



Consider the context,
break down into components,
write scalable, adaptable and maintainable
code



OR WHY DOES CONTEXT MATTER
AND HOW SPLIT COMMON DESIGN AND FUNCTIONALITIES INTO MODULAR COMPONENTS?

Table of Contents

1. Prologue	01
2. Atomic Design	07
3. Content First – Design Last	23
4. Store & Share	29
5. Object-Oriented <code>	34
6. Epilogue	44

The facts

In front end web development and design the word "component" has been popping up more and more, from React.js components to new CSS class naming conventions to web components to style guides.

While the components which are now gaining in popularity are highly useful, the idea of breaking down systems into smaller, more modular parts (which is nothing new but is a fundamentally strong design pattern) can bring us huge benefits and allow for more unity between design and development. These concepts can stretch far and wide and are well worth exploring.

“ We’re not designing pages, we’re designing systems of components.

—



Stephen Hay
@stephenhay

So... what is a component?



component

A part or element of a larger whole

1. less code

The code doesn't grow with the project but only with new components.

2. better,

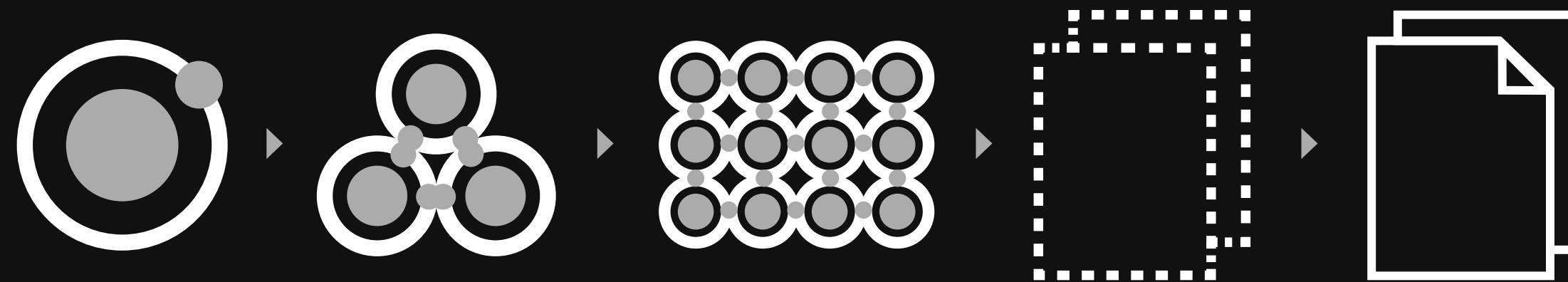
faster. Create new elements with HTML only. Just reuse what has been already lined up.

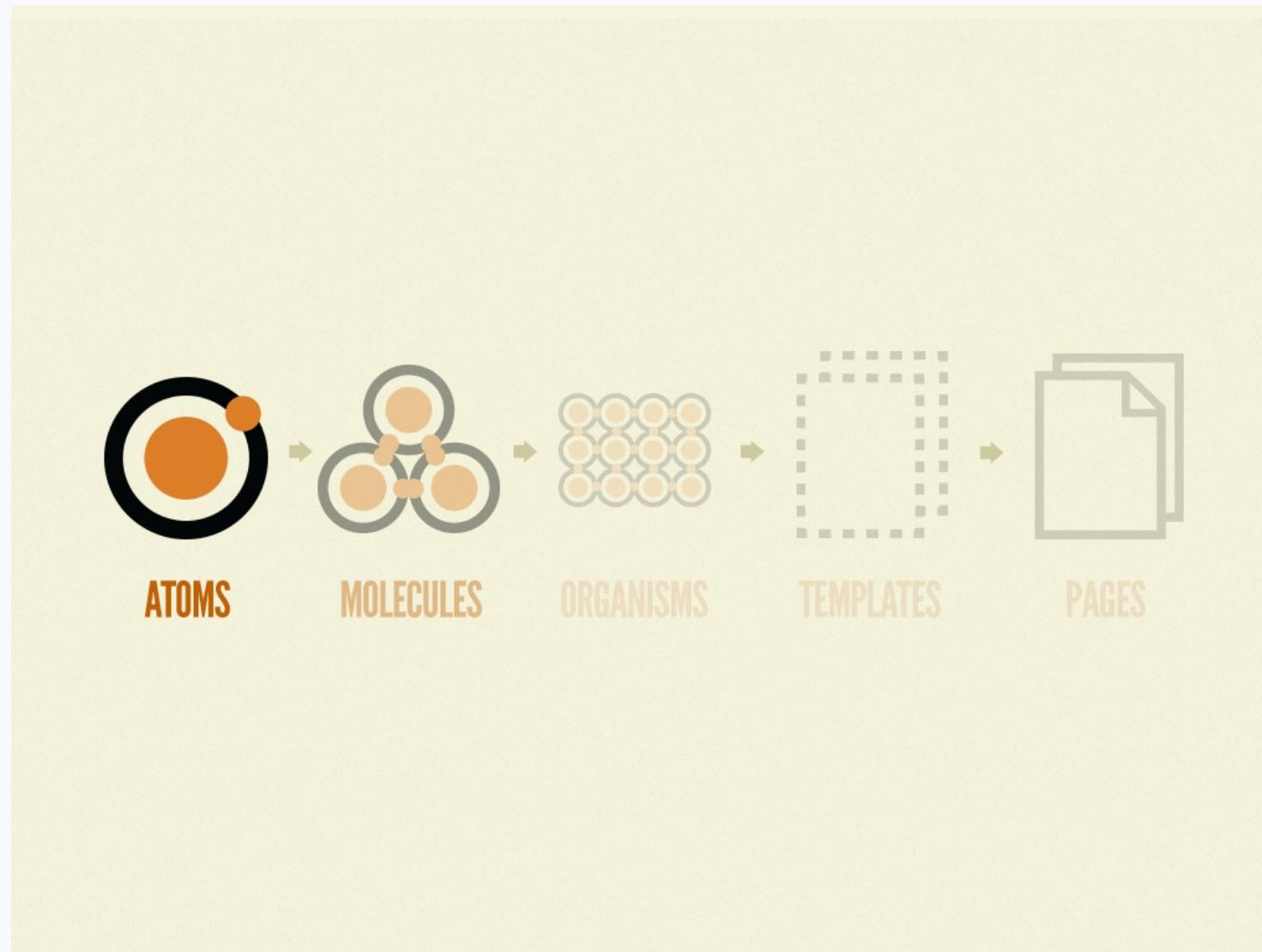
3. e easier

Each component is independent from its context. They can move anywhere because they aren't defined by their location.

METHODOLOGY

Atomic Design





Atoms
are UI elements
that can't be broken
down any further
and serve as the
elemental building
blocks of an
interface.

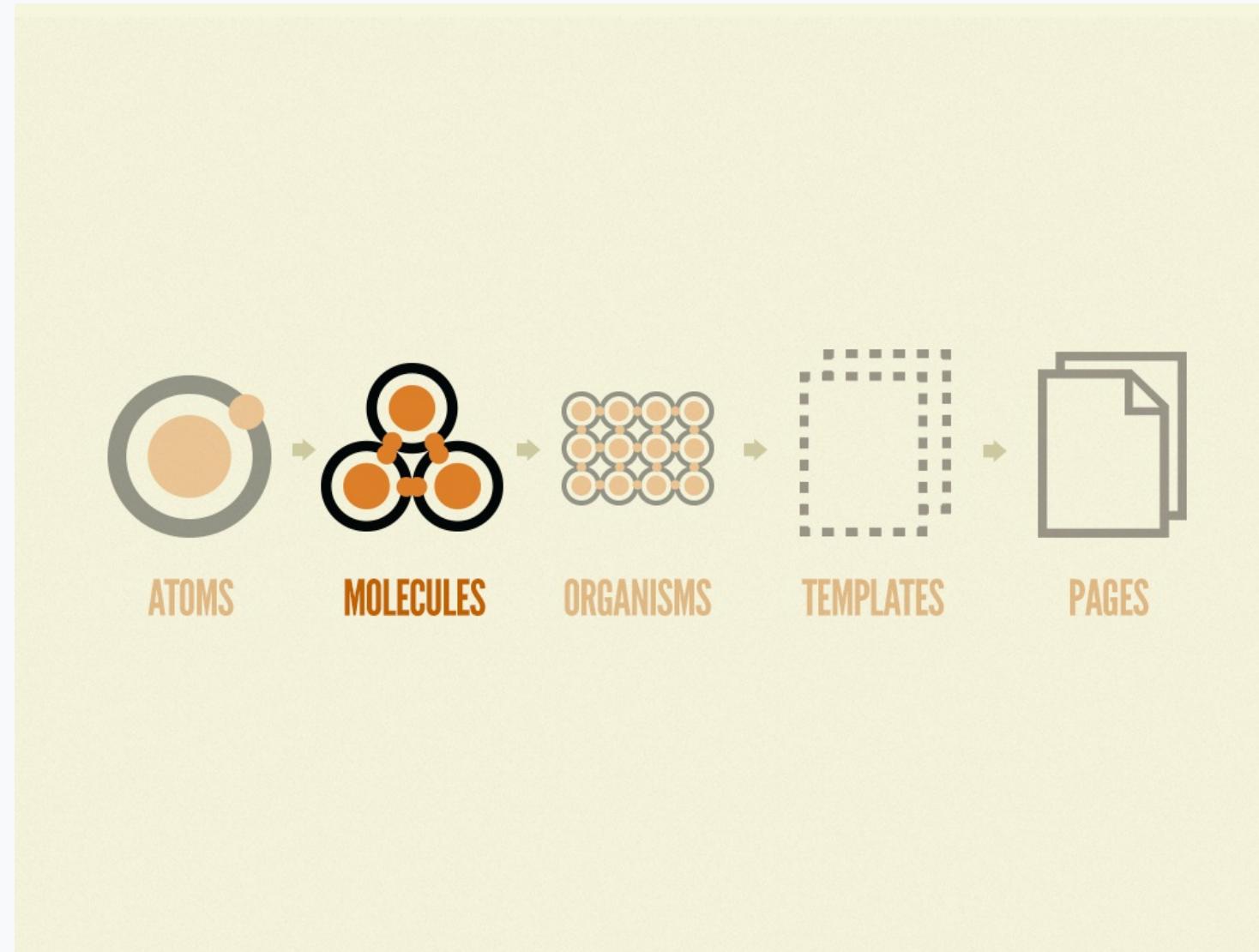
<label> <input> <button>

Search the site

Enter Keyword

Search

Atoms



Molecules
are collections of
atoms that form
relatively simple UI
components.

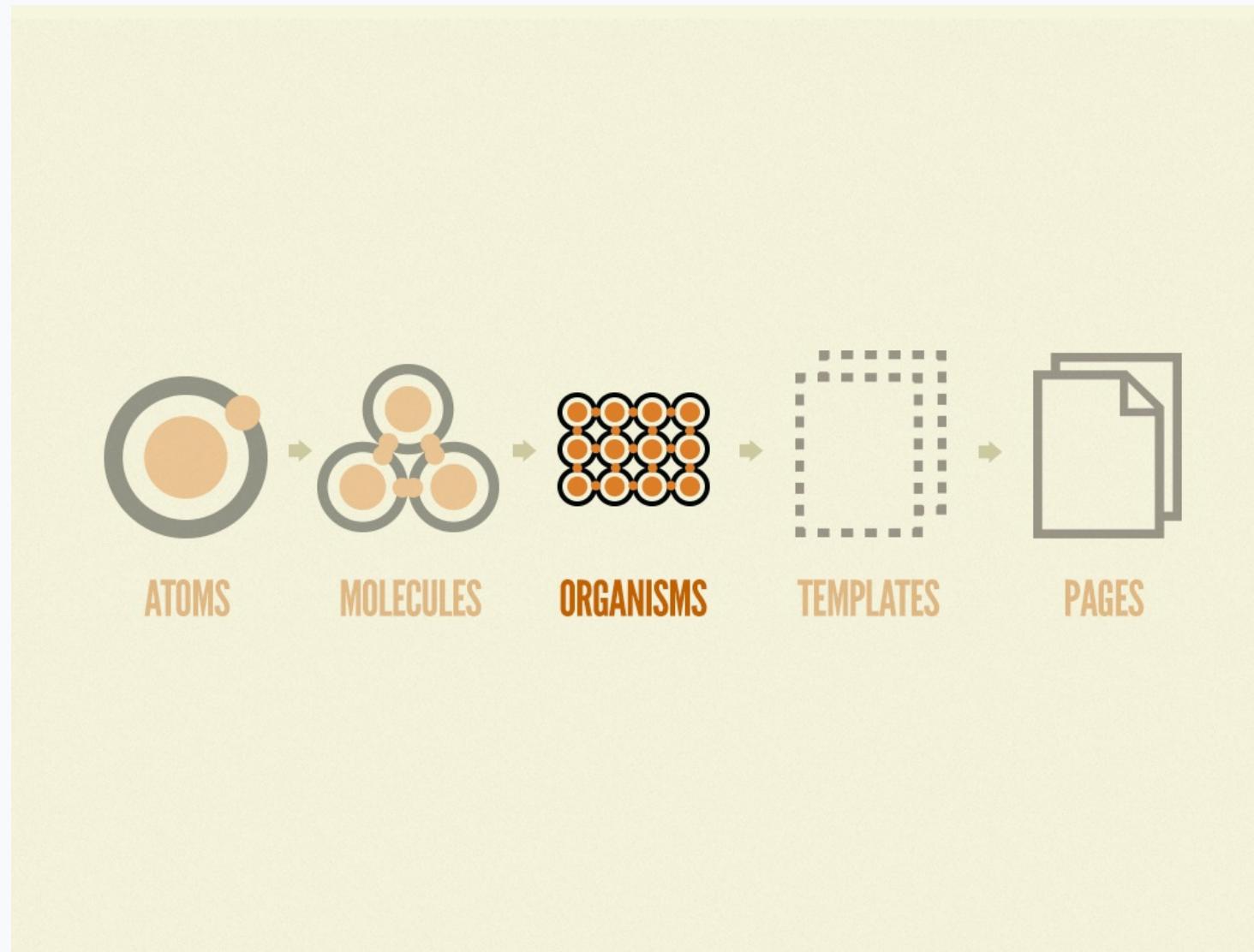
<form>

Search the site

Enter Keyword

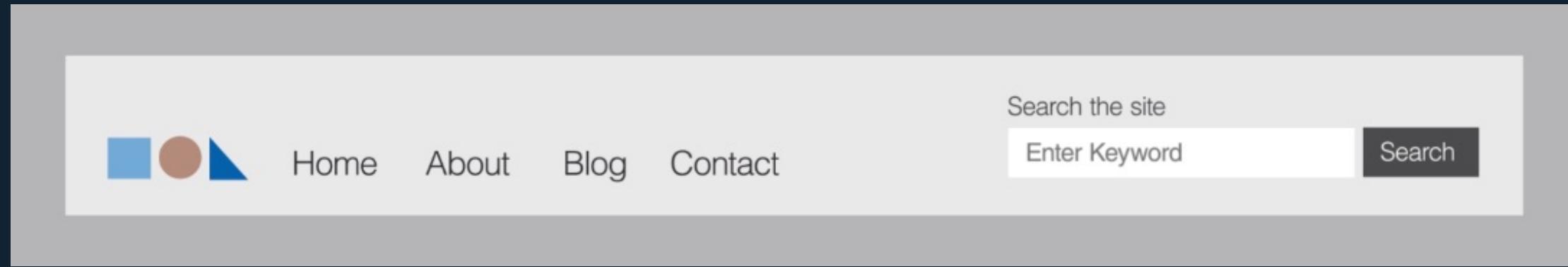
Search

Molecule

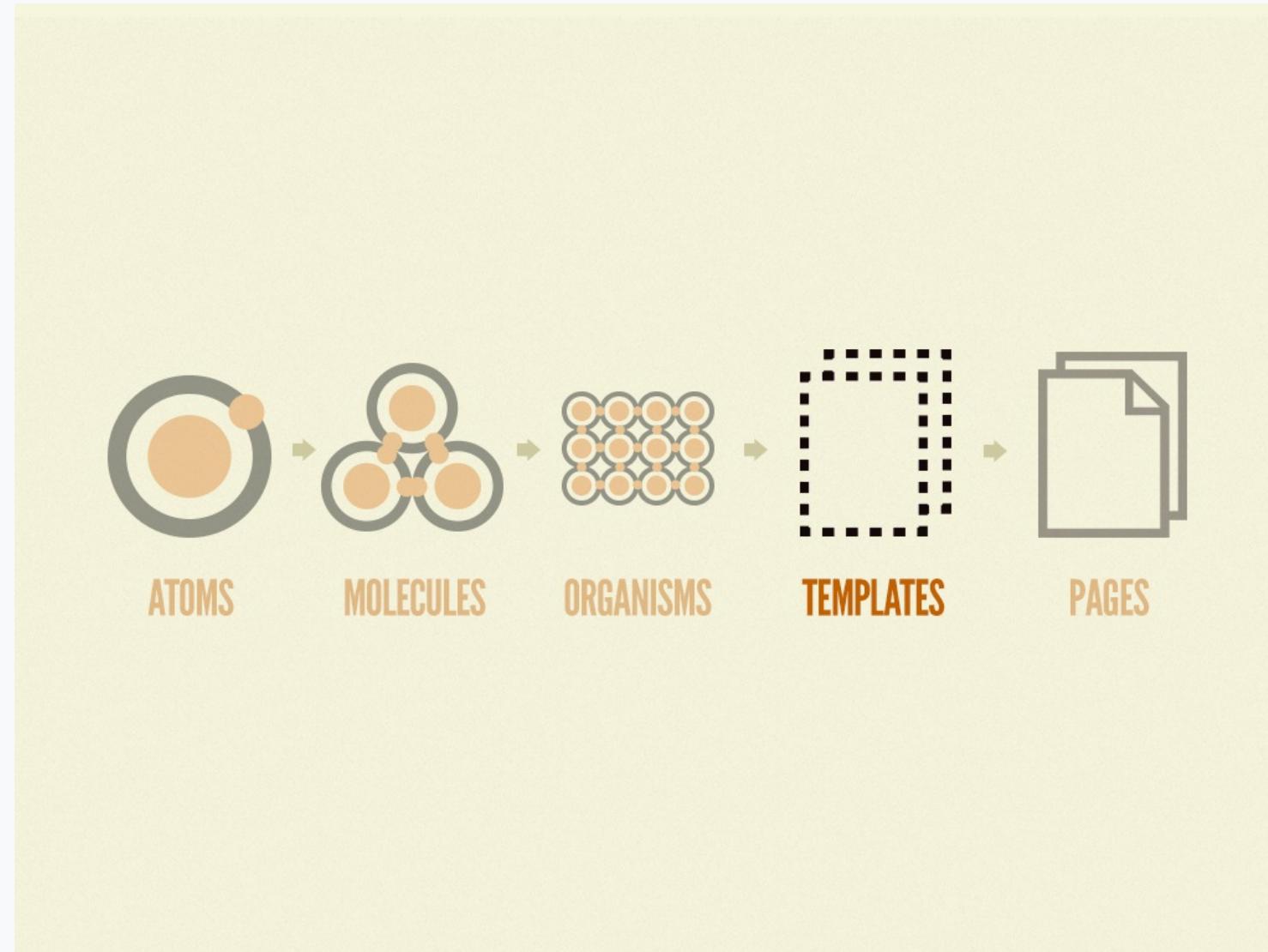


Organisms
are relatively
complex
components that
form discrete
sections of an
interface.

<header>

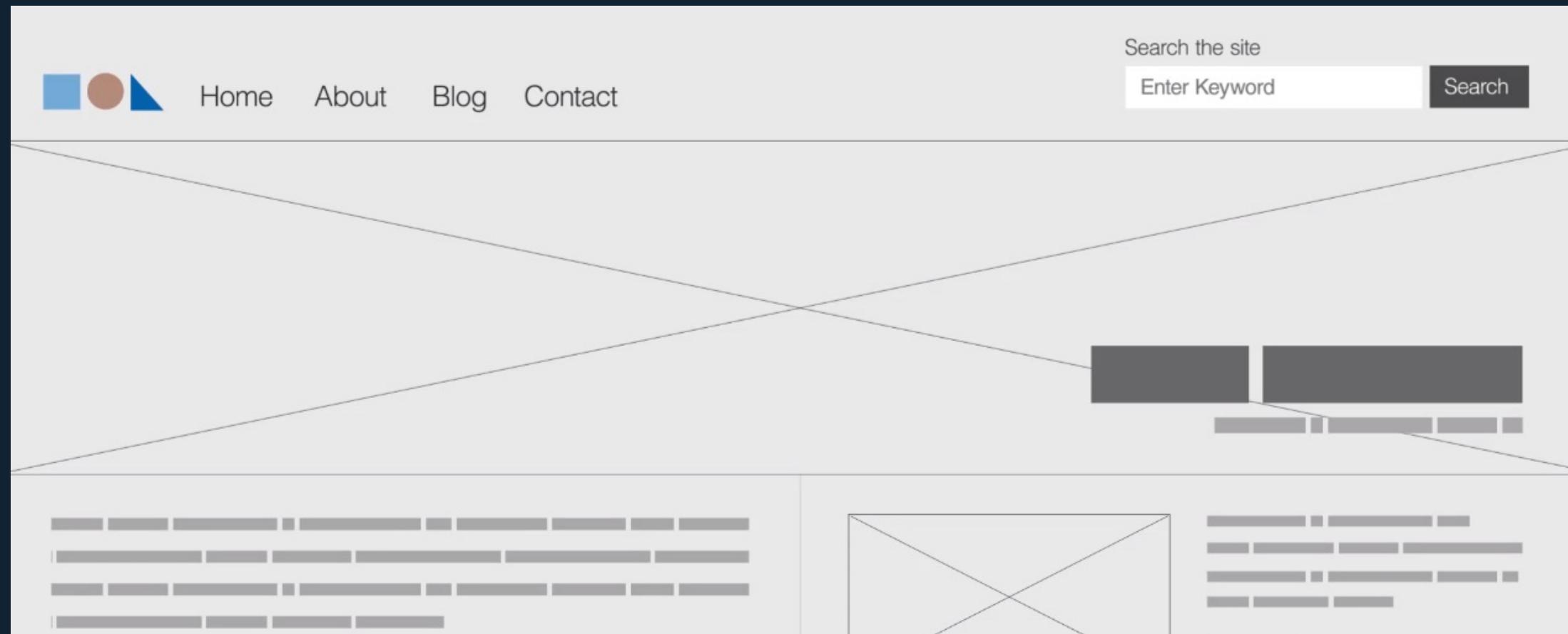


Organism

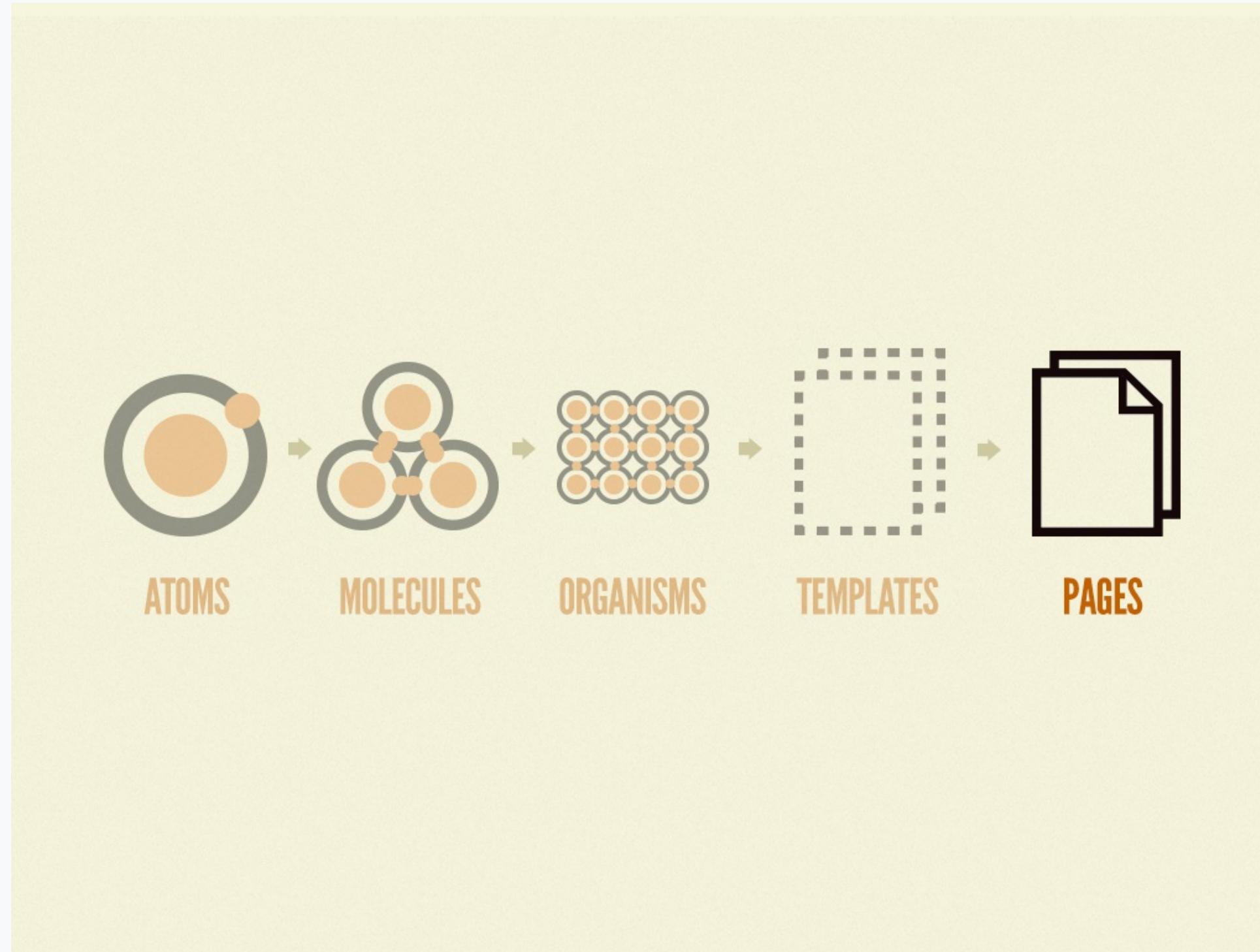


Templates

place components within a layout and demonstrate the design's underlying content structure.



Template



Pages

apply real content to templates and articulate variations to demonstrate the final UI and test the resilience of the design system.

The page stage is most concrete stage of atomic design, and it's important for some rather obvious reasons. After all, this is what users will see when they visit your experience. This is where you see all those components coming together to form a beautiful and functional user interface. Exciting!

Search the site

Enter Keyword

Search

Your Content

Swap in real representative content.

In addition to demonstrating the final interface as your users will see it, pages are essential for testing the effectiveness of the underlying design system.

Page

The part and the whole

One of the biggest advantages atomic design provides is the ability to quickly shift between abstract and concrete.

We can simultaneously see our interfaces broken down to their atomic elements and also see how those elements combine together to form our final experiences.

“ The painter, when at a distance from the easel, can assess and analyze the whole of the work from this vantage. He scrutinizes and listens, chooses the next stroke to make, then approaches the canvas to do it. Then, he steps back again to see what he’s done in relation to the whole. It is a dance of switching contexts, a pitter-patter pacing across the studio floor that produces a tight feedback loop between mark-making and mark-assessing.

—



Frank Chimero

@frank_chimero

Atomic design is not a linear process

Think of the stages of atomic design as a mental model that allows us to think of our user interfaces as both a cohesive whole and a collection of parts at the same time.



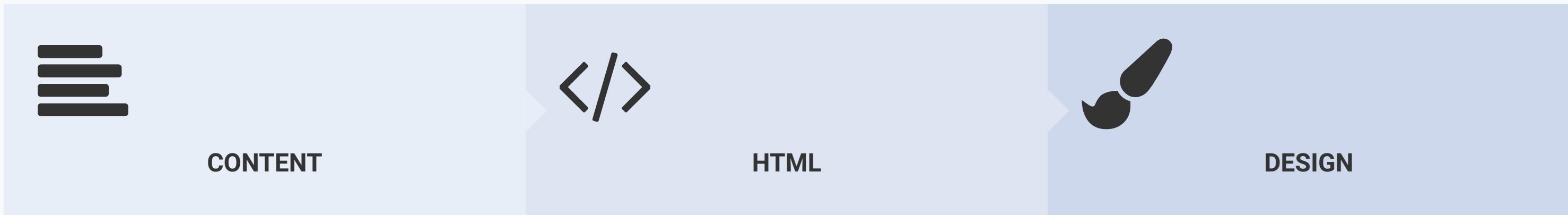
CUTTING DESIGN FILES

Content First – Design Last

With a content-first approach, we move from designing without content to designing based on content – a very important distinction.

Remember the definition of semantic HTML:

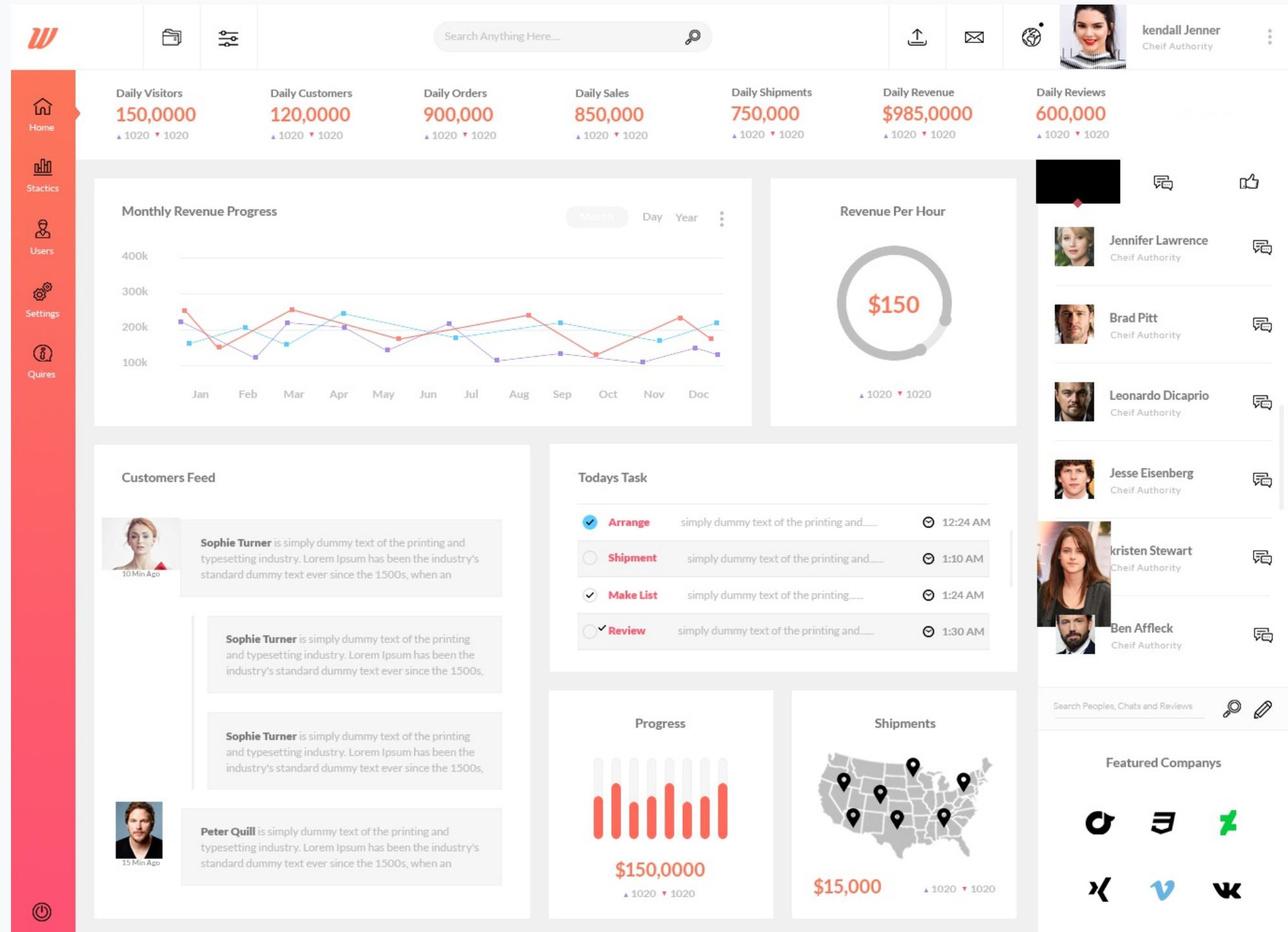
GIVING MEANING TO CONTENT



Really think about what's important

Before we start writing HTML, we need to determine what content to present on the page and how to prioritize it.

<nav> <main> <aside> <article> <section>



Wofsus Dashboard by Sajal Jahan (2017)

e
m
e
n
t
?

CLICK HERE

#RTFD

The HTML `<button>` element creates an interactive control.

The HTML `<a>` element (or anchor element) creates a hyperlink to other web pages, files, locations within the same page, email addresses, or any other URL.

u
t
d
e
s
i
g
n
?

CLICK HERE

```
text-shadow: none;  
box-shadow: none;  
background-color:  
transparent;  
color: black;  
  
// common styles  
margin: 0;  
padding: 0 20px;  
display: inline-flex;  
justify-content:  
center;
```

A black and white photograph showing a group of people from behind as they walk up a series of wide, light-colored stone or concrete steps. The people are dressed in casual attire like jeans, sweatshirts, and jackets. Some are carrying bags. The scene is outdoors, possibly at a historical site or a modern public space.

—
STYLE GUIDE

Store & Share

A front-end style guide is both a **deliverable** created by the UX team (in concert with the engineering team, typically) and a **tool** used by the entire team for maintaining consistent, nimble product design in a modular format.

What is a style guide?

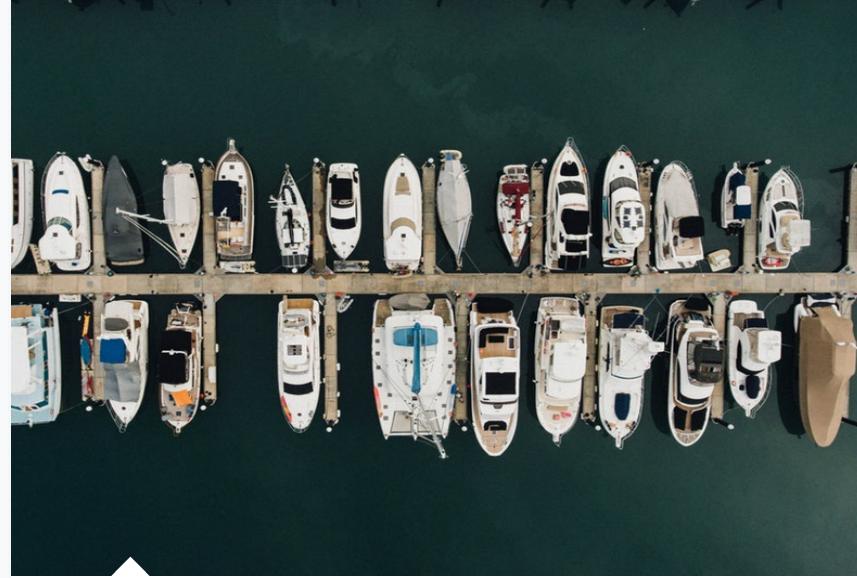
“ A style guide is a living document of code, which details all the various elements and coded modules of your site or application. Beyond its use in consolidating the front-end code, it also documents the visual language, such as header styles and color palettes, used to create the site. This way, it’s a one-stop place for the entire team—from product owners and producers to designers and developers—to reference when discussing site changes and iterations.

—  *Susan Robertson*
@susanjrobertson



BUILD FAST

Creating your guide will take some time up front, but this pays off with **faster build times for new sections and pages.**



BUILD STRONG

A guide allows us to **standardize the CSS, keeping it small and quick to load.**



BUILD ACCURATE

Design consistency is easier to maintain, as the designer can look in one place to reference the site's components and ensure a cohesive look and feel throughout.

A REAL LIFE EXAMPLE

 Azureva style guide

◎ Central reference point

Benefits

ACCURATE
OVERVIEW
OF THE UI

DEVELOP
MENT
PLAYGROU
ND

KICKSTAR
T FOR NEW
PAGES

SIMPLIFIE
D TESTING

COMPONE
NT
CULTURE

COMMUNI
CATION
BENEFITS

GET STARTED



Download/fork style guide boilerplate



APPROACH

Object-Oriented <code>

Object-oriented definition

WHICH ONE?

N O B O D Y A G R E E S O N W H A T O O I S

⌚ 3 principles



ENCAPSULATION

enforces Modularity

Encapsulation is the idea that the attributes of an entity are enclosed in that entity.



INHERITANCE

passes "knowledge" down

Inheritance is the idea that an entity can inherit attributes from another entity.



POLYMORPHISM

takes any shape

Polymorphism means “having many forms”: when any thing perform more than type of task.

What is an object?



object

In essence, an object is a discrete entity that has only the necessary dependencies on other objects to perform its tasks.

says, "Can you get my clothes laundered, please." I happen to know where the best laundry place in San Francisco is. And I speak English, and I have dollars in my pockets. So I go out and hail a taxicab and tell the driver to take me to this place in San Francisco. I go get your clothes laundered, I jump back in the cab, I get back here. I give you your clean clothes and say, "Here are your clean clothes."

You have no idea how I did that. You have no knowledge of the laundry place. Maybe you speak French, and you can't even hail a taxi. You can't pay for one, you don't have dollars in your pocket. Yet, I knew how to do all of that. And you didn't have to know any of it. All that complexity was hidden inside of me, and we were able to interact at a very high level of abstraction. That's what objects are. They encapsulate complexity, and the interfaces to that complexity are high level.

— Steve Jobs

W H A T A B O U T F R O N T - E N D ?

HTML

To build websites, you should know about HTML.

Place to learn?

MDN

The main entry point for HTML documentation, including detailed element and attribute references – if you want to know what attributes an element has or what values an attribute has, for example, this is a great place to start.

Why learn?

Semantics tags have many benefits beyond pure efficiency, device compatibility and SEO. They help us build better site structures, make all programmatic interaction and manipulation much easier, and more importantly, they can seriously improve websites' accessibility.

No matter what methodology you choose to use in your projects, you will benefit from the advantages of more structured CSS and UI. Some styles are less strict and more flexible, while others are easier to understand and adapt in a team.

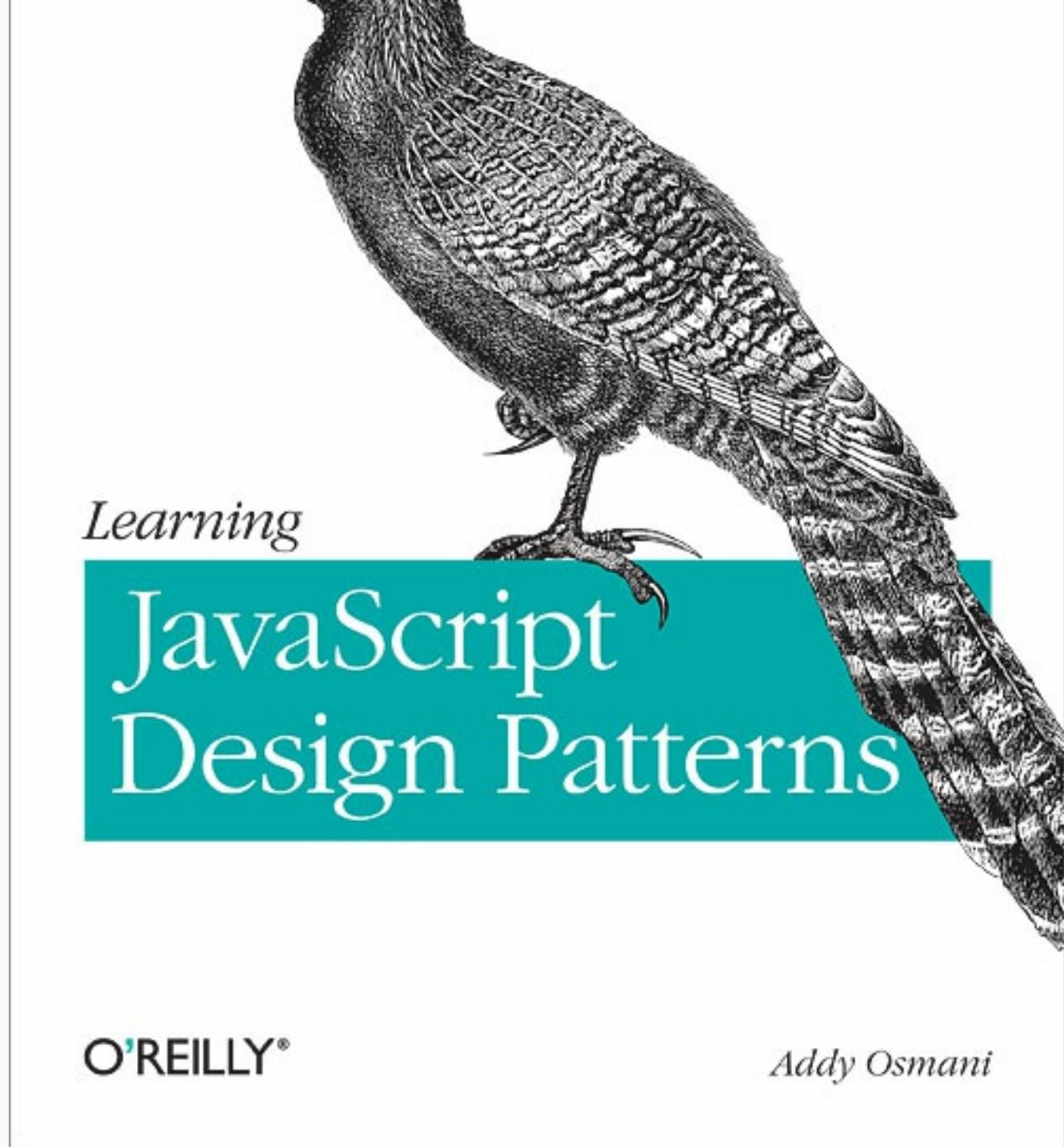
Front-end frameworks

BOOTSTRAP

FOUNDATION

SEMANTIC UI

For sites that need to be heavily customized or introduce a new kind of interaction that isn't standard, it may take longer to customize a framework than to build from scratch.



JavaScript Module Pattern

[READ THE EBOOK](#)

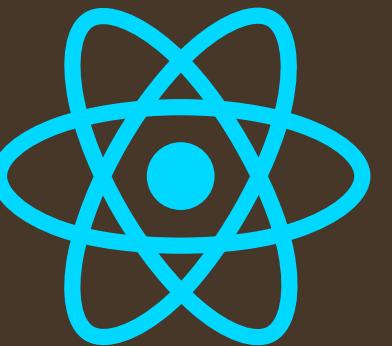
For further reading: [You Don't Know JS](#), a series of books diving deep into the core mechanisms of the JavaScript language.

JavaScript libraries



Vue.js

"The cool kid on the block"



React.js

"Leading the satisfaction statistics"



Angular

"Wisdom of elders"

For further reading: [Angular vs. React vs. Vue: A 2017 comparison](#)

website there are common themes of what a component is.

Independent

Components should be able to be used on their own and rely on only a limited set of dependencies. They should be built so they don't 'leak' or cross-over into other components.

Clearly defined

Useful but limited scope - E.g a 'button' component + a 'search box' component is better than a 'navbar' component, but a search button and search input shouldn't be split up into two separate components if they'll never be used independently.

Encapsulated

Components should 'wrap up' their functionality within themselves and provide set ways of implementation. E.g a button component could expose "size" and "colour" options.

Reusable

Components are often built with reusability in mind, although they may initially only be implemented once.

Componentising our front end has some immediate major advantages, such as:

↔ CONSISTENCY

Implementing reusable components helps keep design consistent and can provide clarity in organising code.

</> MAINTAINABILITY

A set of well organised components can be quick to update, and you can be more confident about which areas will and won't be affected.

↑ SCALABILITY

Having a library of components to implement can make for speedy development, and ensuring components are properly namespaced helps to avoid styles and functionality leaking into the wrong place as projects scale.

 Thanks.

Atomic Design by Brad Frost (2013)

Pattern Lab by Dave Olsen and Brian Muenzenmeyer (2017)

Content First – Design Last by Rik Schennink (2015)

Creating Style Guides by Susan Robertson (2014)

How To Automate Style Guide-Driven Development by Varya Stepanova & Juuso Backman (2015)

Vue, React, AngularJS, and Angular2. Our take on popular JavaScript frameworks by Antoni Żółciak (2017)