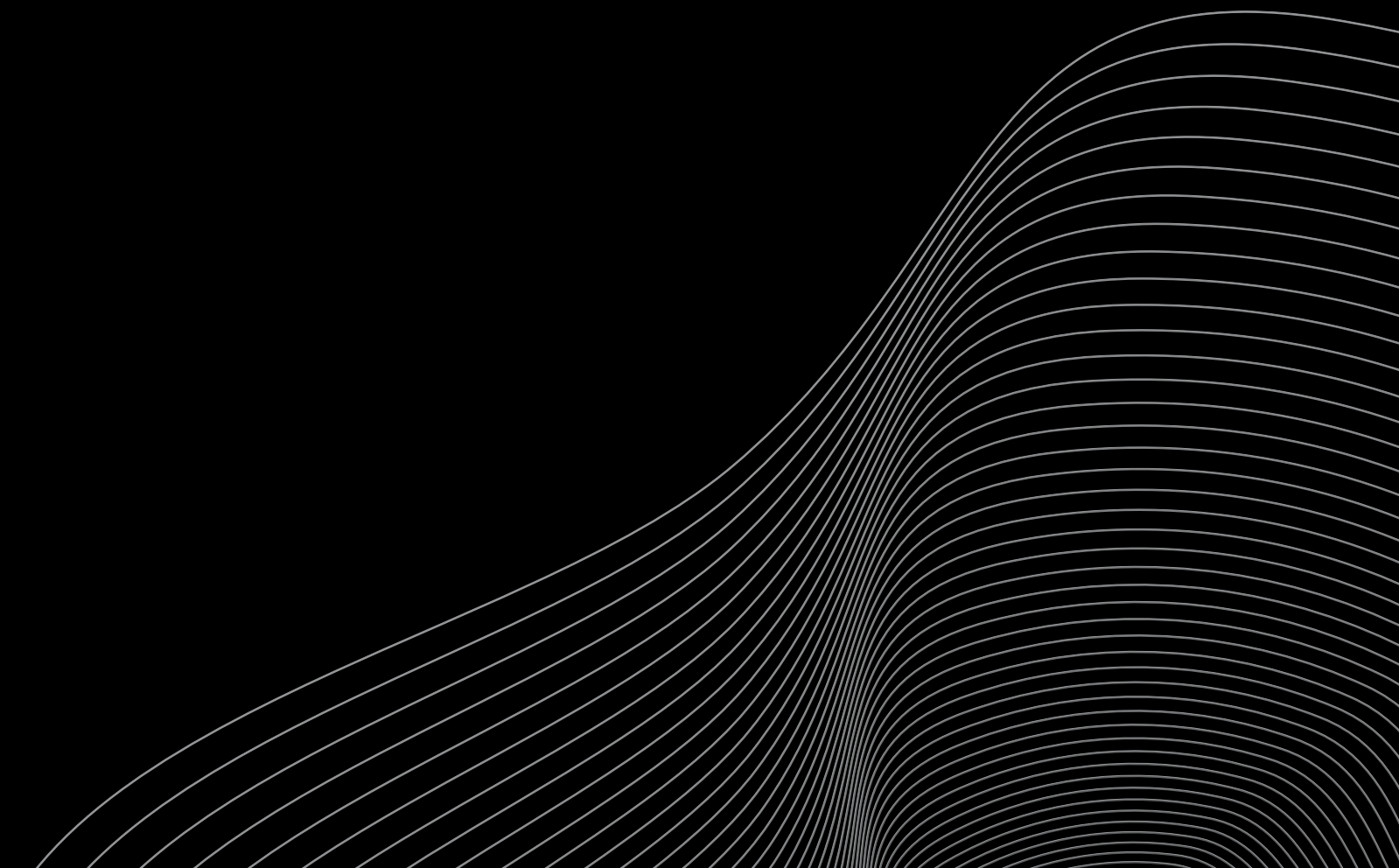
Container Components

- Concerned with how things look
 - No styling & minimal markup
 - `render()` usually only contains one presentational component
 - Provides data & behavior
- Calls actions and provides them as callbacks
 - Often stateful (tend to serve as data sources)
 - Often created by higher-order-components
 - e.g. `connect()` from Redux

Components / A great fit

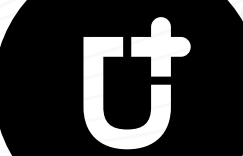
Reap the benefits

- Better separation of concerns
 - Responsibilities are clear
 - No massive components
- Better reusability (e.g. presentation is not coupled with data retrieval)
- Forces you to "extract" UI components & reason about responsibilities, data flow, ...
- All presentational components effectively form your app's UI library



TOOLING

Some interesting evolutions



Tooling / Some interesting evolutions

Living Styleguides

- The ability to document your design through code
 - The ability to **dynamically** render out design assets based on the actual implementation/code
 - "Painting with code"
 - React Sketch.app
<https://github.com/airbnb/react-sketchapp>
 - Storybook
<https://storybook.js.org/>
- 

DEMO

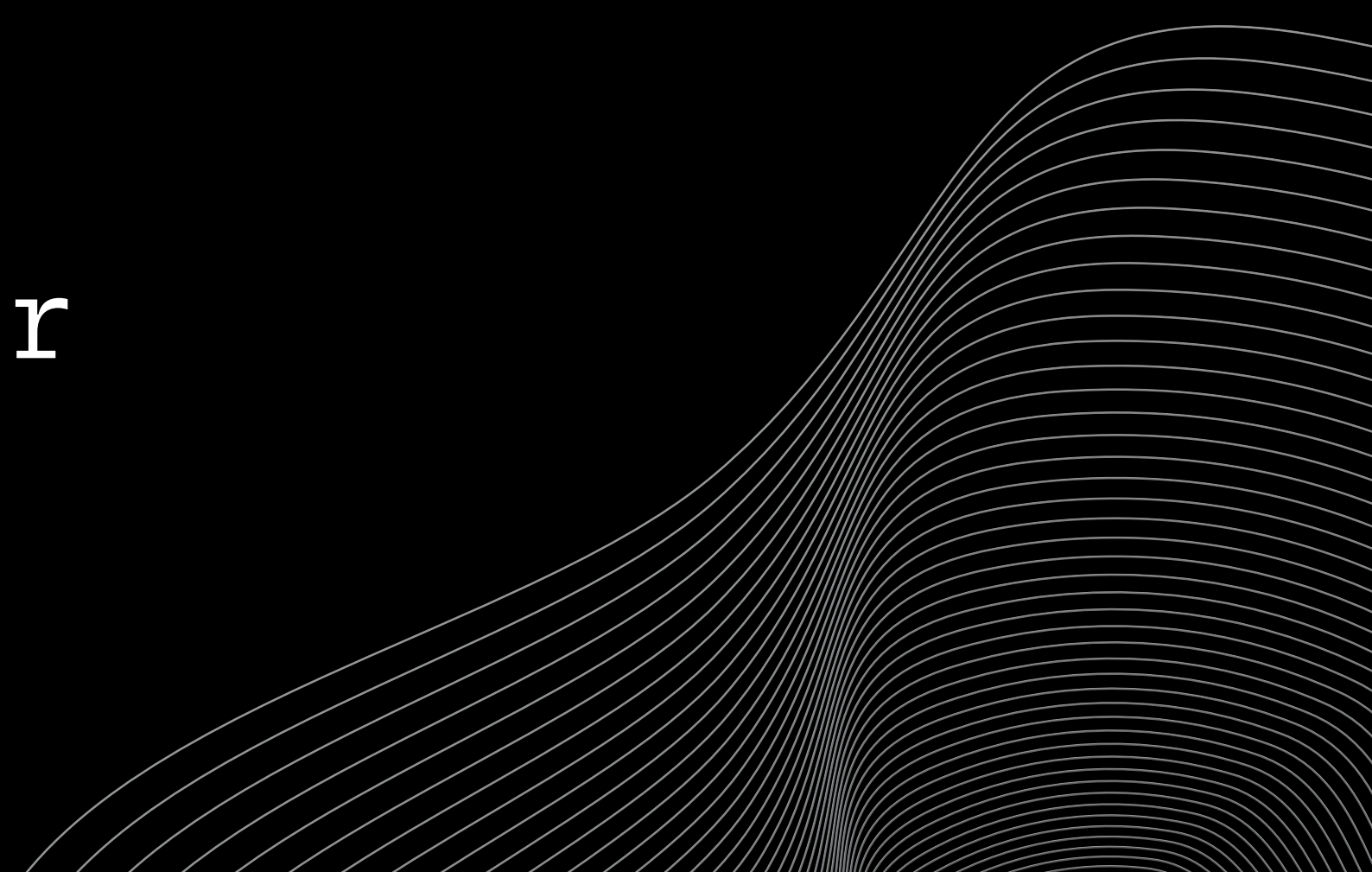
React Sketch.app

```
$ cd ~/Playground/react-sketchapp/examples
```

```
$ cd styleguide
```

```
$ cd foursquare-maps
```

```
$ npm run install && npm run render
```



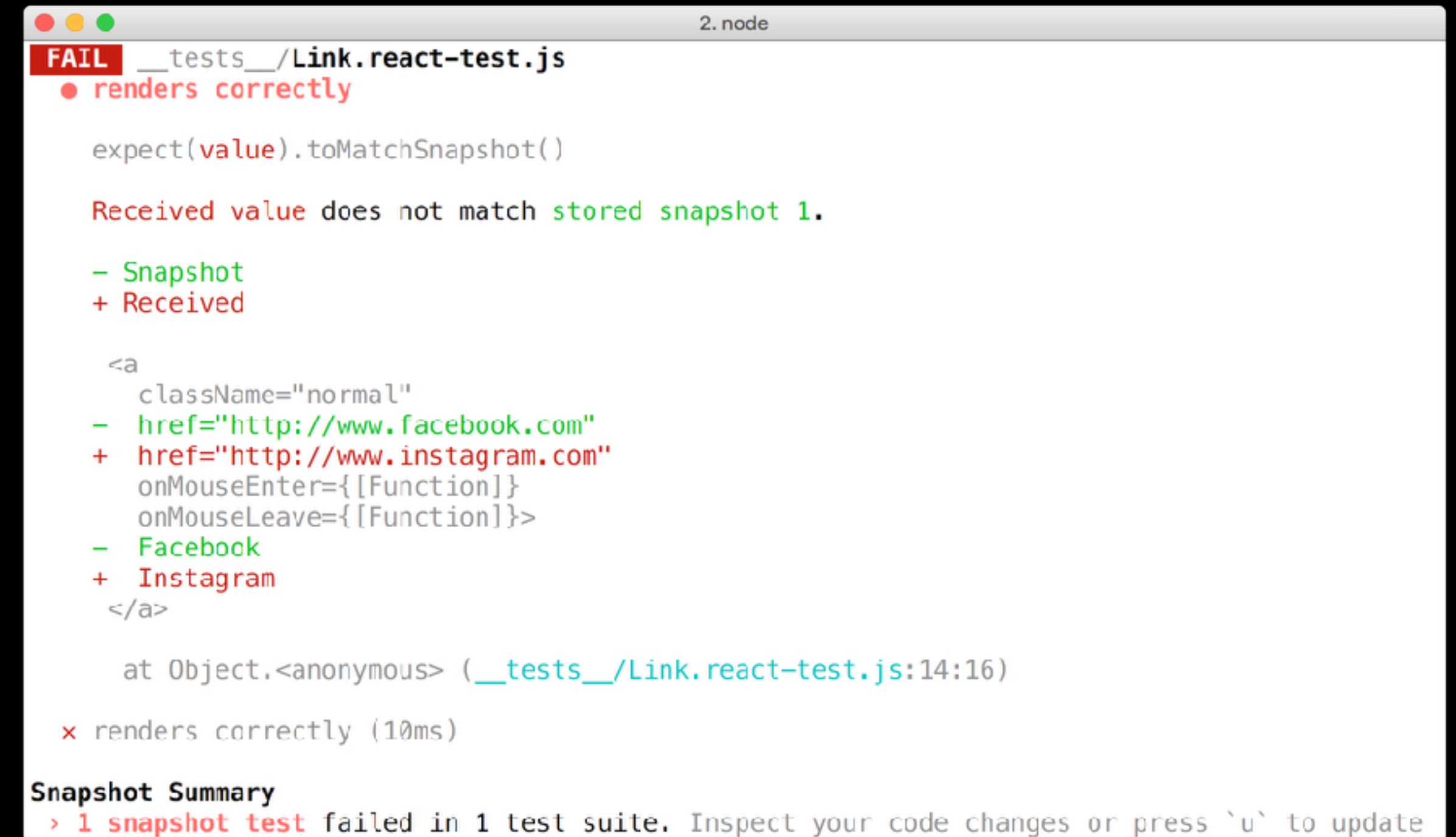
☰ Avatar / large



Tooling / Some interesting evolutions

Snapshot Testing

- The ability to take a snapshot of your components
 - $\neq$ screenshots (requires building the entire app)
- Generates serializable representations
- Used to verify that your UI does not change unexpectedly



```
2. node
FAIL __tests__/Link.react-test.js
  ● renders correctly

    expect(value).toMatchSnapshot()

    Received value does not match stored snapshot 1.

    - Snapshot
    + Received

    <a
      className="normal"
    - href="http://www.facebook.com"
    + href="http://www.instagram.com"
      onMouseEnter={ [Function] }
      onMouseLeave={ [Function] }>
    - Facebook
    + Instagram
    </a>

    at Object.<anonymous> (__tests__/Link.react-test.js:14:16)

    x renders correctly (10ms)

Snapshot Summary
  > 1 snapshot test failed in 1 test suite. Inspect your code changes or press `u` to update
```

- Part of your test setup & very easy to accomplish
- <https://facebook.github.io/jest/docs/en/snapshot-testing.html>

THANK YOU! 🙌

In The Pocket

Sassevaartstraat 46/401

9000 Gent

+32 9 234 34 25

hello@inthepocket.mobi