

Building a large, complex product from the ground up with Typescript & Atomic Design: Lessons Learned

By Nathalia Rus

Co-founder at Custodian
@yeahgirlscore
<https://nrus.io>



Custodian



WeAreDevelopers World Congress



yeahgirlscore

Lesson 1

Always be actively analyzing the products you are using in your everyday life. Search for their engineers' blogs, read CTO interviews, find their tech stack.

Don't just rely on courses, colleagues, mentors : do your own **investigations**, based on what **you** see would be a **great standard for you to reach**.



Lesson 1



Think of a **product you admire**,
and **actively find out how** the company has achieved that.



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Lesson 1



Think of a **library** you **really like using**,
and learn from its **source code**.



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Here's what happened

Startup day 0 → **panic** (what am I even doing)



Case Study - Airbnb

Mentor's advice :

“ Build on the shoulders of giants.
Go and see how Airbnb did it. ”

(yes, we are massively inspired by Airbnb).



My Goal was : REF

Readability, Efficiency, Flexibility

I was looking for workflows & architecture that...



Make code easy to read [good code is simple code]



Keep a low developer overhead



Allow for consistent but flexible changes at once



Custodian



WeAreDevelopers World Congress



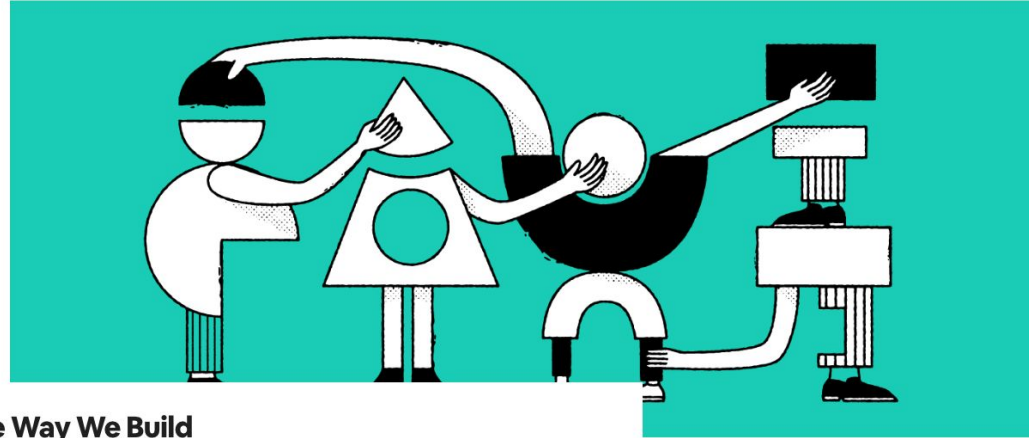
yeahgirlscodes

Lesson 1

This is how I came to...



implement Atomic Design in my startup's codebase.
Its concepts became our DNA



The Way We Build

How rethinking the Airbnb app changed the way we approach design

Perspectives — Alex Schleifer



Custodian




WeAreDevelopers World Congress



yeahgirlscore

Lesson 1

This is how I came to...

 hold on to Typescript, to which I'd initially been quite skeptical about

[Brie Bunge](#), the creator of *ts-migrate*, reported in a [talk at JSConf](#) last year that there were three primary drivers behind the switch to TypeScript: the lower number of bugs making it to production, the improved developer experience, and end-to-end type safety (from back-end to front-end). After conducting a postmortem analysis of bugs, Bunge estimated that 38% of bugs in the Airbnb codebase were preventable with TypeScript. [A research study](#) led on a selection of GitHub repositories reported that 15% of bugs may be preventable by using TypeScript.



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Lesson 1 - Examples

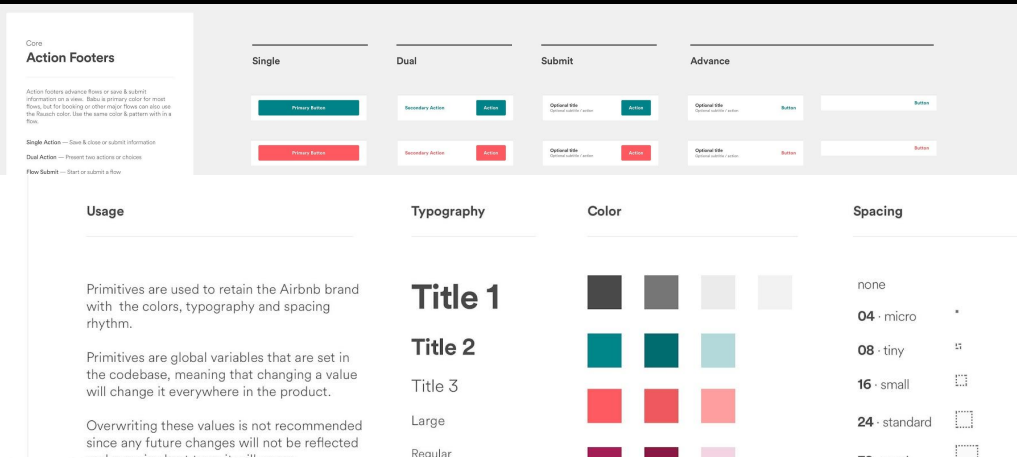
Building on
shoulders of
giants

Google - Material Design
Apple
Atlassian
IBM
Airbnb
Typeform
Deezer
Spotify

It was time to add some bumps to our flat org. But how do you add structure without harming the collaborative culture you've worked so hard to create?

The Spotify experiment

Over the last 20 years, the software industry has tried all kinds of ways to build organizations. In the beginning, teams were grouped by function—mobile engineers in one team, back-end engineers in another, and so on. Makes sense, right?



Atomic Design to the rescue

When addressing these issues, we first had to change the way we understand application design. It was important to stop seeing the apps as sets of *pages*, but instead as a *design system*. As a matter of fact, this idea is not new. Brad Frost developed a methodology to create and maintain robust design systems that he called Atomic Design (really worth a read). His idea was to break down pages into design components in an **organized hierarchy**.



Custodian



WeAreDevelopers World Congress



yeahgirlscore

Lessons learned...

Lesson 2

Invest your time in learning about Design Systems. Take it seriously.

To get started, try following 'Atomic Design' best practices in your work

You build better and faster.

It also helps you detangle business logic from UI logic



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Lesson 2

Hold on - what's Atomic Design?



Makes you think in a rigorous, scientific way.
To read: Brad Frost.



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Lesson 2

It's a way of thinking.

Don't think in terms of pages.

Think in terms of systems: spot & organise patterns.

Notice the smallest bits first.

Combine the bits into progressively larger bits until you end up with a page.

Don't start with the page and work down.

Use, assemble only those bits. Like Lego.



Custodian

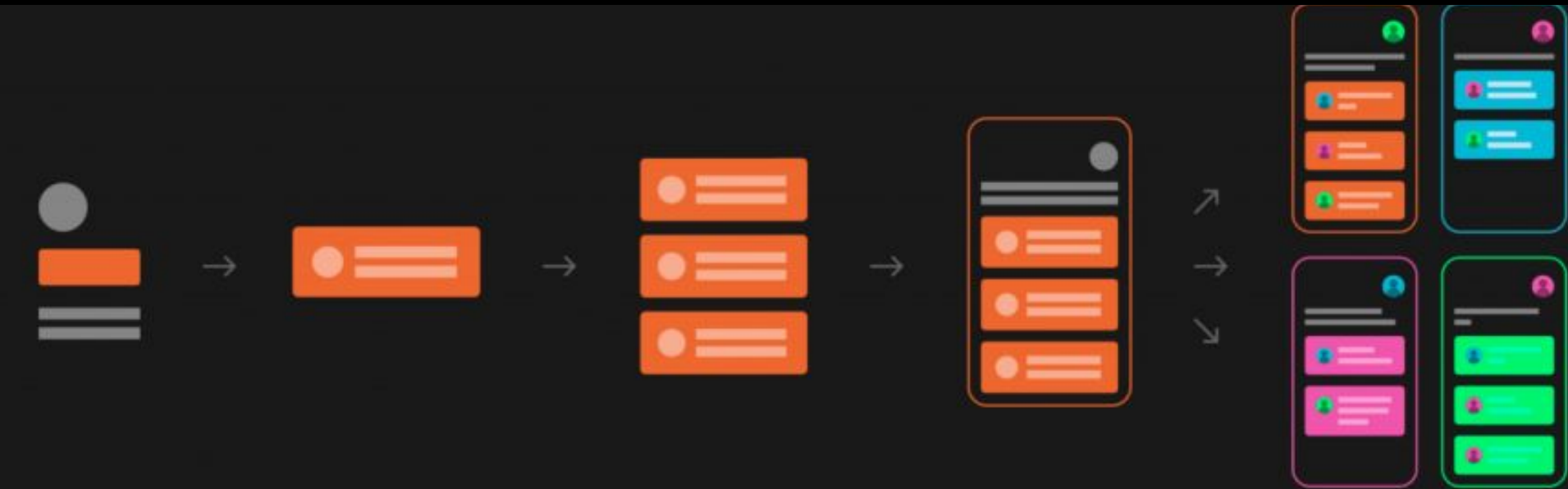


WeAreDevelopers World Congress



yeahgirlscodes

Lesson 2 - Atomic Design



src/components



Atoms → Smallest bits. Extremely reusable. 90% of styling happens here.

Molecules → Assembles a group of atoms, with minimal styling - styling only to make them fit well together (padding, flex layouts)

Organisms → What I call the “logic layer”. No styling. Its components decide of the correct “generation” of UI (molecules) and functionality based on the data it receives.

Pages → Pass down data to the relevant organism.

Templates → Encapsulate a page / transcends pages / main layouts.



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

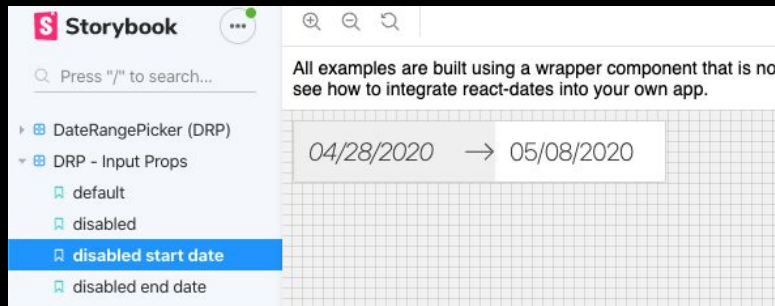
Lesson 2 - Atomic Design

Educate your eye: spot patterns

Airbnb's **storybook** can be a good eye opener & exercise.

You will see their app differently.

(*Storybook is an open source tool for building UI components*)



```
src > components > atoms > button > button.stories.tsx > ...  
1  import { storiesOf } from "@storybook/react";  
2  import * as React from "react";  
3  import Button from "../button";  
4  storiesOf("Button", module).add("with text", () => (  
5    <Button theme="primary">Hello Button</Button>  
6  ));  
7
```



Lesson 2 - Atomic Design

Advice to those who want to adapt existing codebase to Atomic Design :

⇒ implement StoryBook !

Let's run some code



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Lesson 2 - Atomic Design



Slow at start, then everything is much faster.

Maintenance is more expensive than building up a new feature.

If you're time constrained, or resource-constrained, the best decision is to first go slow, **even if it feels uncomfortably slow at first.** (keep that in mind for side-projects too.)



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Lesson 2 - Atomic Design



It's easier to build something **great**,
because a cohesive experience is more easily understood by users

— You build something great, **faster**

because it gives your team (cross department: product / design / engineers + new onboardings)
a common language to work with.
(yourself too when you come back to your code later)



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

A potential problem –

Usual difficult balance between

‘over-engineering’ too early
and it complicates things going forward,

vs

writing code that ‘works for now’
but will later cause huge mess



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Let's see an **example** where **early over-engineering** won't do you good

DRY believer - but it's better to repeat yourself sometimes

Let's run some code



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Atomic Design gives you a framework to **make a better judgement**,
and to **know where things belong** - at any point of time

You divide and conquer

You have separation of concerns.

Each component does one thing, well.



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

One thing I regret?

I should have adopted Hooks earlier

Hooks ideal for managing both simple and complex **UI**

In some cases, classes offer a rigidity and visual structure that be nice for code readability

(this is React-specific)



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

One thing I regret?

I should have prioritised End-To-End Testing, rather than Unit Testing *

(* if you have to choose)

While Typescript does not replace proper tests, it does cover a lot of bugs ground

End-to-end testing (Cypress) was key for rigorous checks on all device displays before deploy to prod

[interestingly, I found the contrary for back-end where Unit Testing = invaluable]



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Things I do not regret.

Keeping Storybook up to date

Do it as soon as you create a new UI element, I promise it doesn't take time.



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Things I do not regret.

I'm glad I kept Redux / a state management library

With new things like Context API, some say state management libraries are redundant

Avoids lock-in, business logic is portable

Very nice separation of concerns, better code organisation

Redux devTools make debugging infinitely easier

Powerful capabilities for managing side effects, persistence, and data serialization

Read :

Why React Context is Not a "State Management" Tool (and Why It Doesn't Replace Redux)
by Redux maintainer Mark "acemarle" Erikson



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

One thing I do not regret.

Whatever choice, make effort to keep business logic **separate** from UI logic

(Examples of what I wouldn't do :

Specific update functions in forms, and having separate forms for each things

Querying data anywhere else outside of the page component,

Hooks spaghetti and untracked *useEffect* side effects

etc)



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Lessons learned...

Lesson 3

If you're using Javascript,
I strongly believe Typescript is **always** a good idea.



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Why are some developers skeptical ?

(I was too !)

- visually, it can makes simple code **looks complicated**
- **libraries'** usage can throw annoying and unnecessary type **errors**
- It can feel like an **overkill**
- TS codebases can **appear cryptic**
- **Initially**, it can make you **feel productivity is slowed**
- you suddenly feel like you have to think in TS, and it's weird

Is this you?



Why TS

1. Less bugs - objectively

JavaScript is a dynamically typed language, which means that types are checked, and data type errors are only detected at runtime.

This can be very dangerous and can create errors during production



Why TS

2. Scalable way of thinking

TypeScript is called “JavaScript that scales”.

Types have a proven ability to enhance code quality and understandability.

Large teams (Google, Microsoft, Facebook) have continually arrived at this conclusion, and every year more companies decide upgrading to TS



Why TS

3. Easier **refactoring**

4. Types are one of the best forms of **documentation** you can have. The function signature is a theorem and the function body is the proof.

5. **Intellisense** helps writing faster and with avoid typo / navigation between files

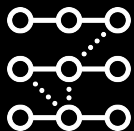
And more...



Typescript and Atomic Design: A match Made in Heaven



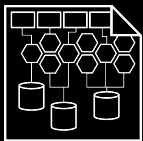
Lesson 3 - Typescript



Atomic Design ensures developers are all on the same page,
to **atomic levels**.



Typescript ensures developers are all on the same page,
to an even **deeper** level -
everything needs to be **defined and expected**. + 'Documents'



You can **define** what comes in and out of your components and
be strict, enforcing rules -making your atoms, molecules and
organisms even more robust.



Lesson 3 - Typescript

```
    </div>
    (JSX attribute) theme: "primary" | "secondary" | "naked-icon" | "naked-light"
    <ButtonWrapper | "naked-bold" | "naked"
    <Button theme="primary" onClick={closeModal}>
    | back
    </Button>
  </ButtonWrapper>
```

```
<Button
  type="submit"
  theme="helloworld"
  disabled={
    isSubmitting ||
    !!((errors.email && touched.email) ||
    !!((errors.password && touched.password)
  }
>
  enter
</Button>
```

```
(JSX attribute) theme: "primary" | "secondary" | "naked-icon" | "naked-light"
| "naked-bold" | "naked"

No overload matches this call.
  Overload 1 of 2, '(props: Readonly<IButtonProps>): Button', gave the
  following error.
    Type '"helloworld"' is not assignable to type '"primary" | "secondary" |
    "naked-icon" | "naked-light" | "naked-bold" | "naked"'.
  Overload 2 of 2, '(props: IButtonProps, context?: any): Button', gave the
  following error.
    Type '"helloworld"' is not assignable to type '"primary" | "secondary" |
    "naked-icon" | "naked-light" | "naked-bold" | "naked"'. ts(2769)
button.tsx(11, 3): The expected type comes from property 'theme' which is
declared here on type 'IntrinsicAttributes & IntrinsicClassAttributes<Button>'
theme="helloworld"
```

Error shows up on the spot..

And explain why + what is expected instead.



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Lesson 3 - Typescript

```
// FORM TYPES
```

```
export const OFormType = {  
  checkbox: 'CHECKBOX',  
  radio: 'RADIO',  
  rating: 'RATING',  
} as const
```

```
export type FormType = keyof OFormType[keyof OFormType]
```

```
// RATING TYPES
```

```
export const ORatingType = {  
  heart: 'HEART',  
  sentiment: 'SENTIMENT',  
  grade: 'GRADE',  
} as const
```

```
export type RatingType = keyof ORatingType[keyof ORatingType]
```

```
export type ValidationOptions = {  
  isRequired?: boolean  
  maxChar?: number  
  minChar?: number  
}
```

Union Types, or objects `as const` rather than enums



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Lesson 3 - Typescript

```
renderFormBody = (item: Question) => {  
  switch (item.formType) {  
    case 0FormType.radio:  
      return <FormElements.RadioGroup items={item.options} />  
  
    case 0FormType.checkbox:  
      return <FormElements.CheckboxGroup items={item.options} />  
  
    case 0FormType.rating:  
      return (  
        <FormElements.RatingGroup  
          ratingTheme={item.ratingTheme}  
          items={item.options} />  
      )  
    }  
}
```

```
export type Question = {  
  title: string  
  answers: string[]  
  options: FormItem[]  
  formType: FormType  
  hasValidation?: ValidationOptions  
}
```



Lesson 3 - Typescript

Explicitly define...

- the way components **relate to each other**
- and **what is needed** for everything to work as a unified system

⇒ you should be able to **visualise** the components and **hierarchies** as you read your **types**.



Another win 🙄🙄

This structure (atomic design + typescript)
makes you **avoid** most common front-end **anti-patterns**



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Let's have a look at some code



Custodian



WeAreDevelopers World Congress



yeahgirlscodes

Thank you

PS - We are Hiring !

Please contact me, or head to
custodian.club/careers

Nathalia Rus

View Roles



**Custodian
Careers**

<https://www.custodian.club/careers>



yeahgirlscodes



WeAreDevelopers World Congress