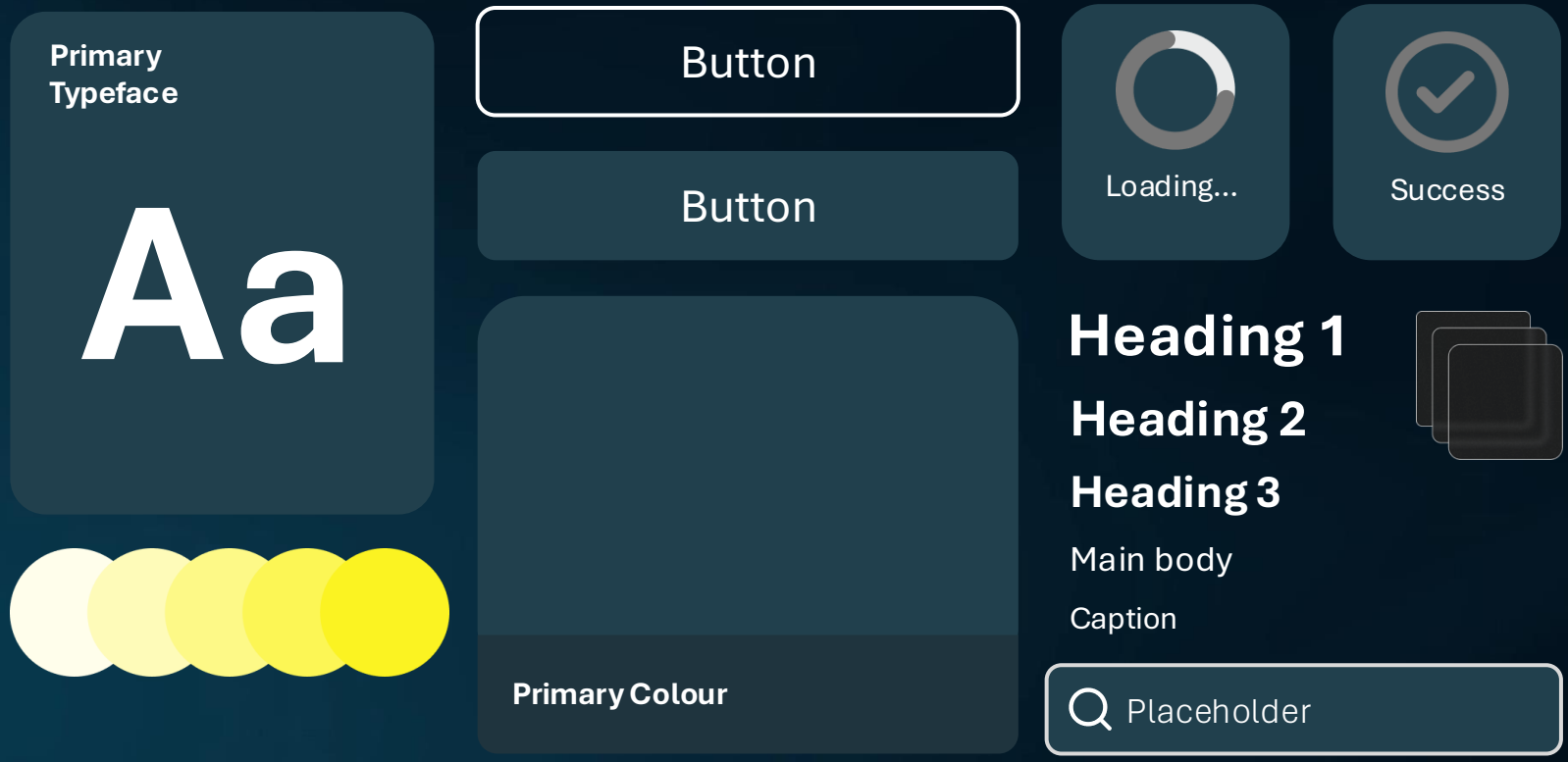
Synechron

Design System Playbook

A practical guide to help anyone understand and grow a design system. For design teams, digital leaders or personal learning.

What is a **design system?**



A design system is a set of standards, to manage and scale design which includes reusable components, design principles and guidelines.

Synechron

Core **elements**

Foundations

Foundations are the color styles, effects, typography, and structure that govern the creation of components within the design system.

Components

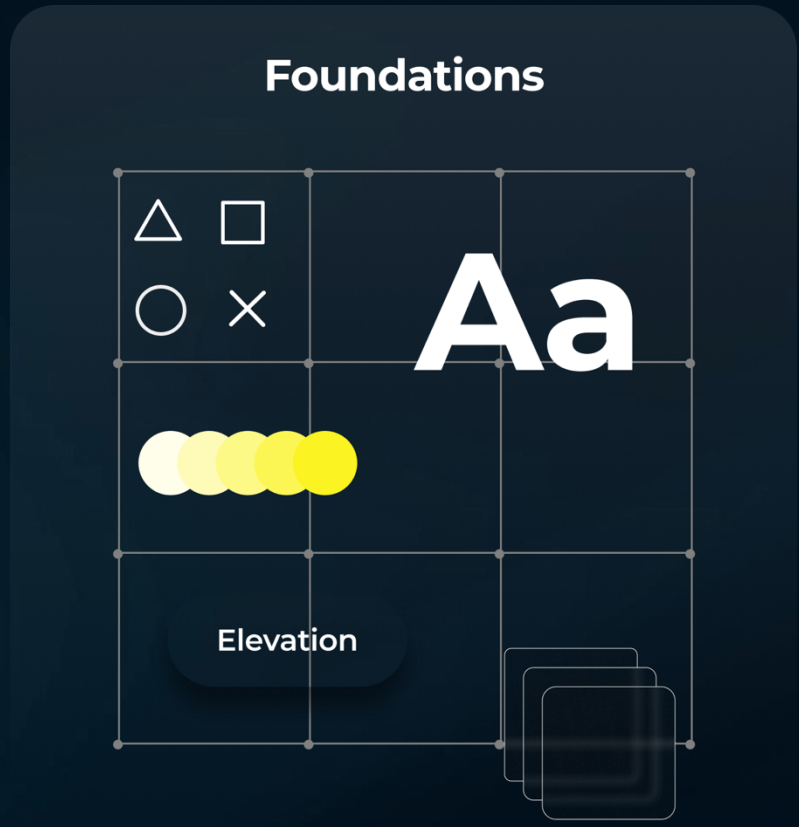
A collection of reusable user interface (UI) elements like buttons, input fields, and menus.

Pattern

A reusable combination of components and templates that address common user objectives with sequences and flows.

Documentation

Instructions on how to use each element, when to use it, and best practices. It ensures consistency and helps new team members get up to speed quickly.



Synechron

Core **elements**

Foundations

Foundations are the color styles, effects, typography, and structure that govern the creation of components within the design system.

Components

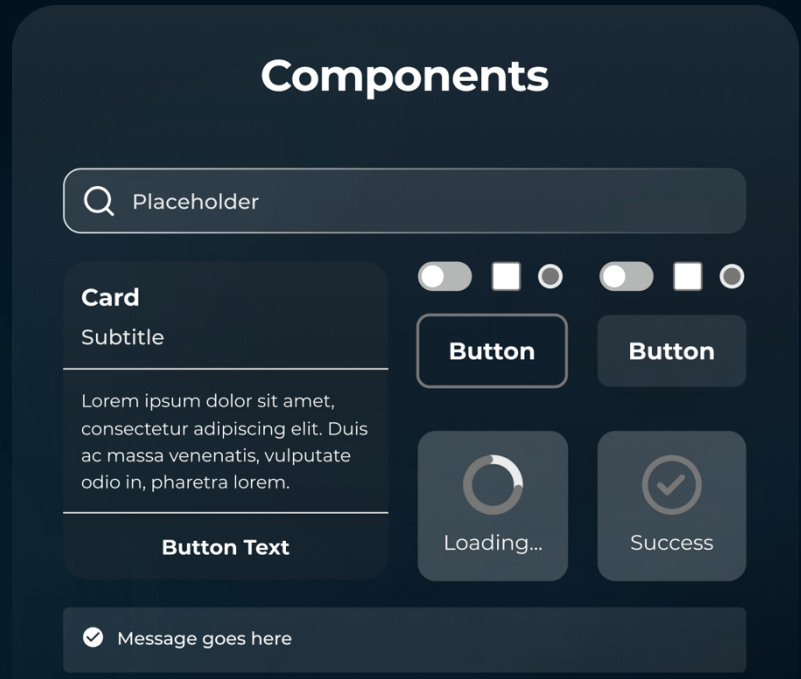
A collection of reusable user interface (UI) elements like buttons, input fields, and menus.

Pattern

A reusable combination of components and templates that address common user objectives with sequences and flows.

Documentation

Instructions on how to use each element, when to use it, and best practices. It ensures consistency and helps new team members get up to speed quickly.



Synechron

Core **elements**

Foundations

Foundations are the color styles, effects, typography, and structure that govern the creation of components within the design system.

Components

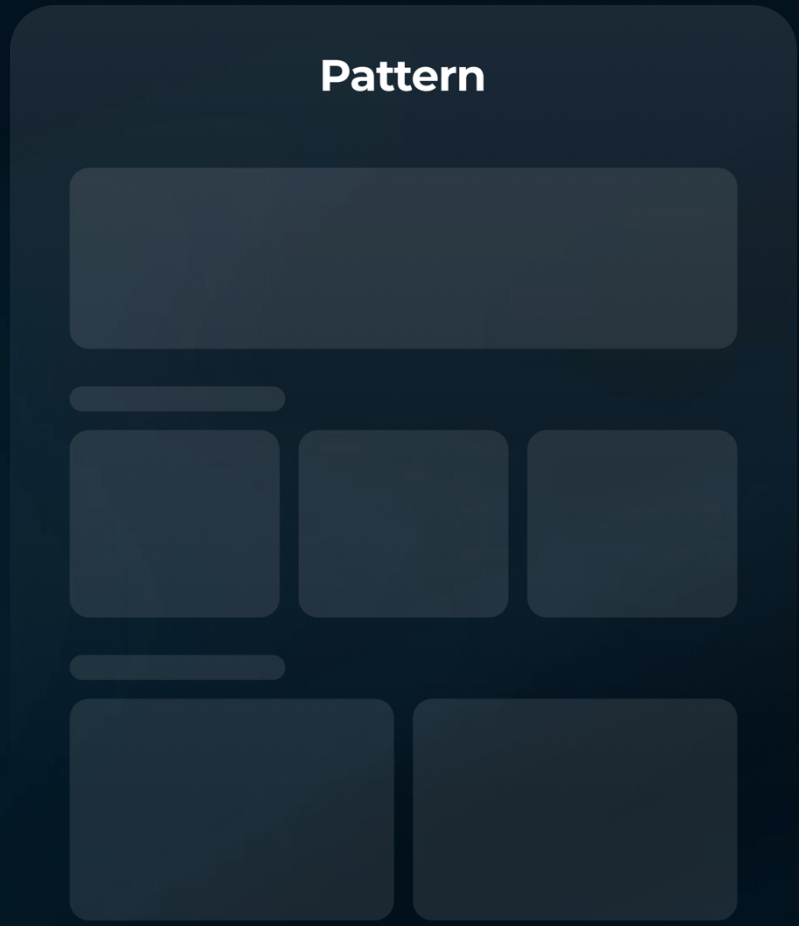
A collection of reusable user interface (UI) elements like buttons, input fields, and menus.

Pattern

A reusable combination of components and templates that address common user objectives with sequences and flows.

Documentation

Instructions on how to use each element, when to use it, and best practices. It ensures consistency and helps new team members get up to speed quickly.



Synechron

Core **elements**

Foundations

Foundations are the color styles, effects, typography, and structure that govern the creation of components within the design system.

Components

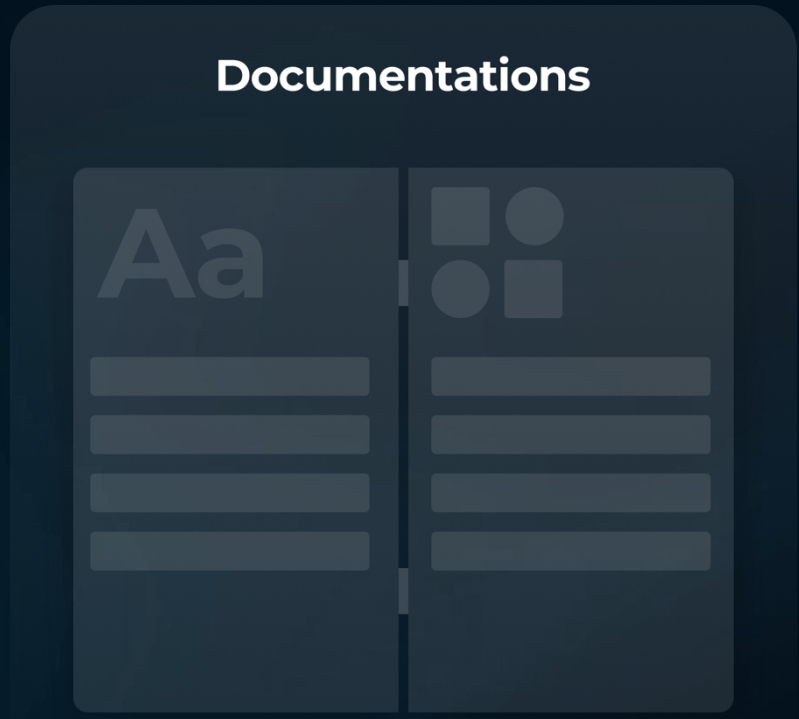
A collection of reusable user interface (UI) elements like buttons, input fields, and menus.

Pattern

A reusable combination of components and templates that address common user objectives with sequences and flows.

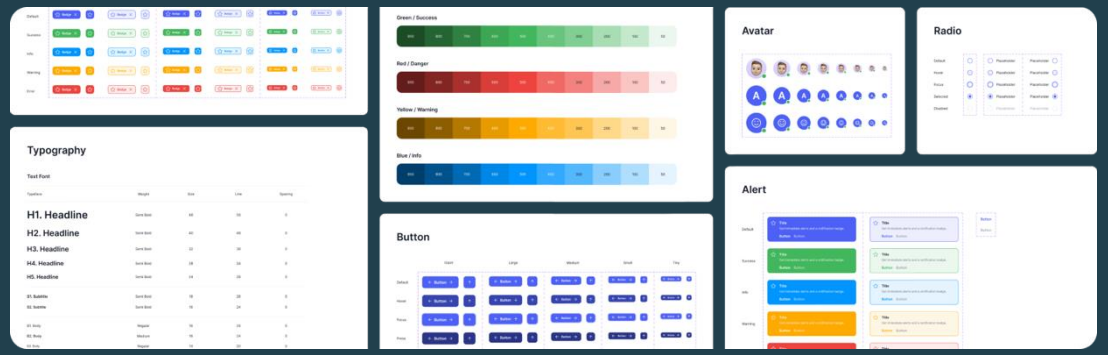
Documentation

Instructions on how to use each element, when to use it, and best practices. It ensures consistency and helps new team members get up to speed quickly.



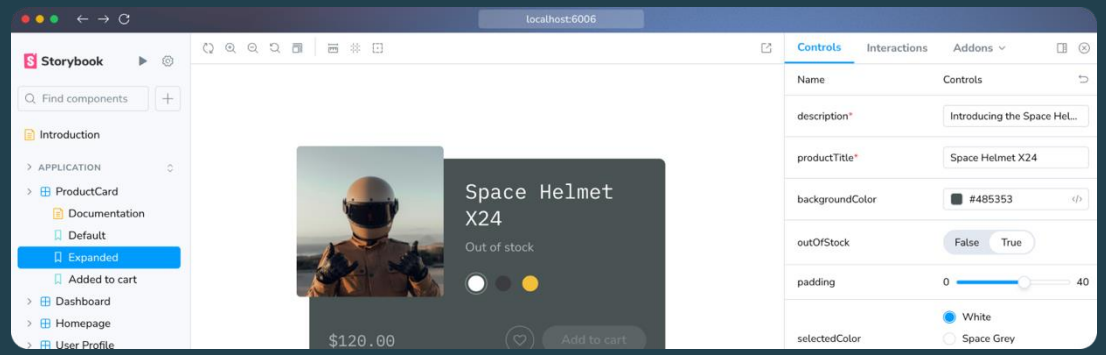
Synechron

Design System **deliverables**



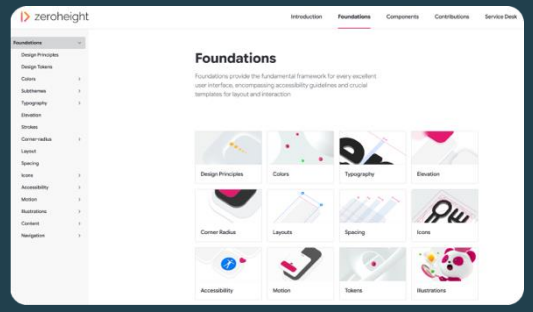
Design Libraries

Ready-to-use design assets and components for creating consistent interfaces.



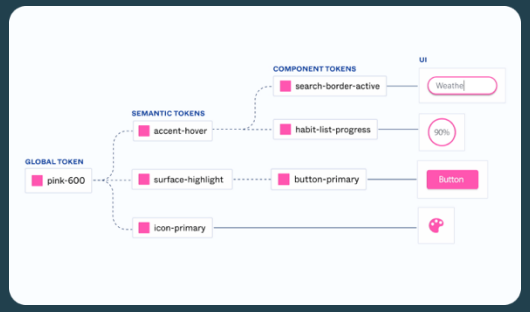
Storybook

A coded component library for developers to build, test, and align UI with design.



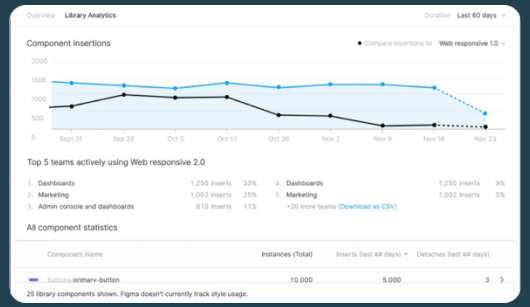
Documentation

Comprehensive guide on using a design system with an explanation of how to use them, best practices and rules.



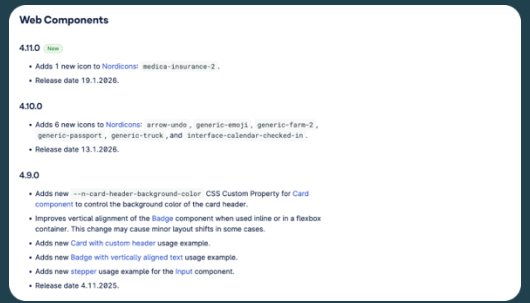
Design Tokens

Shared values (color, spacing, typography) that connect Figma and code for consistency.



Usage Analytics

Measures how the design system is used across projects and shows its impact on the business.



Version Control

Information to track changes, updates, and improvements across design system releases.

Synechron

Synechron

Why do you need a design system?

Why do Design Systems **matter**?

A design system helps the organization work better by aligning designers, developers, brand, and business goals, while improving consistency and efficiency.

Designer

- Ensures every UI element across all products remains consistent
- Speeds up the design process
- Reduces visual inconsistencies and alignment issues

Developer

- Easy to write and maintain test cases
- Easy to upgrade/rollback to versions
- Focus on application business logic

Brand

- Builds a recognizable and unified brand experience
- A consistent visual style increases brand trust and maturity

Company

- Faster time to market
- Reduces the long-term cost of maintaining large product ecosystems

Synechron

Why do Design Systems **matter**?

A design system helps the organization work better by aligning designers, developers, brand, and business goals, while improving consistency and efficiency.

Designer

34%

Faster than those without one

Saving ~6.8 hrs/week
per designer

Source: Figma-Design system 104: Making metrics matter - 2025

Developer

37%

Reduction in development time

Saving ~14.8 hrs/week
per developer

Source: Zeroheight-What is the value of a design system? - 2024

Brand

50%

Time-savings when updating the master file

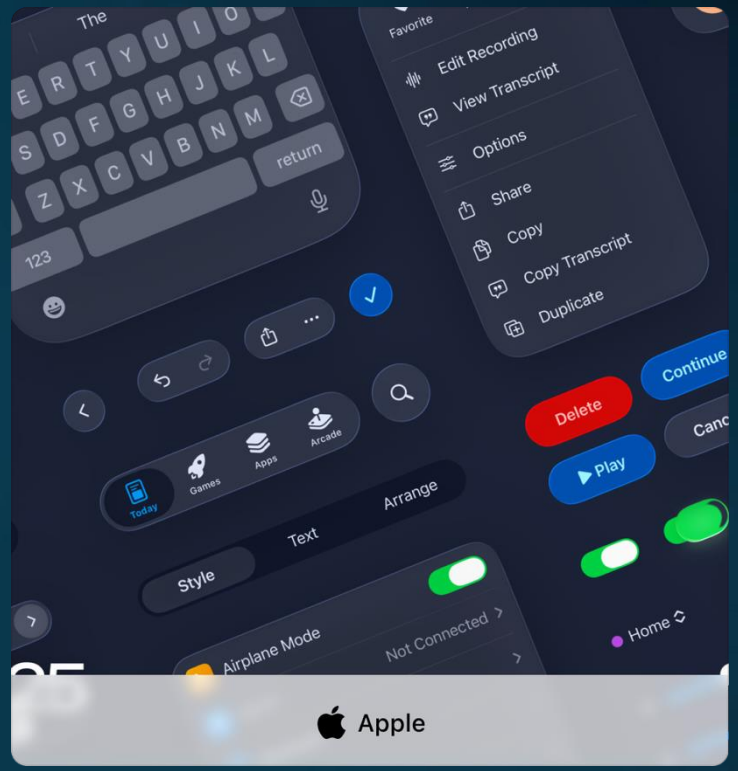
Saving ~12 hrs/week
per designer and developer

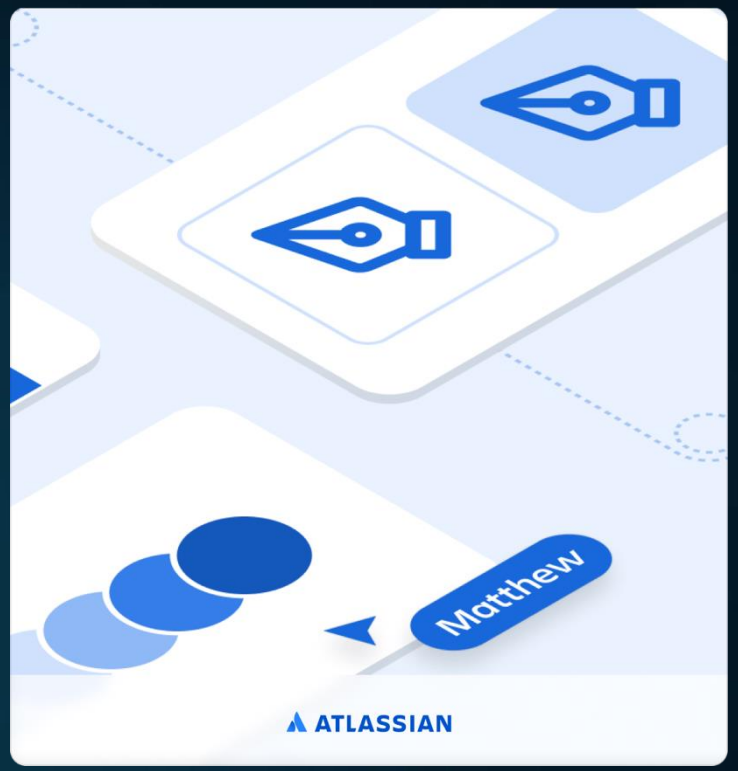
Source: Figma-Vanguard designs - 2024

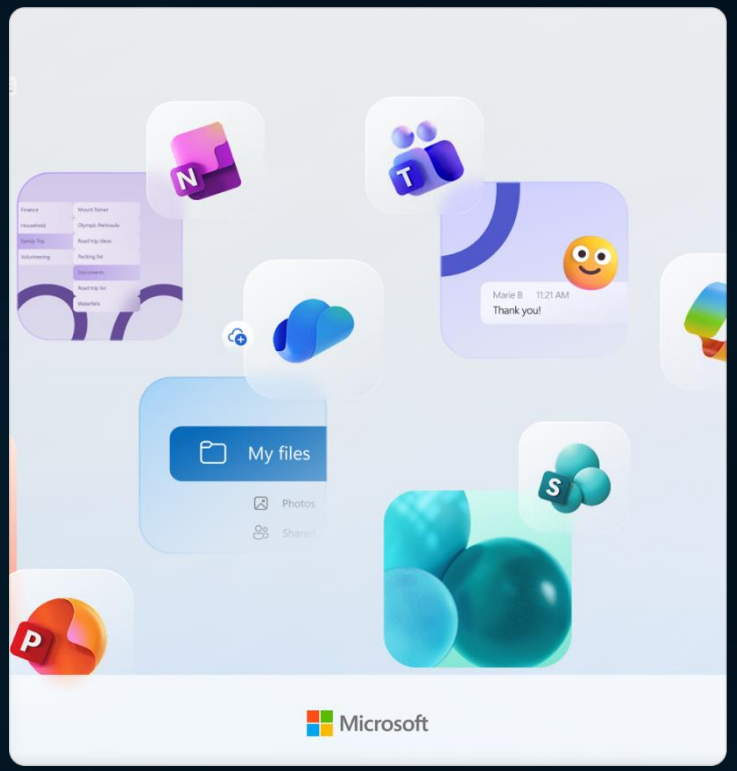
Synechron

Why do Design Systems **matter**?

Top companies with their design system



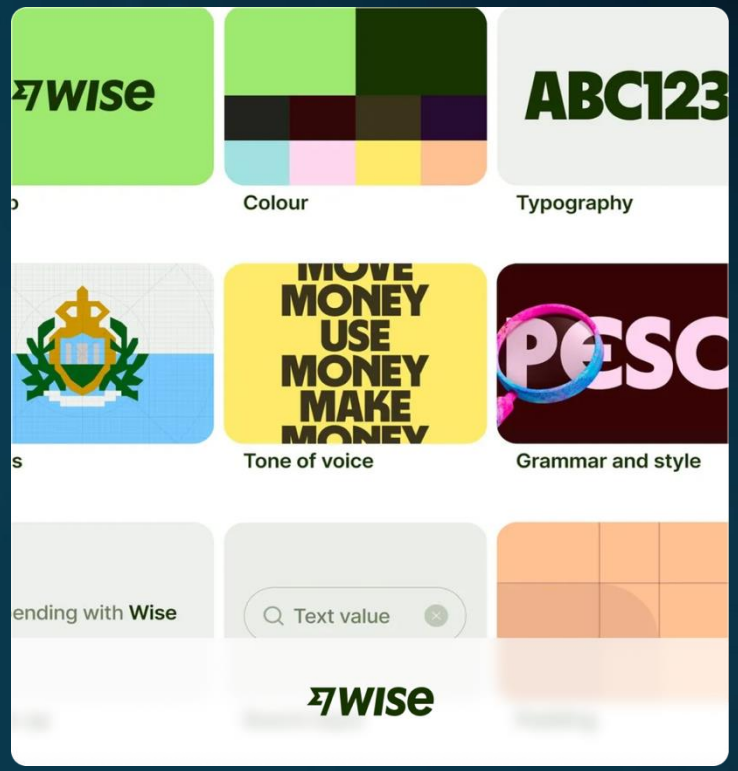


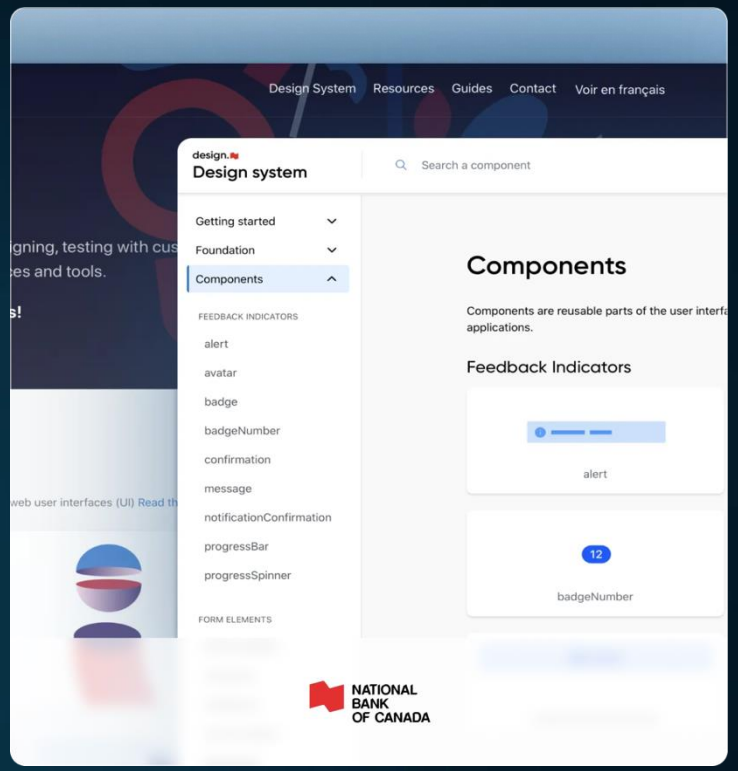


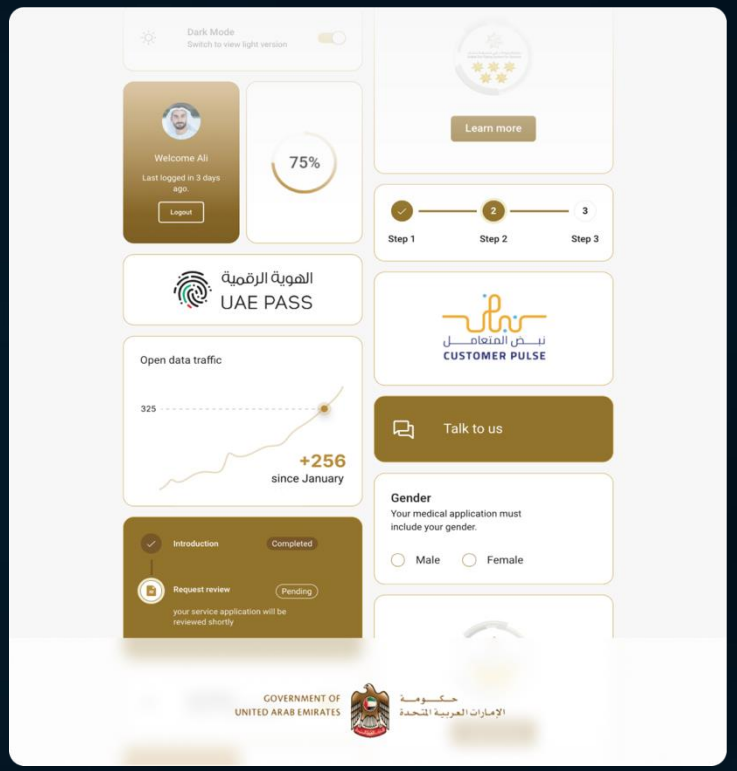
Synechron

Why do Design Systems **matter**?

Design System in **our landscape**







Synechron

Synechron

How do you build a Design System team?

How teams work together

The Design Systems Team has direct responsibility to support product growth and to optimize processes, but the main logic will be the same everywhere:

Following agile processes

Hypothesis > Experiment > Test > Measure. And again ...

Discover problems

Test on multiple platforms, and talk with your coworkers and users.

Talking and brainstorming

There is always more than one possible solution.

Provide support

Be available for feedback—reserve time for educating people.

Have a plan to solve problems

Make some actionable tasks.

Influence

Be the influencer of good practices. People love optimized processes.

4 common team models

When scaling a design system, choosing the right team model is essential for managing growth and maintaining efficiency. There are 4 main models, each suited to different team sizes, workflows, and organizational needs:

Centralized Team Model

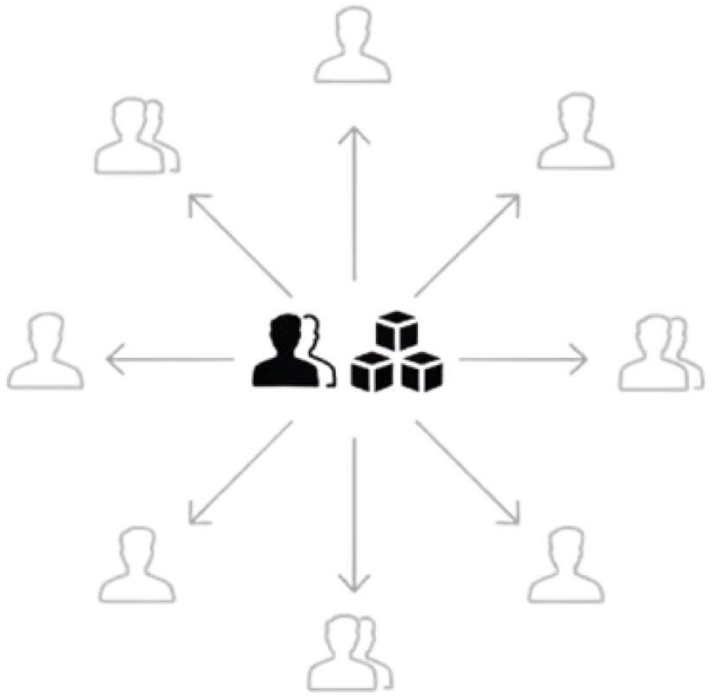
Federated Team Model

Cyclical Team Model

Blended Team Model

Each model has unique trade-offs in collaboration, governance, scalability, and speed. Selecting the right approach depends on your organization's size, maturity, and workload. Next, we'll cover a quick comparison to help you decide.

Setting up the **Team**



Centralized Team Model

One core team builds and manages everything in the design system. All designers and developers depend on this team.

How it works

- A dedicated DS team creates components, tokens, documentation
- Product designers just consume the DS, they don’t contribute
- All decisions come from the DS core team

Pros

- Very consistent
- High quality control
- Clear ownership

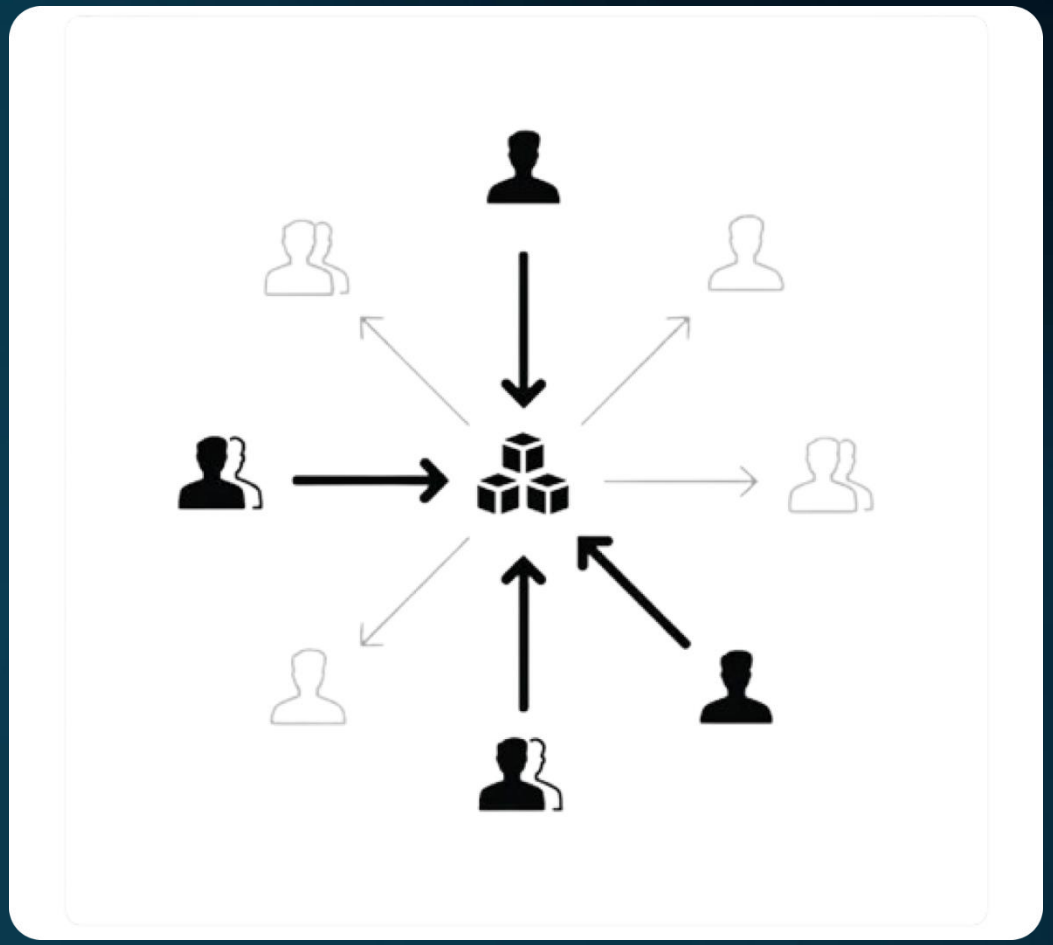
Cons

- DS team becomes bottleneck

Source: https://medium.com/salesforce-ux/the-salesforce-team-model-for-scaling-a-design-system-d89c2a2d404b

Synechron

Setting up the **Team**



Federated Team Model

The DS core team sets the rules and then product designers help build the system. This is the model Salesforce, Shopify, and others use.

How it works

- DS core team owns governance, quality, vision
- Product teams can contribute components, improvements, bug fixes
- Shared responsibility, but centralized quality control

Pros

- Very scalable
- Faster improvements
- Teams feel ownership

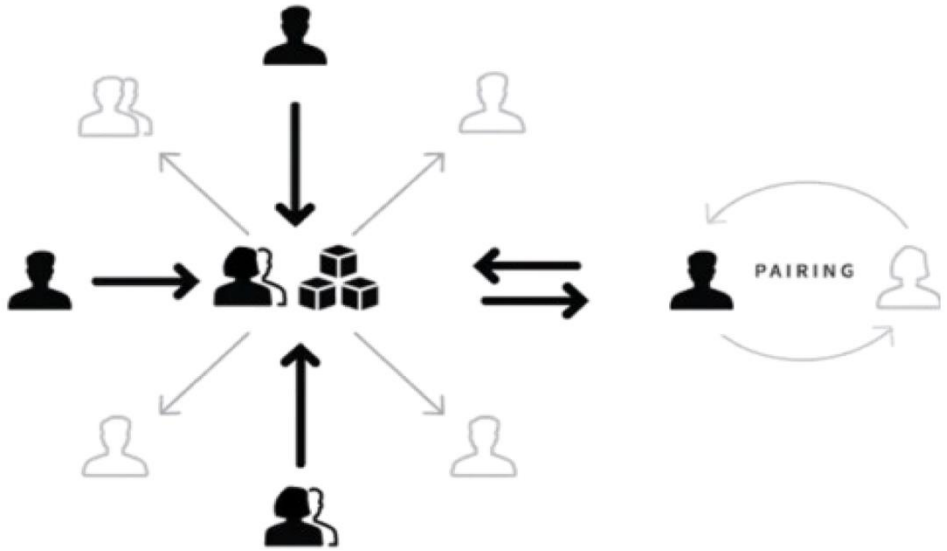
Cons

- Requires strong governance
- Risk of inconsistency if rules are weak

Source: https://medium.com/salesforce-ux/the-salesforce-team-model-for-scaling-a-design-system-d89c2a2d404b

Synechron

Setting up the **Team**



Cyclical Team Model

Designers from different product teams take turns working on the DS. After rotation, designers returned to their product work.

How it works

- Product designers from different product teams rotate into the DS team for a set period
- They build components or improve documentation
- After cycle ends, they return to their product work and next team rotates in

Pros

- Allows everyone to learn DS deeply
- Builds strong alignment
- Shares workload

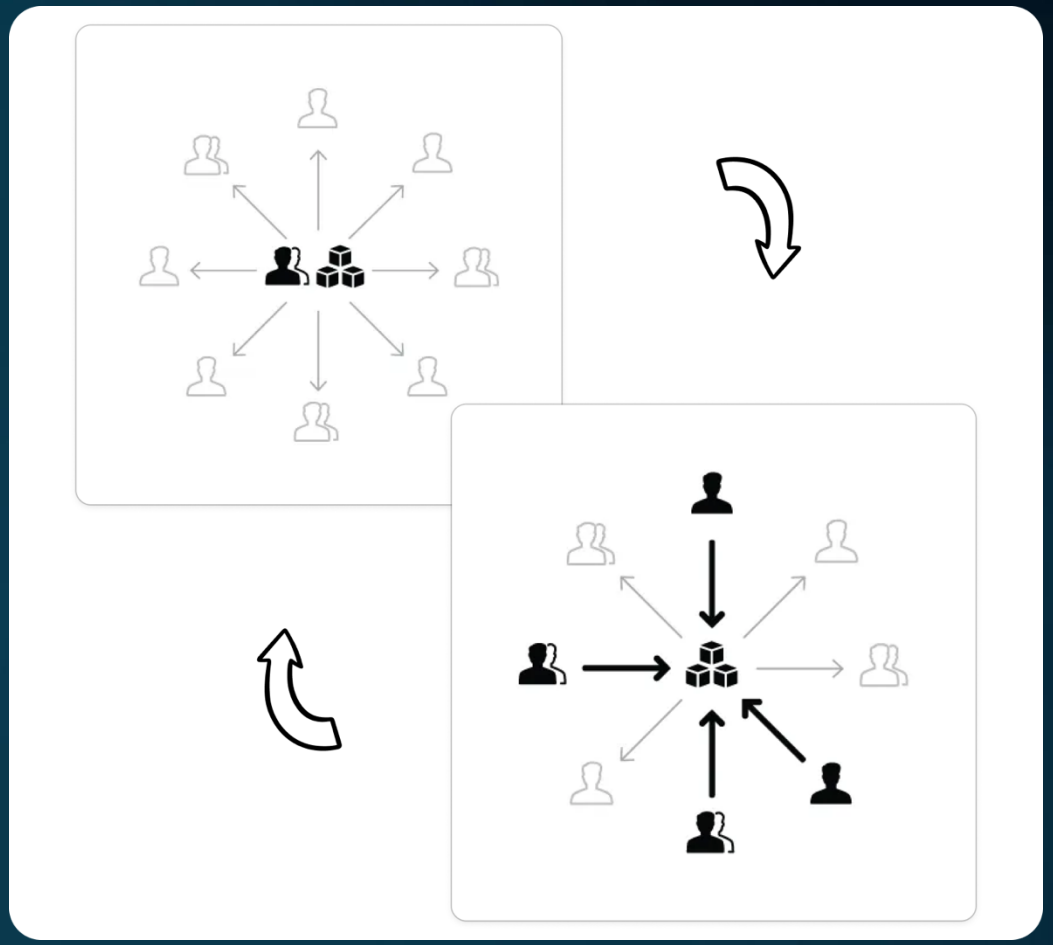
Cons

- Hard to maintain long-term consistency

Source: https://medium.com/salesforce-ux/the-salesforce-team-model-for-scaling-a-design-system-d89c2a2d404b

Synechron

Setting up the **Team**



Blended Team Model

Members of product teams and DS core team work together continuously for a permanent mix, not a cycle.

How it works

- DS team includes embedded or part-time contributors from different product teams
- They split time: 50% DS work, 50% product delivery
- Knowledge flows constantly between product teams and DS

Pros

- Continuous collaboration
- Faster updates
- Product insights flow naturally

Cons

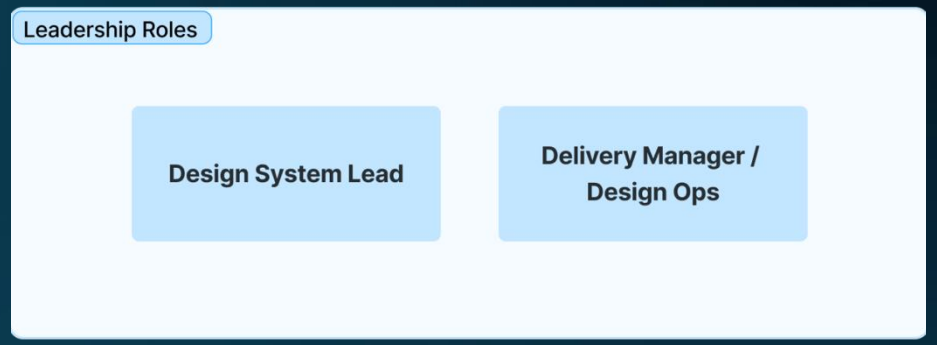
- Hard to manage priorities
- Risk of burnout if not structured well

Source: https://medium.com/salesforce-ux/the-salesforce-team-model-for-scaling-a-design-system-d89c2a2d404b

Synechron

Team Structure and **Responsibilities**

An overview of the key roles involved in building and scaling the design system. Each role plays a distinct part in ensuring the system remains consistent, scalable, and aligned with both product and business goals.



Design System Lead

Setting the roadmap, ensuring the design system aligns with company goals, managing contributions from various teams, and ensuring buy-in from the company.

Delivery Manager / Design Ops

Manages workflows, sprints, and team alignment.

UI Designers

Maintaining the library through designing components, managing design tokens, and ensuring consistency and high quality across all products.

Frontend Developer

Build and maintain the code-based components, ensuring performance, scalability, and seamless integration into various products and platforms.

UX Writer

Support creating clear, comprehensive guides for designers and developers.

UX Researcher

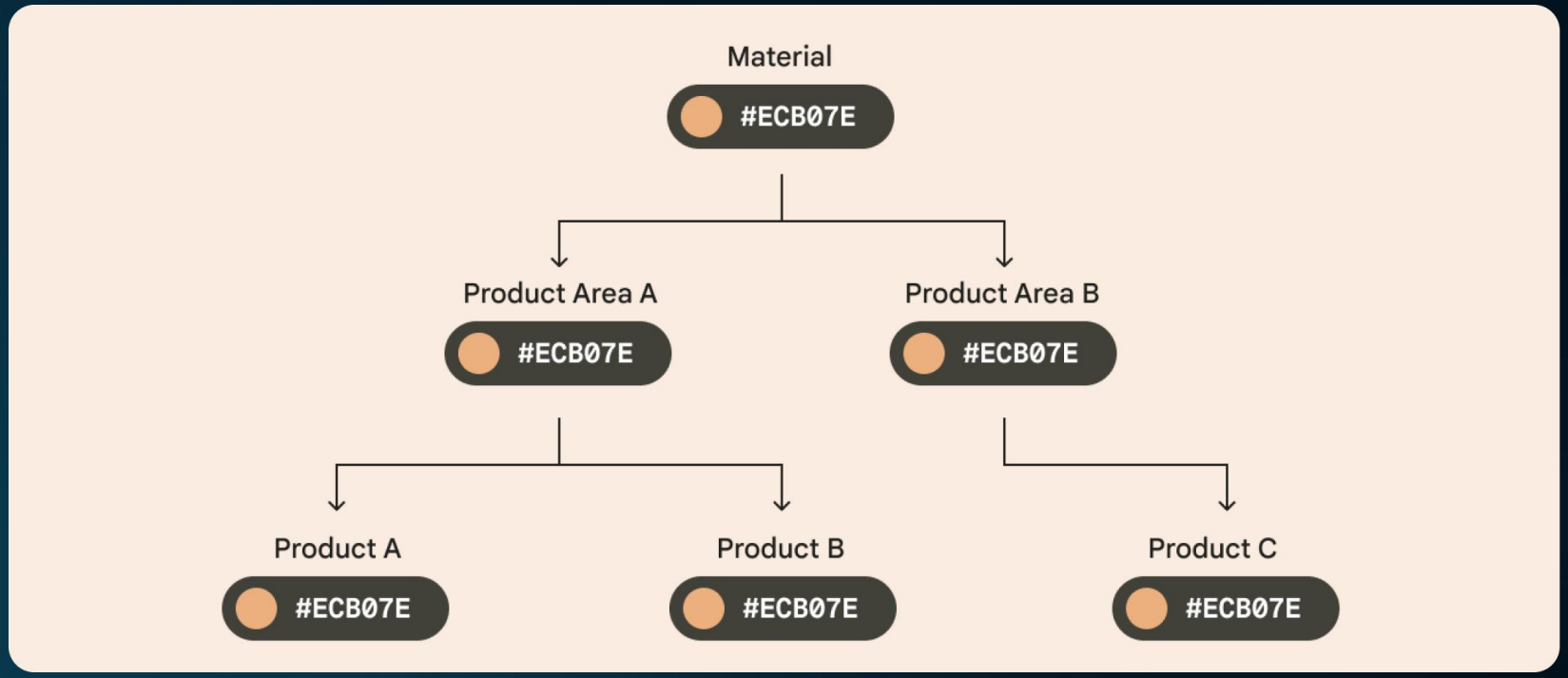
Help gather insights and feedback for optimizing design system and ensure align with user needs/stakeholder.

Synechron

Synechron

Fundamentals of creating a design system

What are **Design Tokens?**



Design tokens are the backbone of Design Systems. They keep the brand's visual style consistent and easy to manage.

Design System **Fundamentals**

Why use **Design Tokens?**

With design tokens, we achieve:

Use of the same language

Improved communication between teams.

Synced files

One source of truth, consistency on all platforms.

Easy maintenance

Edit in one place, update all at once

Solid foundations

Steady design system brings value.

Brand consistency

Creating new products and managing a brand is becoming more accessible, faster, and cheaper

Less design and tech issues

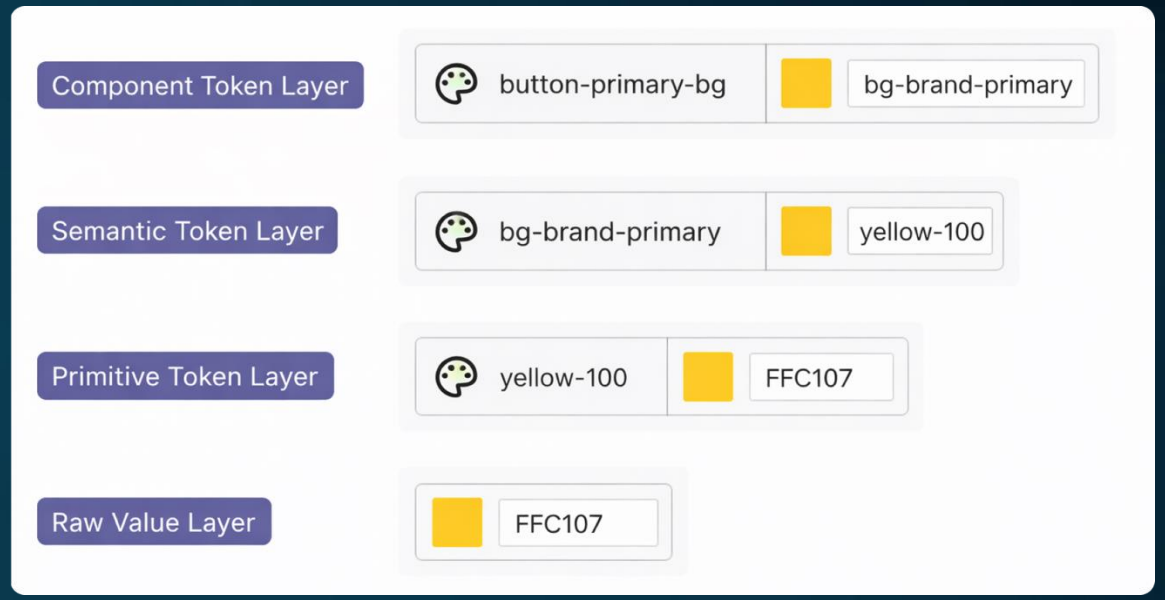
Means fewer resources are used and it mitigates future risk

Synechron

Design System **Fundamentals**

Design Token **Naming Conventions**

Tokens are named based on what they represent and how they are used, not how they look. This ensures discoverability, maintainability, and prevents breaking changes when visual styles evolve.



Names should be

short | meaningful | modular | consistent

Primitive tokens

are the primitive values

yellow-100

Alias tokens

relate to specific context

bg-brand-primary

Component-specific tokens

relate to a component

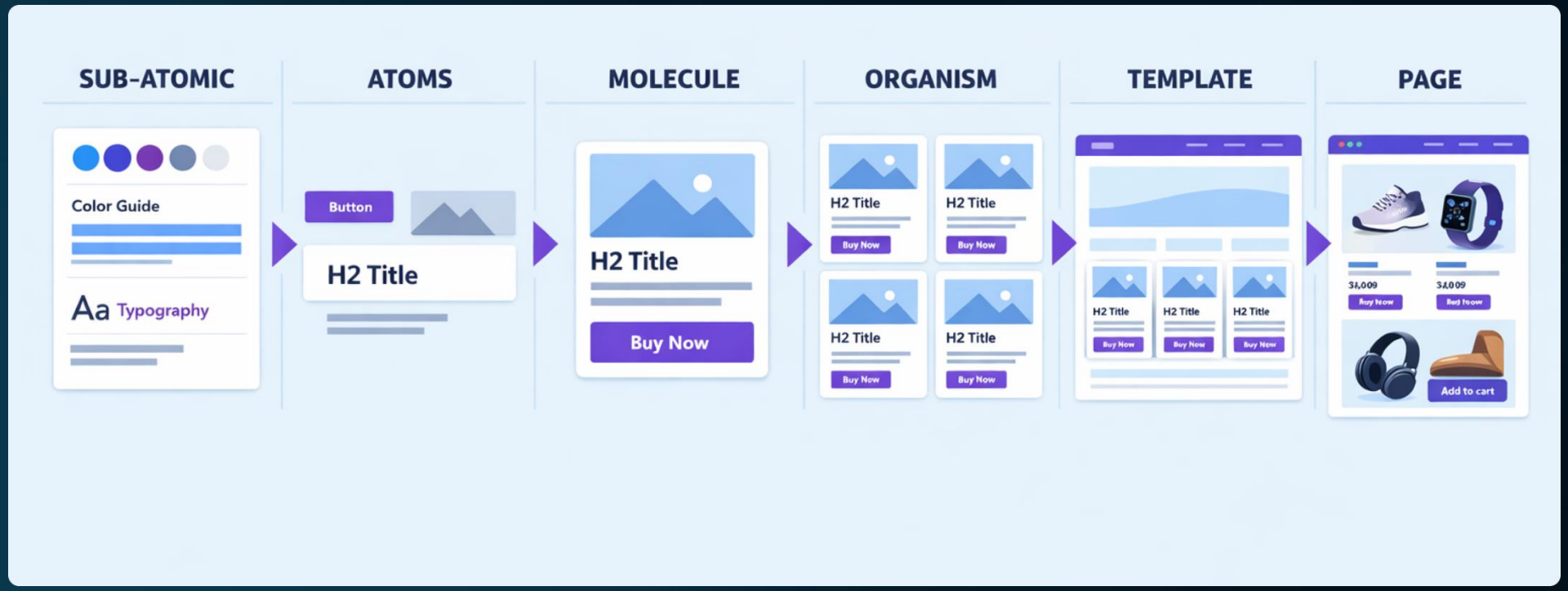
button-primary-bg

Synechron

Design System **Fundamentals**

Atomic **Design**

Atomic design is a framework that helps designers and developers create consistent and scalable design systems without losing creativity. It's like building with a set of perfectly organized LEGO pieces; everything fits, and every piece has a role to play.



Synechron

Create colour, text, effect and layout guide styles

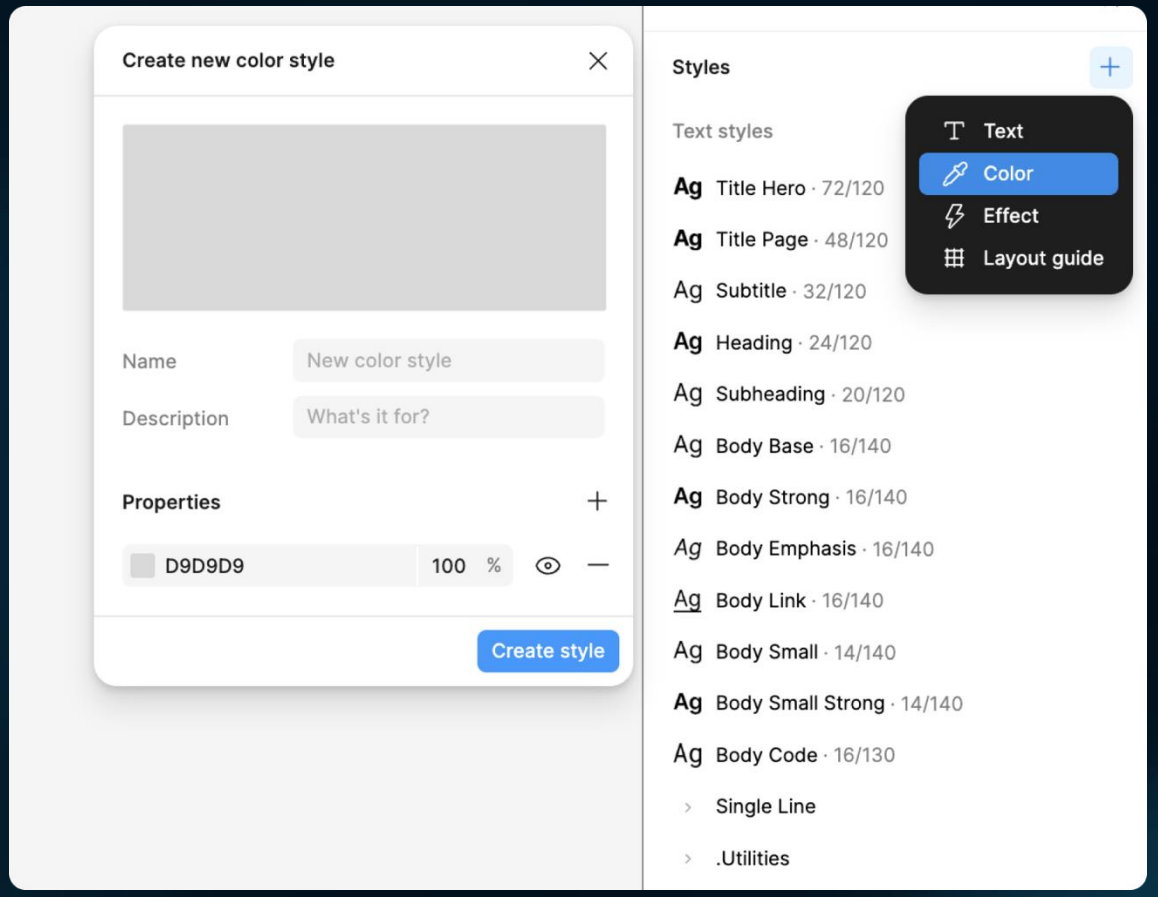
From local styles

Local styles are style definitions that exist in the current file. You can access local styles in the right sidebar when nothing is selected, or from the style picker. Styles are grouped by type:

- Text
- Colour
- Effect
- Layout guide

To create a new local style:

1. In the right sidebar, click ⊕ next to **Local styles**.
2. Select the property you'd like to create a new style for.
3. Give the style a name and description and click **Create style**. Descriptions display when hovering over the style in the style picker.

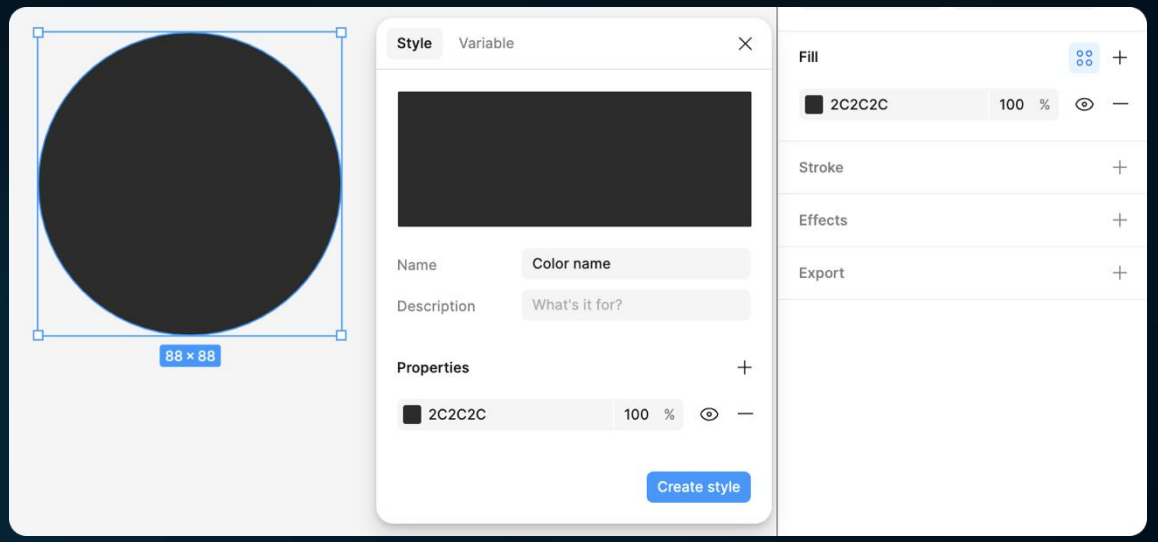


Synechron

Create colour, text, effect and layout guide styles

From an existing object

1. Select the object or frame you'd like to create a style for.
2. In the right sidebar, click ⁘ next to the section corresponding to the style you want to create.
3. Click + in the style picker. In the Create **new style modal**, you can click **Show more options** to view and edit style properties.
4. Give your style a name and description and click **Create style**. Descriptions display when hovering over the style in the style picker.



Synechron

Component Creation in **Figma**

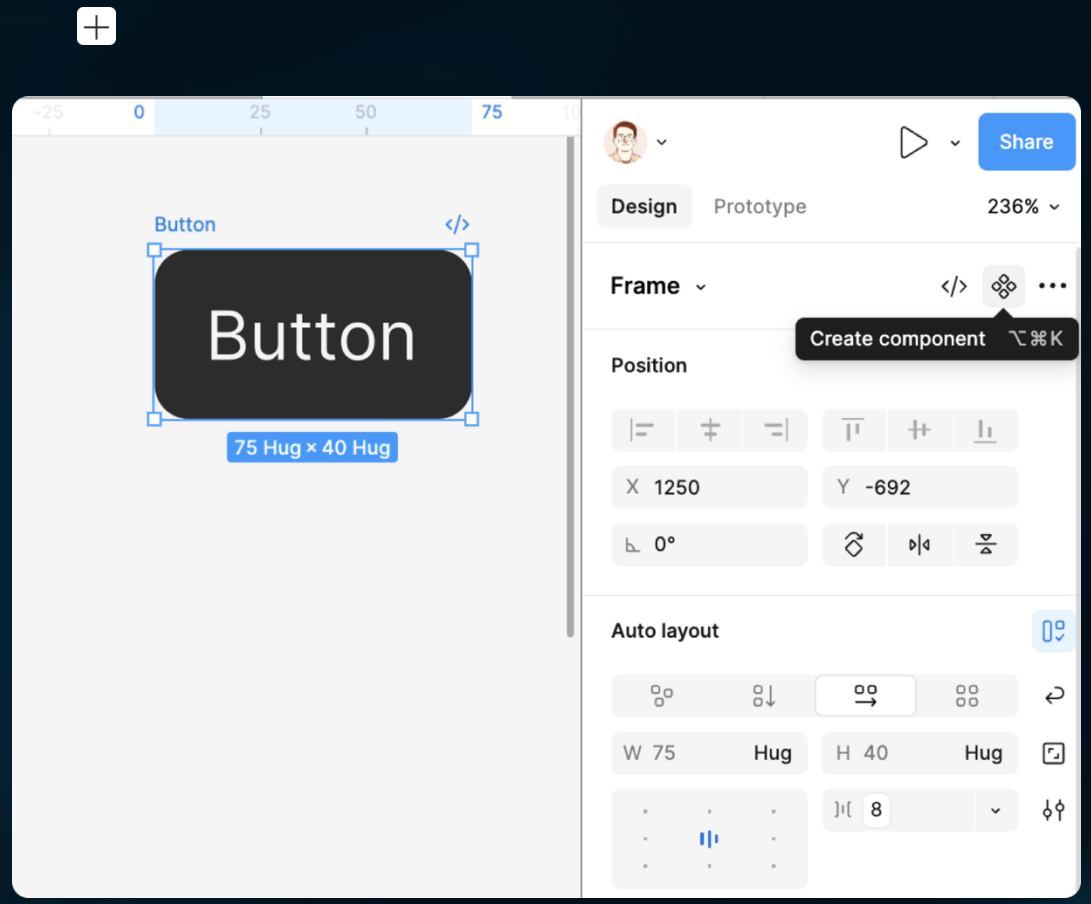
Create a **component**

To create a component:

1. Select the layers you'd like to be included in the component.
2. Do one of the following:

   - Click Create component next to the selection's name in the right sidebar
   - Right-click on your selection and choose **Create component**
   - Use the keyboard shortcut:
   - Mac: Option Command K
   - Windows: Ctrl Alt K

Figma will nest the layers within a special component frame. Identify components in the Layers panel using the purple Component icon. Once created, you can use the Component configuration settings to add a description and documentation link for collaborators.



Synechron

Create components in **bulk**

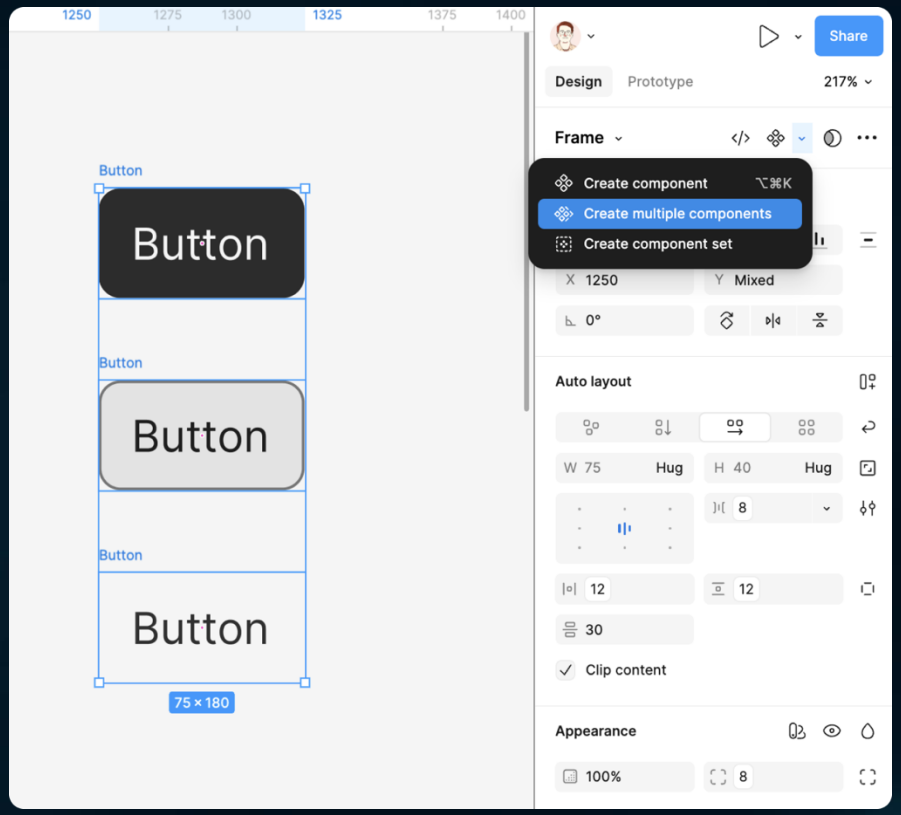
To create components in bulk:

You can also create components in bulk. This allows you to select multiple layers and create components out of them. Create multiple components from:

- Objects and layers in frames
- Objects and layers in groups
- Single layers, like a path or vector network
- Layers in a boolean operation

Note: If you select more than one layer that isn't on one of the above configurations, Figma will create a component for each individual layer.

1. Select the layers you want to create components from,
2. Click ⌄ next to the selection's name in the right sidebar to open the **Create component options** menu.
3. Select **Create multiple components** from the options.
4. Figma will create a component for each frame, group, boolean operation, or path.



Synechron

Delete a component

To delete a component:

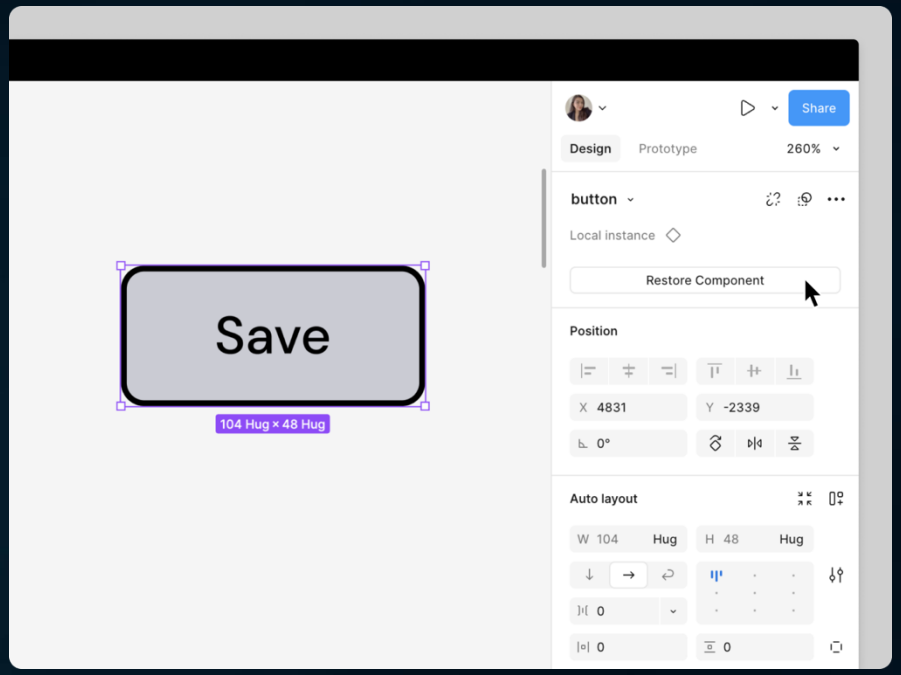
You can delete components at any time. Deleting a main component does not remove instances of that component from your files.

1. Select the component you want to delete.
2. Press **Delete**.

Restore a component

From the right sidebar:

1. Select an existing instance of the deleted component.
2. Do one of the following:
   - If you are in the library file that contained the main component, click **Restore Component** in the right sidebar
   - If you are in a file that did not contain the main component, click Go to **main component in library**. Then click the **Restore** button in the dialog window.

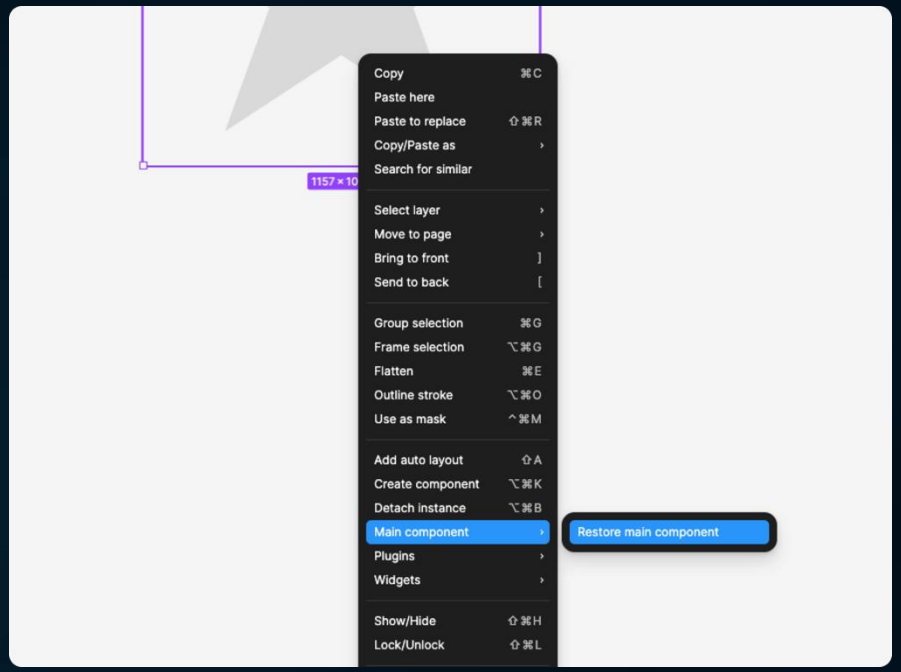


Synechron

Restore a component

From the right-click menu:

1. Right-click on the instance in the canvas.
2. Do one of the following:
   - If you are in the library file that contained the main component, hover over the **Main component** option and click **Restore main component**
   - If you are in a file that did not contain the main component, click **Go to main component**. Then click the **Restore** button in the dialog window

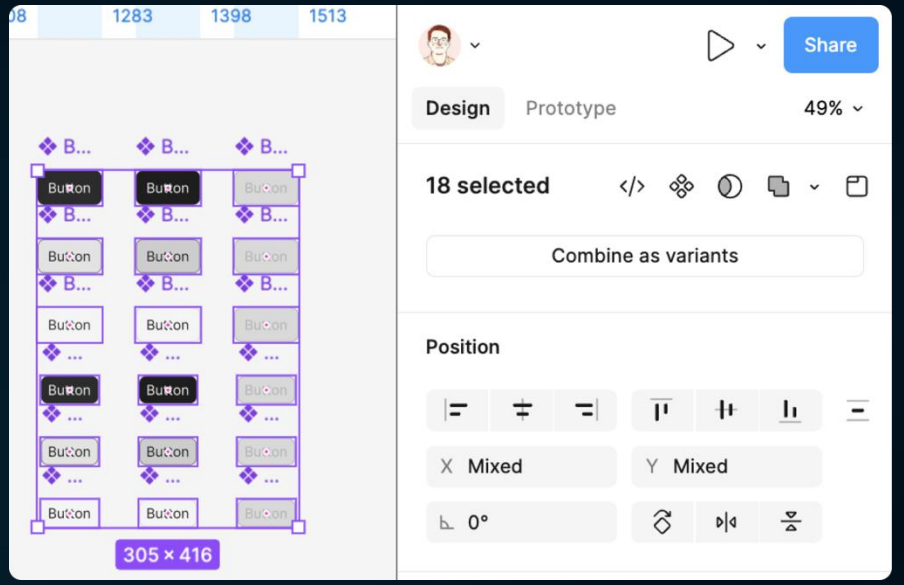


Synechron

Create and use variants

From the right sidebar:

1. Select the components you want to combine
2. In the right sidebar, click **Combine as variants**
3. Figma will add all components to a single component set



Synechron

Create and use variants

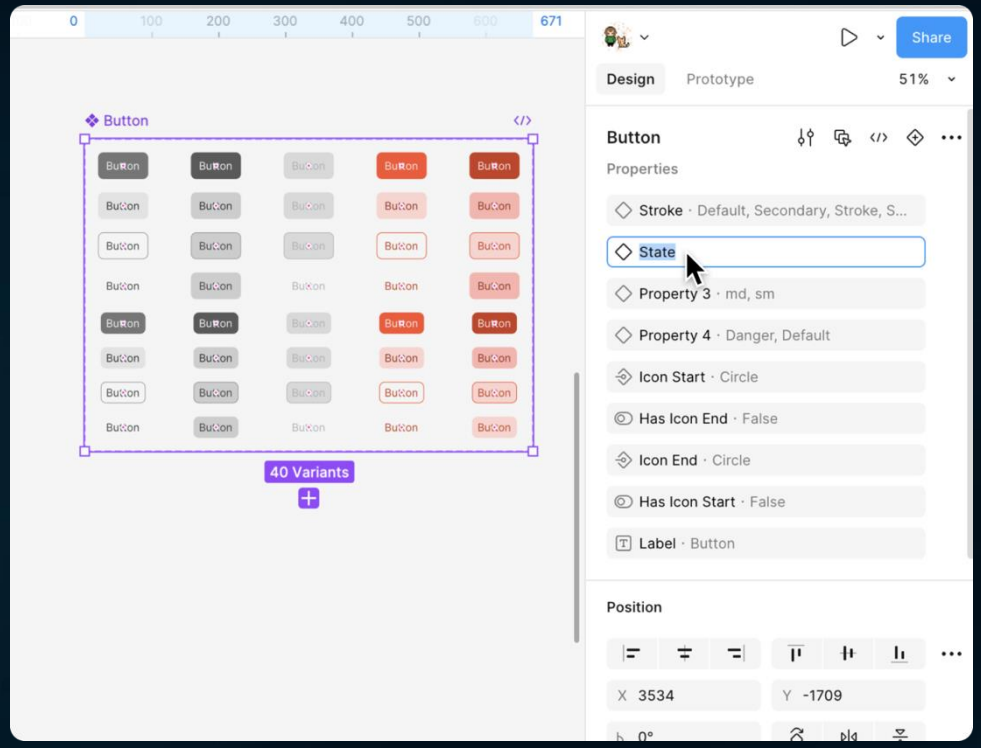
Give properties descriptive names

As part of the conversion process, Figma will create generic properties and add any attributes you added to the component's name as values.

As Figma doesn't know the names of the properties, it will name the first property `Variant` then number them sequentially: `Property 2` then `Property 3` and so on.

You'll need to rename those properties to something more descriptive.

1. Select the component set.
2. Find the Properties section in the right sidebar.
3. Hover over a component property name and double-click it.
4. Edit the name and press **Enter** / **Return** to apply the edit.
5. Repeat for the remaining properties.

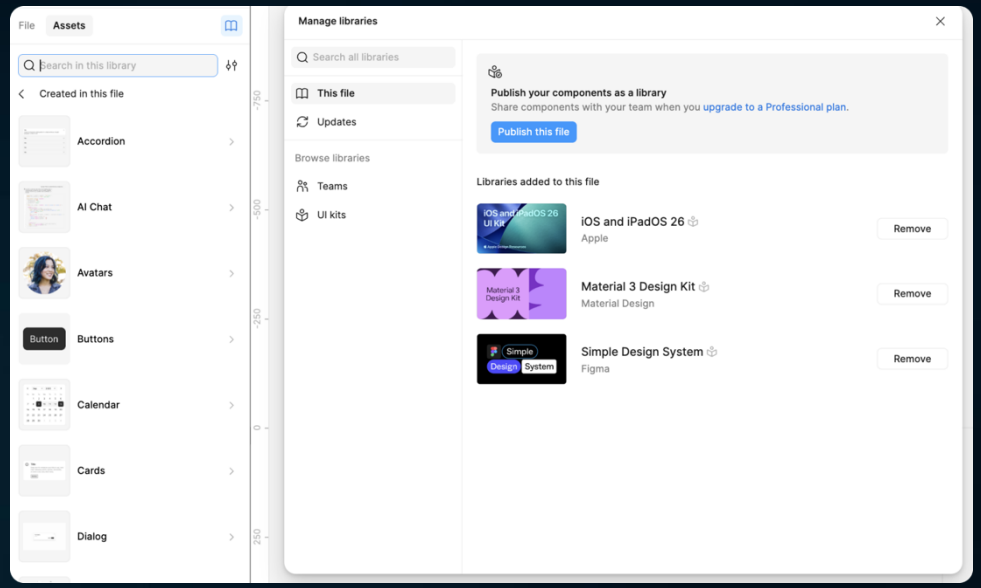


Synechron

Publish a Library

To publish a library

1. In a file with components, styles, or variables you'd like to share, select the **Assets** tab in the left sidebar.
2. Click the **Libraries** icon. The tooltip may read **Review library updates** if there are updates to libraries you are using in the file, or **Review unpublished changes** if you have unpublished updates in your file.
3. From the **This file** section, find your current file and click **Publish**.
4. Add a description of the library's purpose, or a summary of any decisions or changes.
5. From the list of styles, components, or variables that have been added, modified, or removed, uncheck any assets you don't want to publish. You can also uncheck **Changes** to deselect everything.
6. Organization and Enterprise plans only: Use the dropdown menu to choose where to publish the library.
7. Click **Publish**. A notification will appear confirming your library has been successfully published.



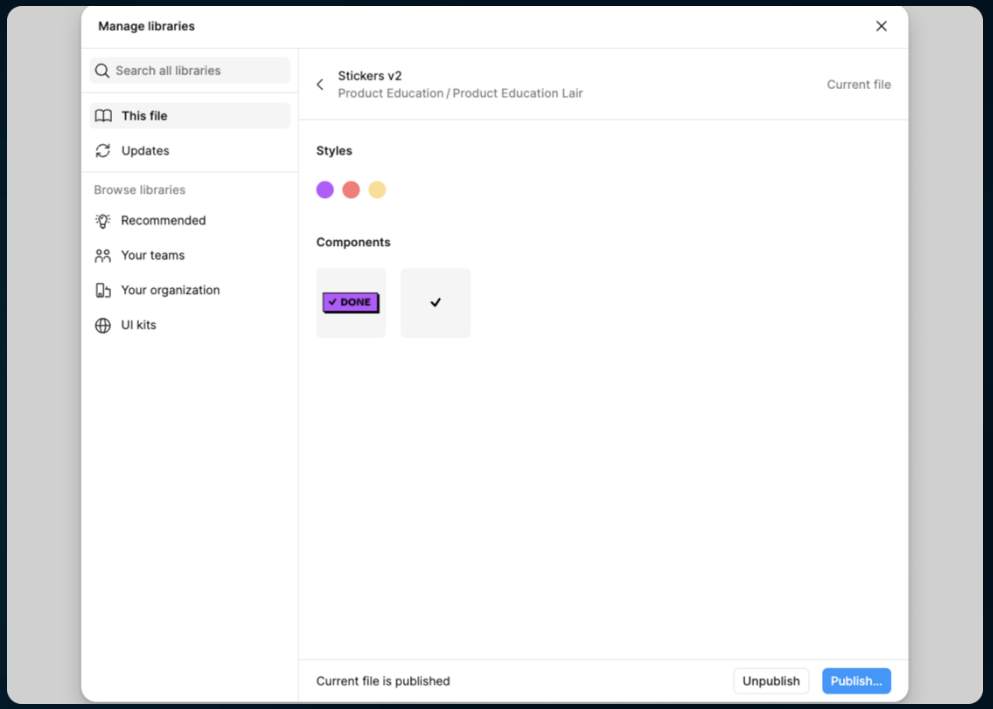
Synechron

Unpublish a Library

To unpublish a library

If you maintain a library of styles, components, and variables, you may need to unpublish a library. This means anyone using this library will no longer be able to access the library or receive updates to its styles, components, and variables.

1. Open the file that you want to unpublish. Make sure you select the library file where your main components and styles are located.
2. Go to the **Assets** tab and click the **Library** icon. The tooltip may read **Review library updates** if there are updates to libraries you are using in the file, or Review unpublished changes if you have unpublished updates in your file.
3. Select the **This file** section and click on the current file to view the styles, components, and variable collections in the library.
4. Click the **Unpublish** button at the bottom of the modal.
5. Click **Remove File from Library** to confirm. Figma will remove that file and any styles and components from the library modal.



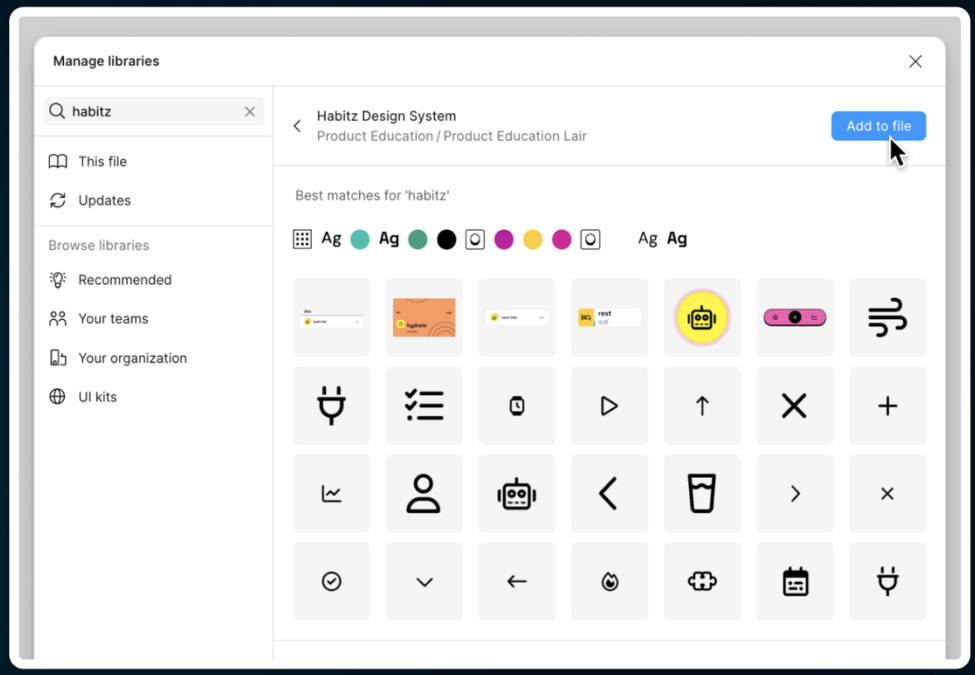
Synechron

Add a library to a design file

To add a library

If you maintain a library of styles, components, and variables, you may need to unpublish a library. This means anyone using this library will no longer be able to access the library or receive updates to its styles, components, and variables.

1. In a design file, select the **Assets** tab in the left sidebar.
2. Click the **Libraries** icon to open the Libraries modal. The icon's tooltip may read **Review library updates** if there are updates to libraries you are using in the file, or **Review unpublished changes** if you have unpublished updates in your file.
3. Locate the library you want to enable. Use the search bar to search your library by name. If available, use the sections under **Browse libraries** to find relevant libraries across your team or organization.
4. Click the library to view its assets. From there you can:
5. Click **Add to file** to enable the library in the file
6. Click **Open file** to view the library file
7. Click ☒ to close the modal.



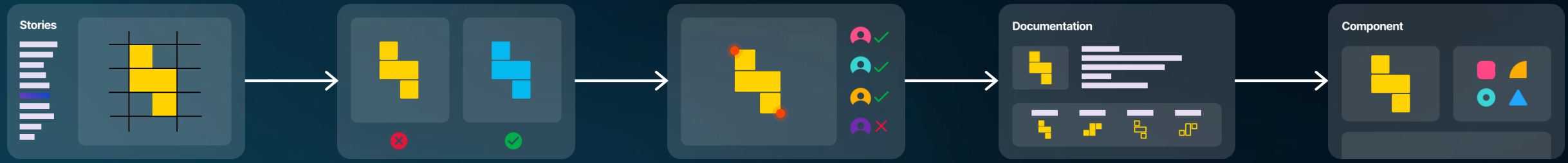
Synechron

Synechron

Collaborate with Developers

Collaborate with **Developers**

Design Workflow



Design

- Requirement gathering
- Research best practices
- Design UI components

Test

- Test components and collect insights
- Apply the changes based on the testing feedback

Review

- Review with Design team
- Review with FE team
- Get feedback, resolve issue

Document

- Usage documentation
- Interaction and motion guidelines

Publish

- Publish approved components
- Announce new changes

Synechron

Synechron

Fundamentals of Design System Development

Design tokens **pipeline**

Figma variables

↓

Tokens studio

↓

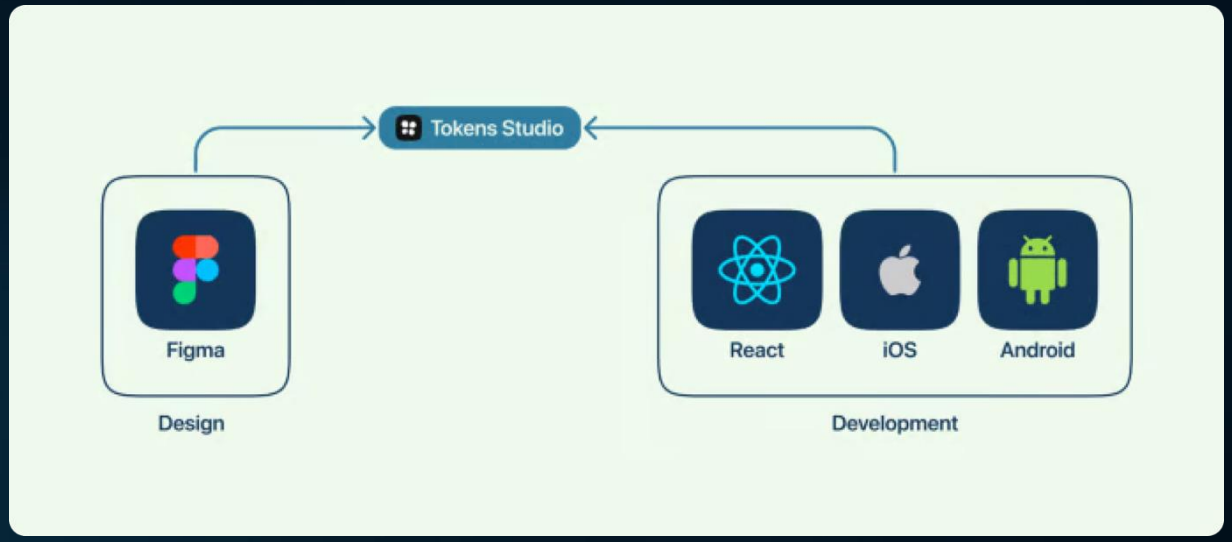
Design Tokens (JSON)

↓

Style Dictionary

↓

Web | Android | iOS



“Design tokens allow Web, Android, and iOS to speak the same visual language without sharing code.”

Design tokens **pipeline**

Figma variables

color-background-button-primary-default

space-md

body-md-regular

At this stage, these live as Figma Variables (with modes like Light/Dark, Brand A/B).

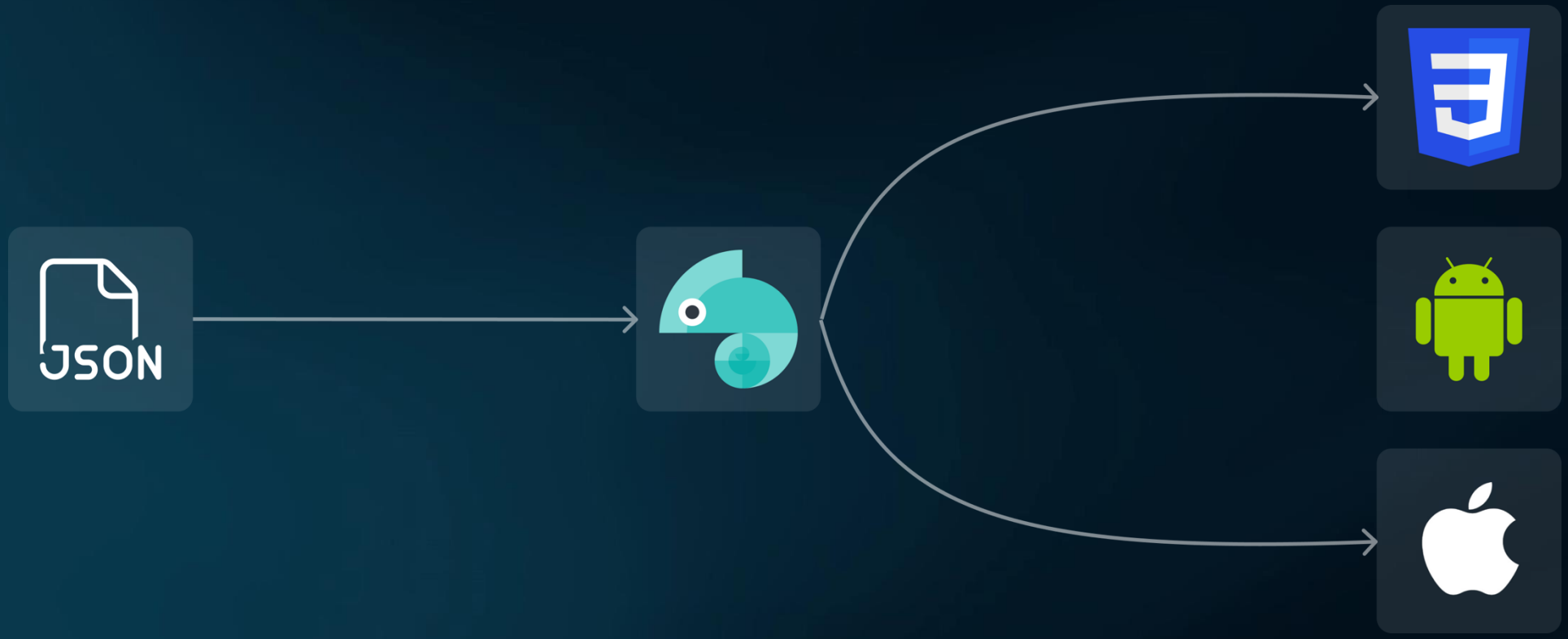
tokens studio

- Single source of truth
- No platform assumptions
- Safe to version & diff in Git

Raw Token Export (JSON)

```
{
  "color": {
    "brand": {
      "primary": {
        "value": "#0052CC",
        "type": "color"
      }
    }
  },
  "spacing": {
    "md": {
      "value": 16,
      "type": "spacing"
    }
  },
  "font": {
    "size": {
      "body": {
        "value": 14,
        "type": "fontSize"
      }
    }
  }
}
```

Design tokens **pipeline**



Design System Development **Fundamentals**

Tokens usage - **Web**

CSS Variables

```
:root {
  --color-brand-primary: #0052cc;
  --spacing-md: 16px;
  --font-size-body: 14px;
}
```

```
<button style={{ backgroundColor: 'var(--color-brand-primary)' }}>
  Primary Button
</button>
```

Theming via Token Modes (Light / Dark)

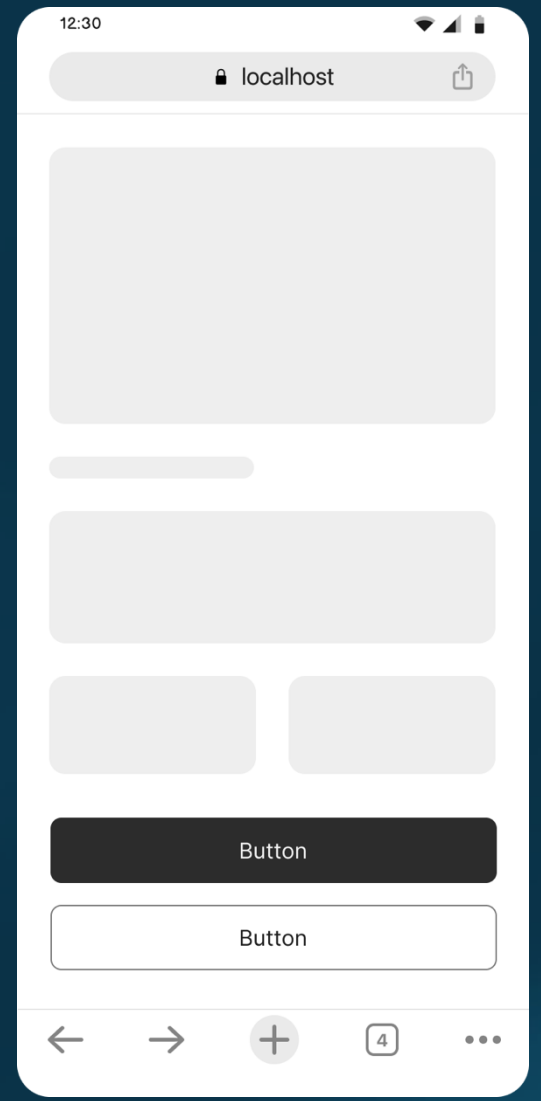
```
[data-theme='light'] {
  --color-brand-primary: #0052cc;
}

[data-theme='dark'] {
  --color-brand-primary: #4c9aff;
}
```

Using JS/TS Token Output

```
export const spacing = {
  md: 16
};
```

Using JS/TS Token Output

```
<Card style={{ padding: spacing.md }} /
>
```



Synechron

Design System Development **Fundamentals**

Tokens usage - **Android**

Android (XML)

```xml
<resources>
  <color name="color_brand_primary">#0052CC</color>
  <dimen name="spacing_md">16dp</dimen>
  <dimen name="font_size_body">14sp</dimen>
</resources>
```

Using Tokens in XML Layouts

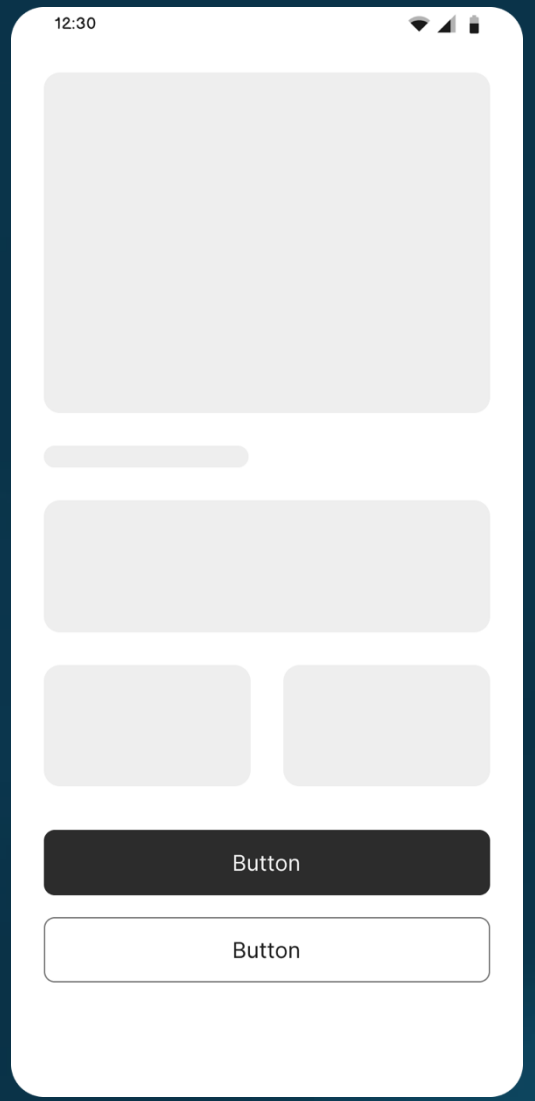
```xml
<!-- res/layout/button_primary.xml -->
 <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:backgroundTint="@color/color_brand_primary"
    android:padding="@dimen/spacing_md"
    android:textSize="@dimen/font_size_body"
    android:text="Primary Button"
/>
```

Android theming Example (Light/Dark)

```xml
<!-- values/themes.xml -->
<style name="Theme.App.Light">
    <item name="colorPrimary">@color/
color_brand_primary</item>
</style>

<!-- values-night/themes.xml -->
<style name="Theme.App.Dark">
    <item name="colorPrimary">@color/
color_brand_primary_dark</item>
</style>
```

Using Tokens in Kotlin (Programmatic UI)

```kotlin
val button = Button(context).apply {
    setBackgroundColor(
        ContextCompat.getColor(context, R.color.color_brand_primary)
    )
    setPadding(
        resources.getDimensionPixelSize(R.dimen.spacing_md)
    )
    textSize = resources.getDimension(R.dimen.font_size_body)
}
```



Synechron

Design System Development **Fundamentals**

Tokens usage - **iOS**

iOS (Swift)

```swift
class Tokens {
  static let colorBrandPrimary = UIColor(
    red: 0.0,
    green: 0.32,
    blue: 0.8,
    alpha: 1.0
  )

  static let spacingMd: CGFloat = 16
  static let fontSizeBody: CGFloat = 14
}
```

Using Tokens in SwiftUI

```swift
Button("Primary Button") {
    print("Clicked")
}
.padding(Tokens.Spacing.md)
.font(.system(size: Tokens.FontSize.body))
.background(Color(Tokens.Color.brandPrimary))
```

Using Tokens in UIKit

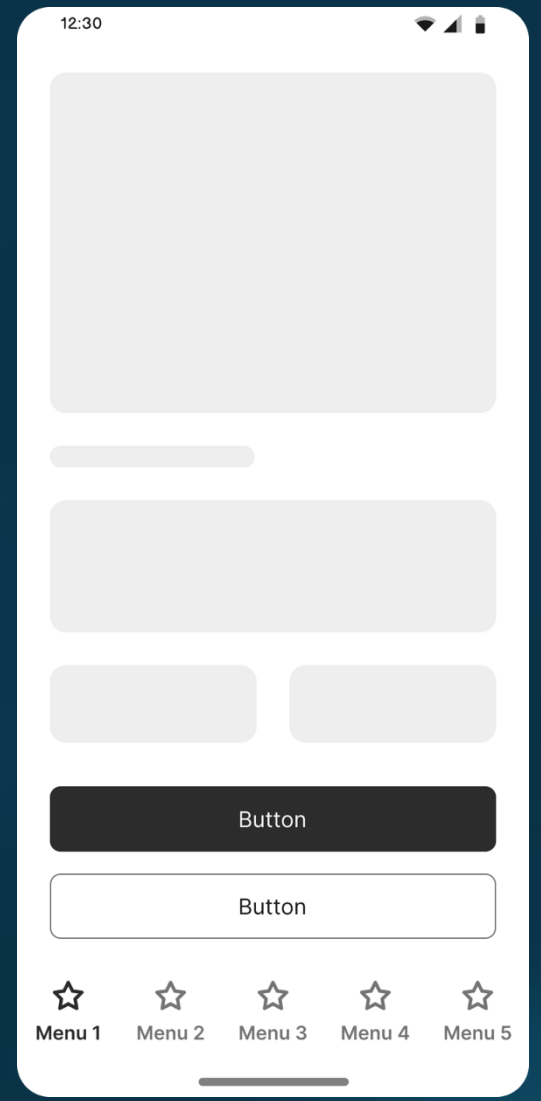
```swift
let button = UIButton(type: .system)
button.backgroundColor = Tokens.Color.brandPrimary
button.contentEdgeInsets = UIEdgeInsets(
    top: Tokens.Spacing.md,
    left: Tokens.Spacing.md,
    bottom: Tokens.Spacing.md,
    right: Tokens.Spacing.md
)
button.titleLabel?.font = UIFont.systemFont(
    ofSize: Tokens.FontSize.body
)
```

iOS Dynamic Colors (Light / Dark Mode)

```swift
extension UIColor {
    static let brandPrimary = UIColor { trait in
        trait.userInterfaceStyle == .dark
        ? UIColor(red: 0.3, green: 0.6, blue: 1.0, alpha: 1.0)
        : UIColor(red: 0.0, green: 0.32, blue: 0.8, alpha: 1.0)
    }
}
```

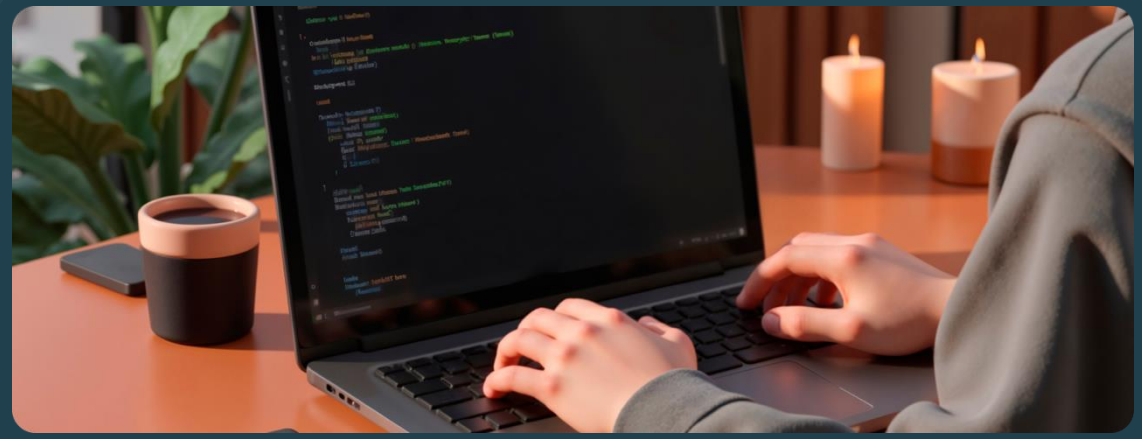
```swift
button.backgroundColor = .brandPrimary
```

Same token, adaptive behavior based on system appearance



Synechron

Key **Takeaways**



Developer

- No hardcoded values
- Predictable APIs
- Easier refactors
- Safer upgrades

Client Manager

- Faster delivery
- Consistent branding
- Reduced UI bugs
- Platform scalability

Synechron

Core Principles of Design System Development

Core Principles & **Guidelines**

S	O	L	I	D
Single Responsibility	Open/Closed Principle	Liskov Substitution	Interface Segregation	Dependency Inversion





Core **Principles**

S - **Single Responsibility** Principle

One component should do one job really well

```
function SubmitButton() {
  const dispatch = useDispatch();
  const [loading, setLoading] = useState(false);

  const handleClick = async () => {
    setLoading(true);
    await dispatch(saveUser());
    setLoading(false);
  };

  return <button disabled={loading}>Save</button>;
}
```

- Handles UI rendering
- Handles business logic / API calls
- Handles state management for loading

```
// DS component
function Button({ label, onClick, disabled }) {
  return <button disabled={disabled} onClick={onClick}>{label}</button>;
}

// App handles logic
const App = () => {
  async function handleSave() {
    await dispatch(saveUser());
  }
  return <Button label="Save" onClick={handleSave} />
}
```

- Button only knows how to look and behave
- App handles what happens when clicked
- One reason to change → either UI or business logic, not both

Synechron

Core **Principles**

O – **Open/Closed** Principle

Components should be open for extension, but closed for modification.

- Add variants instead of editing existing behavior.
- Extend via props, not hacks.

```
1  <Button style={{ background: 'red' }} />
```

```
1  <Button variant="secondary" size="sm" />
```

Existing screens don’t break - Backward compatibility is preserved

Synechron

Core **Principles**

L – **Liskov Substitution** Principle

If it looks like a Button, it should behave like a Button.

- All variants respect the same API contract
- No surprise behavior.

```
1  <DangerButton disabled />
```

```
1  <Button disabled />
2  <Button variant="danger" disabled />
```

Predictable components - Developer trust

Synechron

Core **Principles**

I – **Interface Segregation** Principle

Don’t force components to accept props they don’t need.

- Small, focused props
- No bloated components

```
<FormField type="input" type2-  ect" type3="checkbox"
  {...50 props}
/>
```

```
<Input value="" onChange={...} />
<Select options={[]} onChange={...} />
```

Easier to learn - Cleaner APIs

Synechron

Core **Principles**

D – **Dependency Inversion** Principle

UI components should depend on abstractions, not app logic. A high-level module (component) should not depend on low-level details (like Redux, API calls, or app logic).

```
function Button() {
  const dispatch = useDispatch();

  return (
    <button onClick={() => dispatch(saveUser())}>Save</button>
  );
}

```

- Button now depends on Redux → low-level detail
- Cannot reuse Button in another app without Redux
- Tight coupling → harder to test and maintain

```
function Button({ onClick }) {
  return <button onClick={onClick}>Save</button>;
}

// App decides what to do
<Button onClick={handleSave} />

```

- Button depends on an abstraction (onClick)
- App provides implementation details
- Component is reusable in any app or context

Synechron

Core **Principles**

Code reusability

One component reused everywhere with different props.

Base Element: **Button**

Design system component

```
function Button({ label, variant = "primary", size = "md" }) {
  return (
    <button className={`btn ${variant} ${size}`}>
      {label}
    </button>
  );
}
```

Consuming in application

```
const App = () => {
return <div>
  <Button label="Save" />
  <Button label="Delete" variant="danger" />
  <Button label="Next" size="sm" />
</div>
}
```

Same component - Different contexts - No duplication

Synechron

Core **Principles**

Scalability

System grows safely without breaking existing UI

New requirements → “We need a loading state on Button”

Updated Design system component

```
function Button({ label, loading, variant = "primary", size = "md" }) {
  return (
    <button className={`btn ${variant} ${size}`}>
      {loading ? "loading..." : label}
    </button>
  );
}
```

Extend, don’t rewrite.

Consuming in application

```
const App = () => {
return <div>
  <Button label="Save" />
  <Button label="Delete" variant="danger" />
  <Button label="Next" size="sm" loading />
</div>
}
```

Old buttons still work alongside new button variant.

New feature added - No breaking changes - Existing screens untouched

Synechron

Core **Principles**

Accessibility

Design system follows WCAG standards and global accessibility regulations to ensure apps are usable by everyone, everywhere.

Base Element: **Input**

Design system component

```
function Input({ label, id, ...props }) {
  return (
    <div>
      <label htmlFor={id}>{label}</label>
      <input id={id} {...props} />
    </div>
  );
}
```

Consuming in application

```
const App = () => {
  return <div>
    <Input id="email" label="Email Address" />
    <Input id="password" label="Password" type="password" />
 </div>
}
```

Label always connected - Screen readers work automatically - Keyboard-friendly by default

Synechron

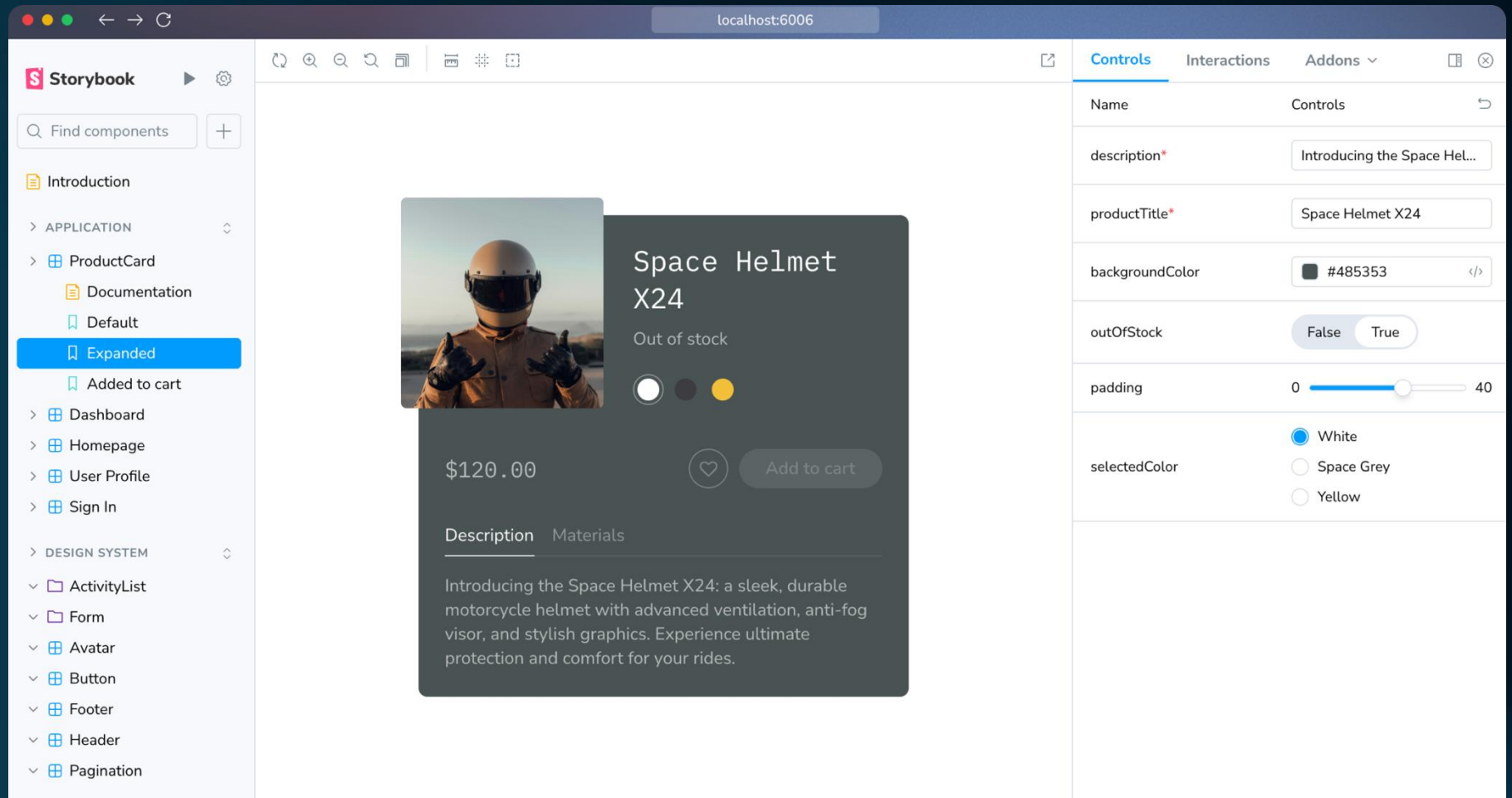
Synechron

Introduction to Storybook

Introducing **Storybook**

Storybook – The Living Catalog

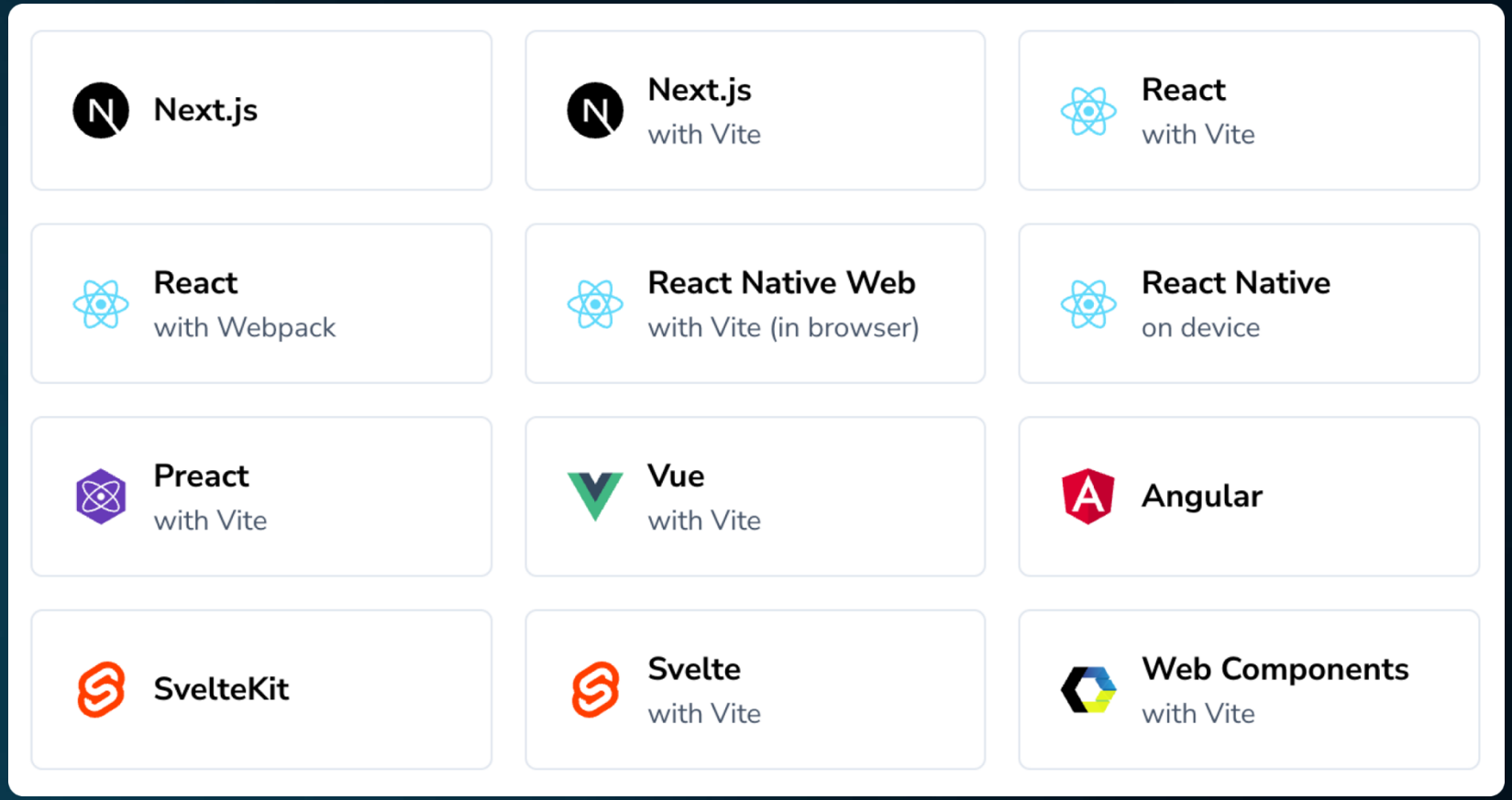
Storybook serves as the single source of truth for all UI components. It allows designers, developers, and stakeholders to explore, test, and document components interactively



Synechron

Introducing **Storybook**

Supported **Frameworks**

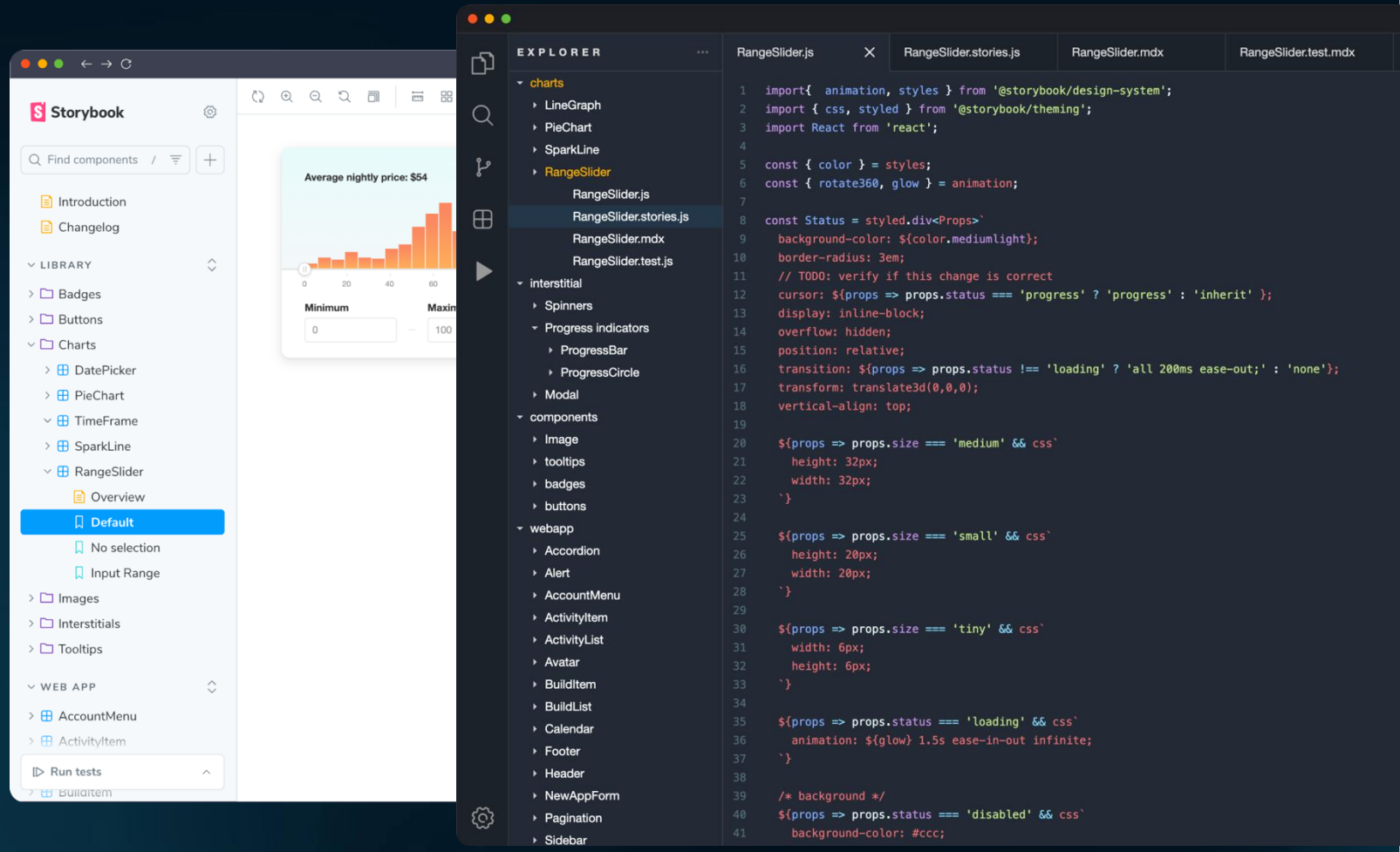


Synechron

Introducing **Storybook**

Main **Features**

- **Stories**
- **Docs**
- **Testing**
- **Sharing**

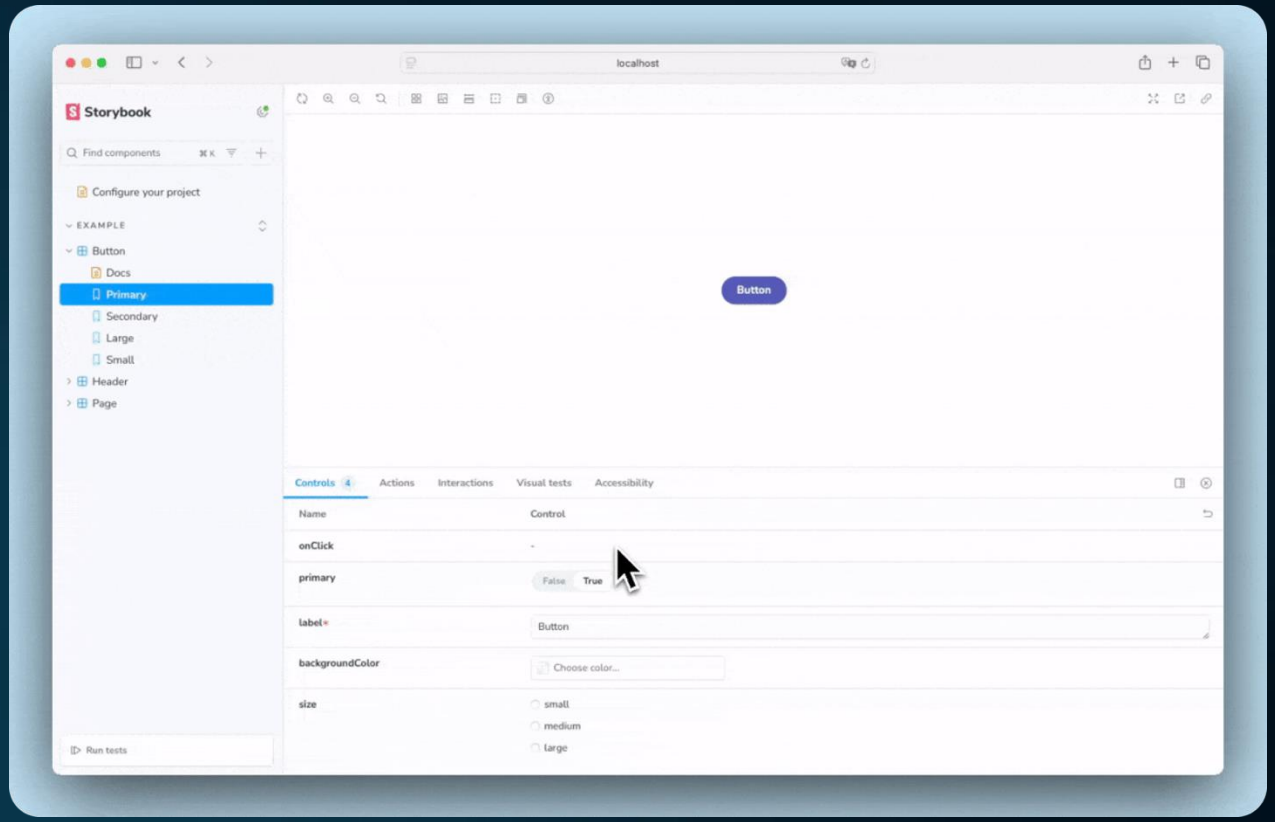


Synechron

Introducing **Storybook**

Stories

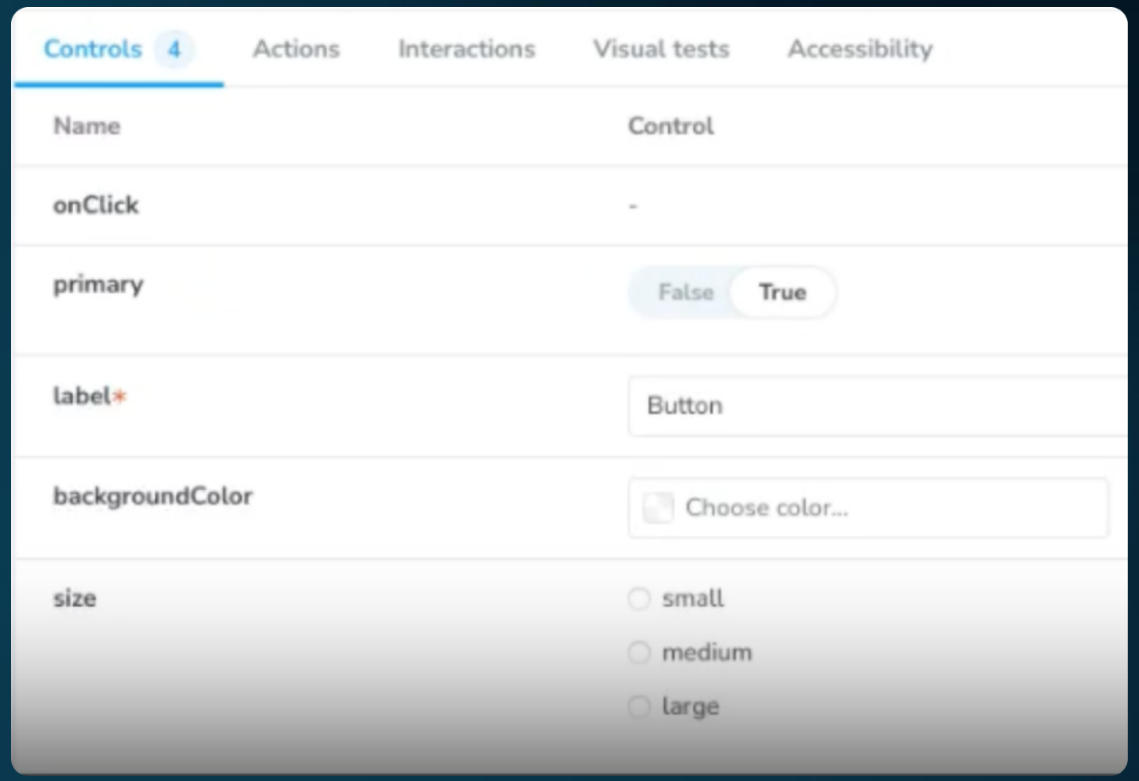
A story is one visual example of a component. We create multiple stories to show all possible states of that component.

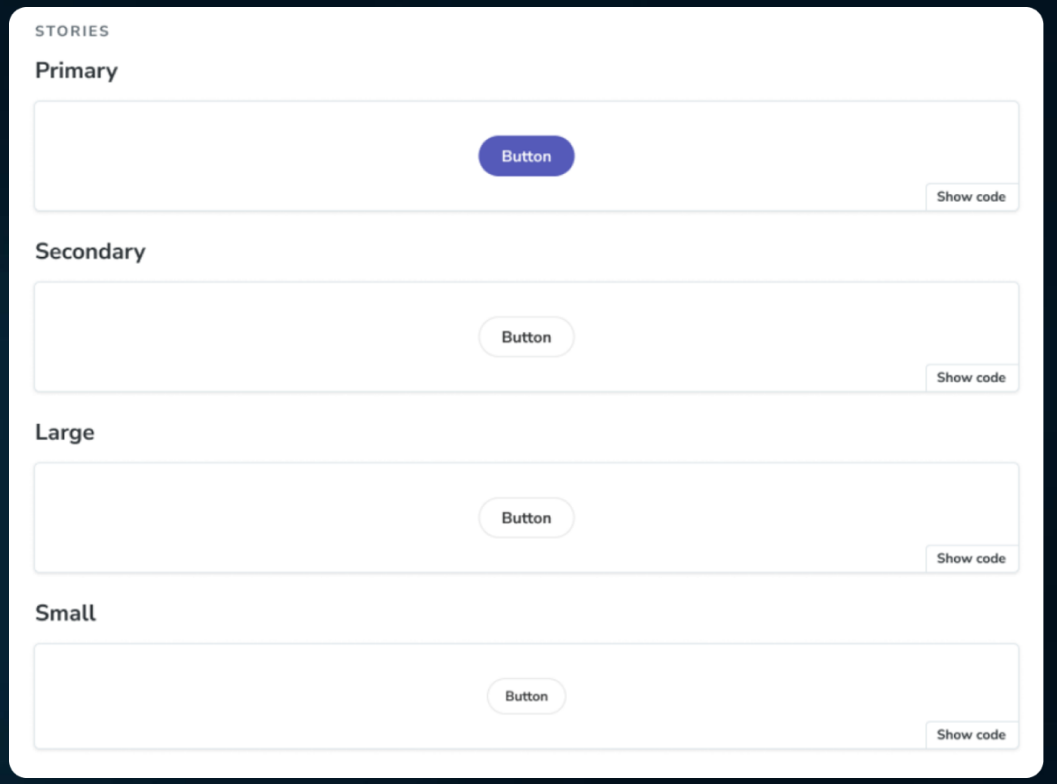


Synechron

Introducing **Storybook**

Stories





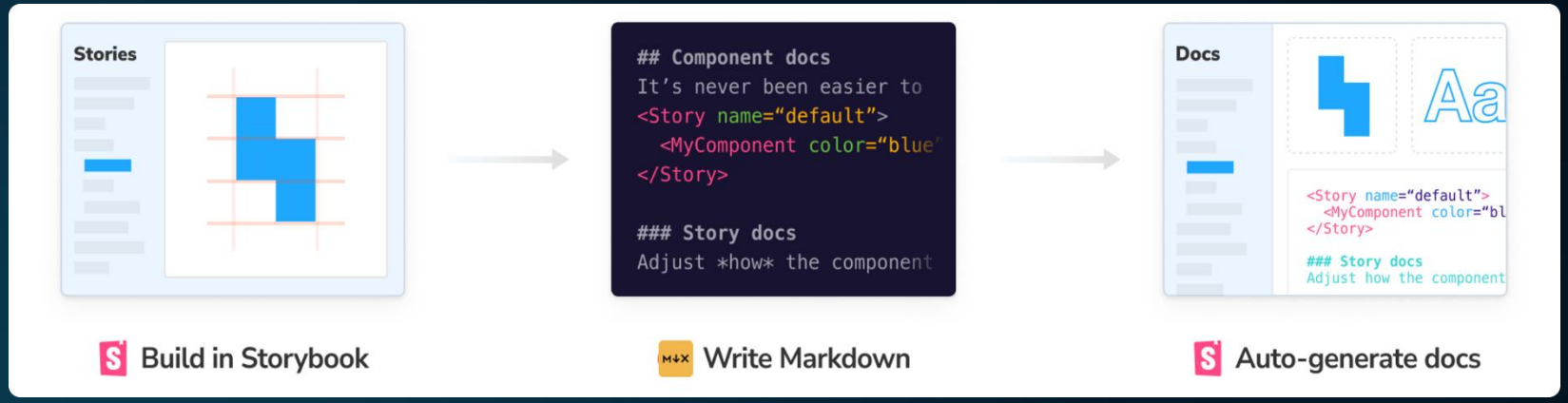
Through Storybook's control panel, change component state to stress test and find edge cases.

Synechron

Introducing **Storybook**

Docs

Storybook turns components and stories into live documentation. As you build stories, your documentation grows with it.



Synechron

Introducing **Storybook**

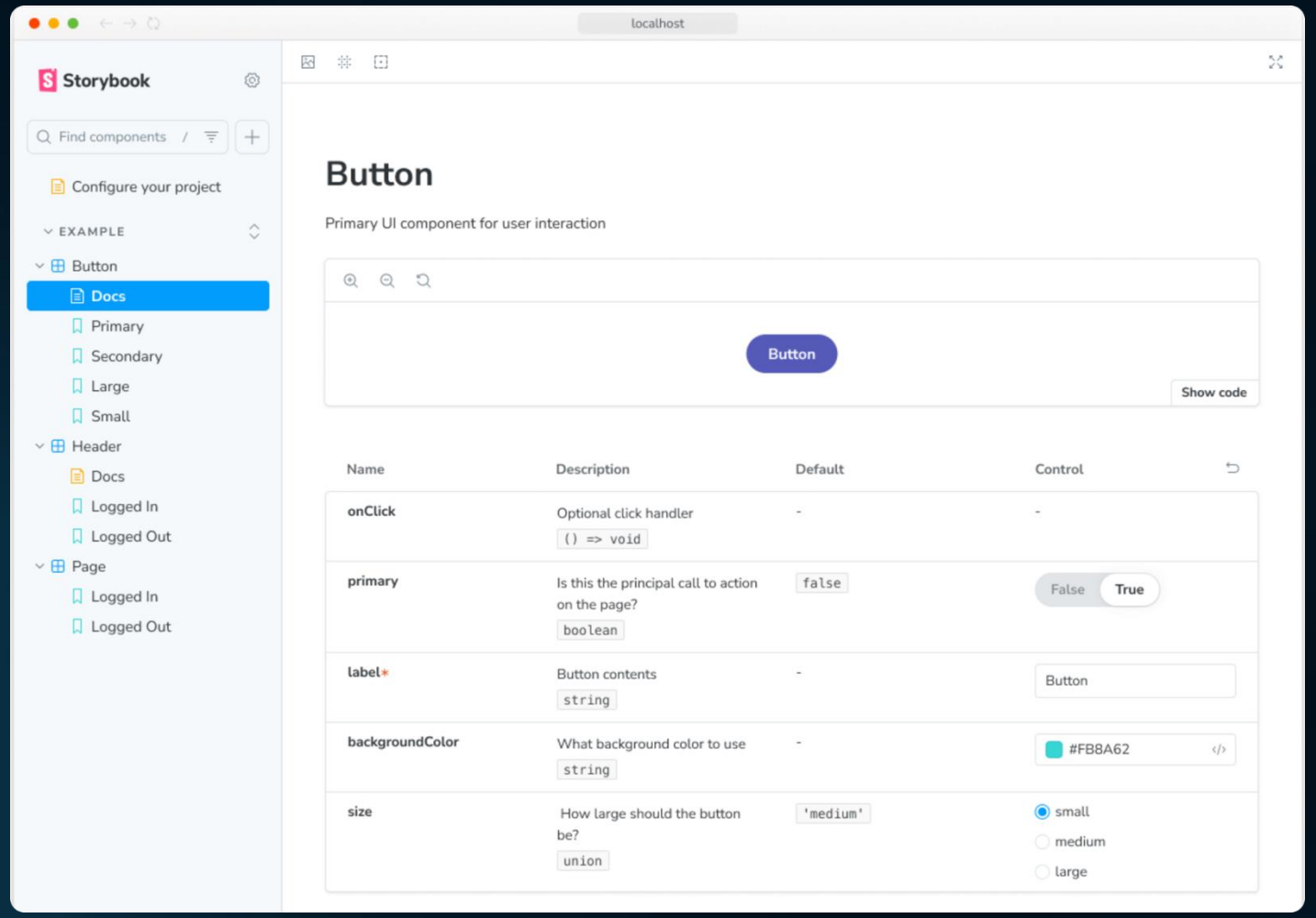
Docs

Auto docs

Storybook infers the relevant metadata (e.g., args, argTypes, parameters) and automatically generates a documentation page.

MDX docs

MDX files mix Markdown and Javascript/JSX to create rich interactive documentation.



Synechron

Introducing **Storybook**

Docs

Doc blocks

Storybook offers several doc blocks to help document your components and other aspects of your project.

Code panel

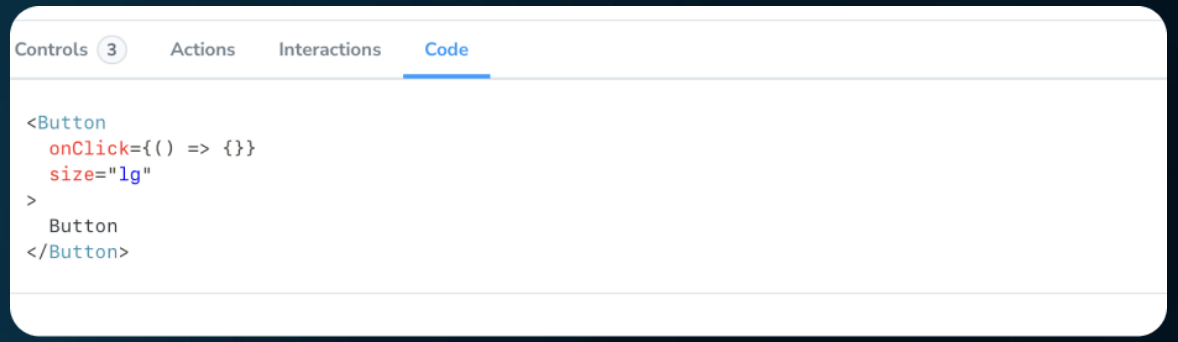
The Code panel renders a story's source code when viewing that story in the canvas.

Available blocks

controls | description | ArgTypes | Markdown | Unstyled

Subtitle | Story | Primary | Meta | Title | IconGallery

Source | Stories | ColorPalette | Canvas | Typeset



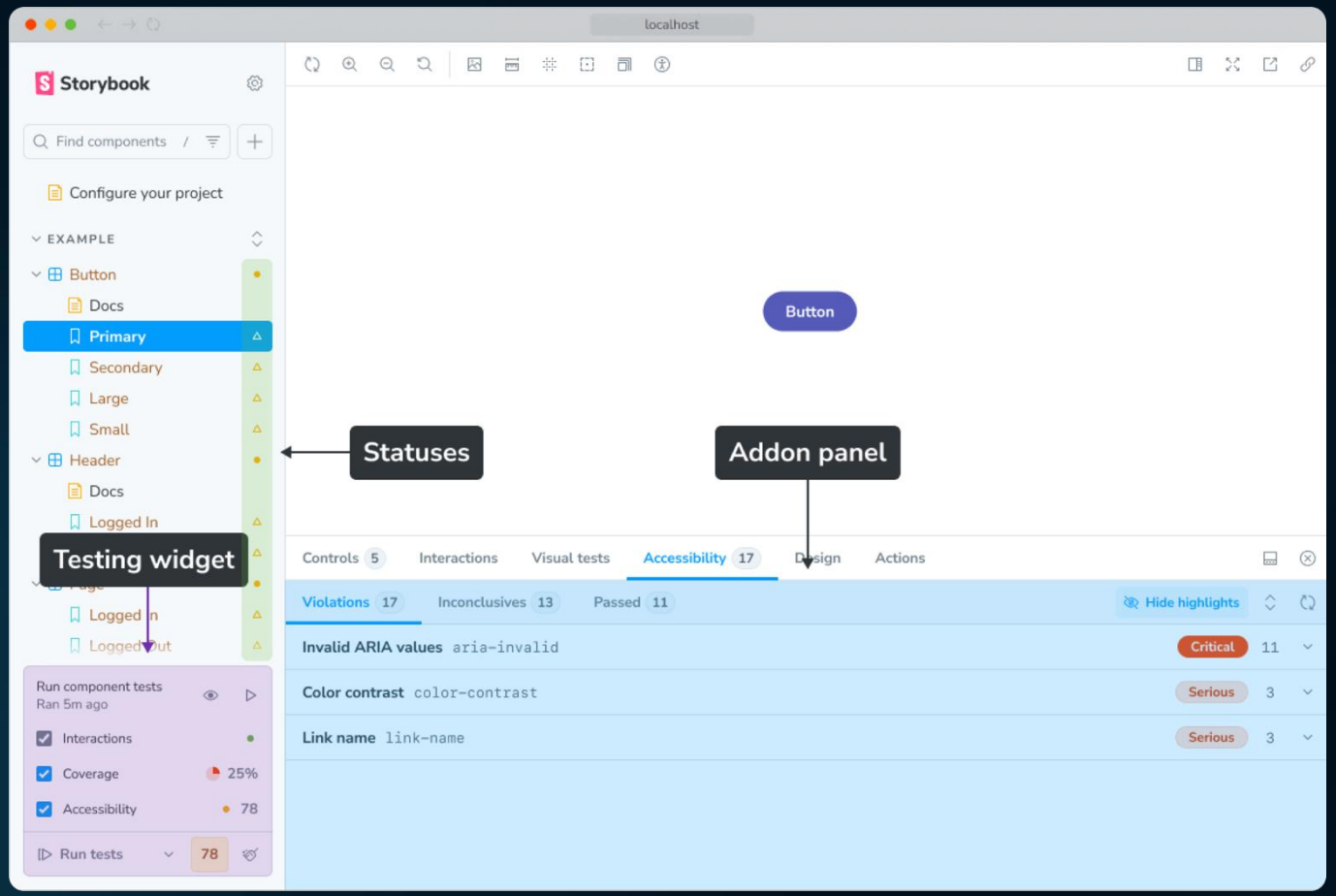
Synechron

Introducing **Storybook**

Testing

Stories double as test cases for UI components

Build, test, and validate components instantly in Storybook.



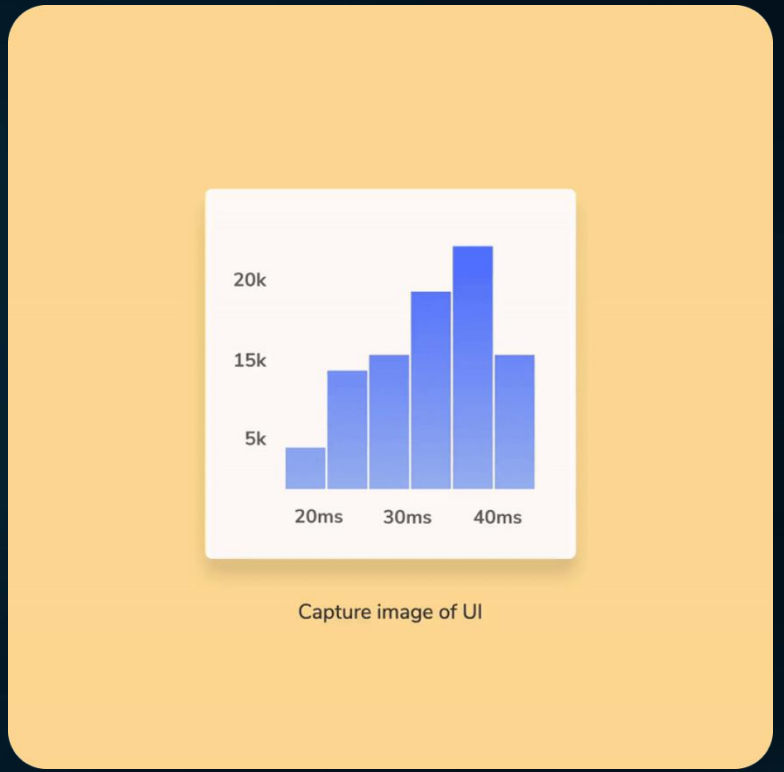
Synechron

Introducing **Storybook**

Testing

Spot test
Stories are tests you can debug in dev and QA

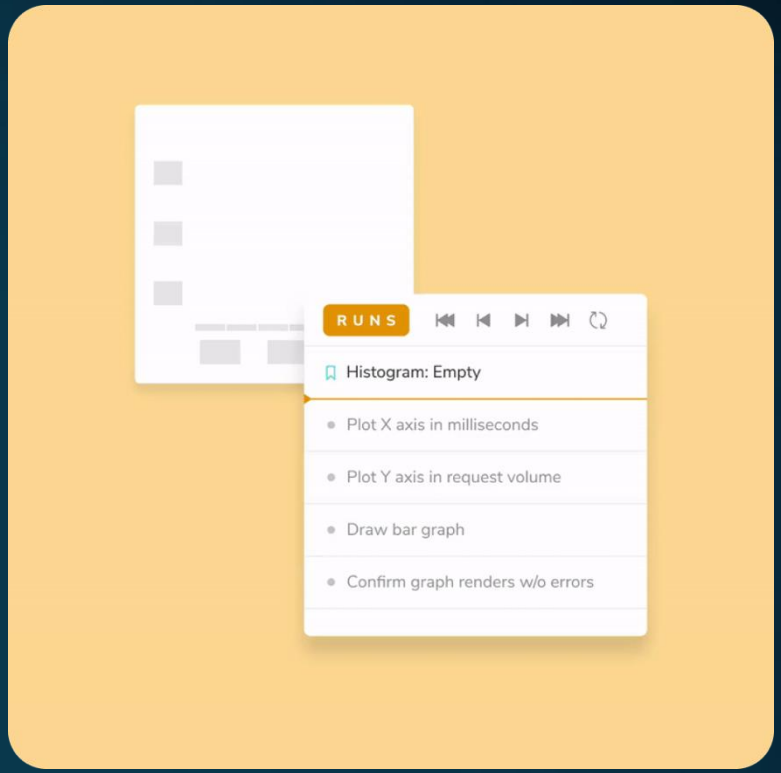
Visual test appearance
Pinpoint UI changes down to the pixel.



Synechron

Testing

Interaction test behavior
Simulate user behavior and assert in the browser.

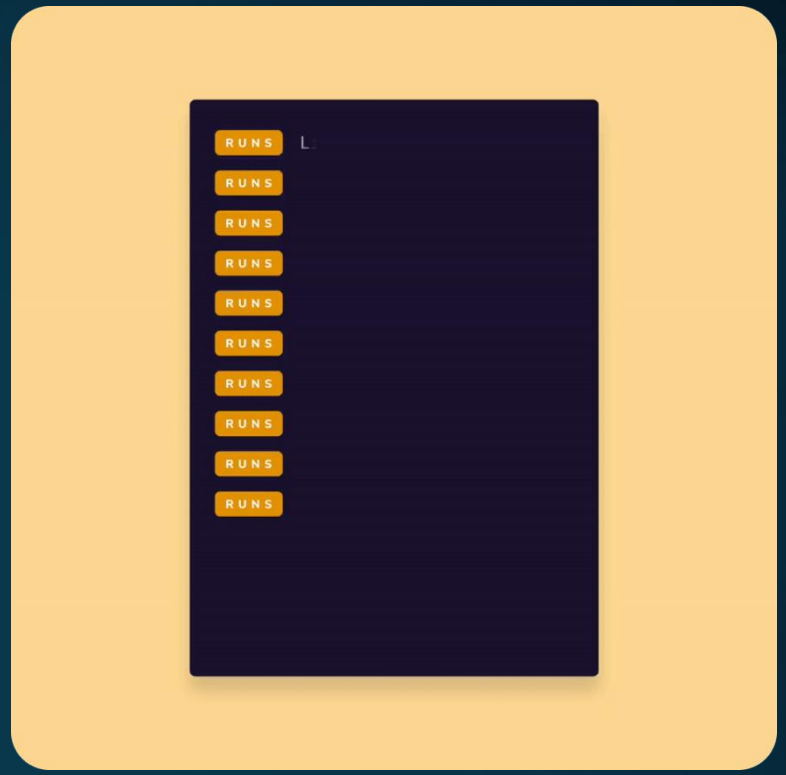


Accessibility tests
Check stories for WCAG and ARIA issues.

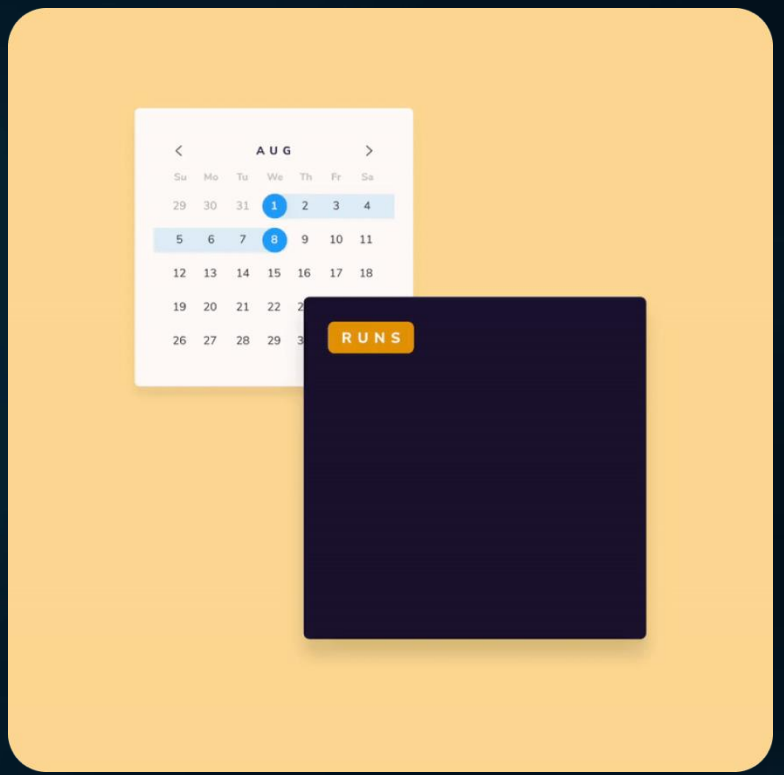
Introducing **Storybook**

Testing

Coverage Reports
Track how much of your frontend code is tested.



Snapshot test markup
Detect regressions in DOM markup.



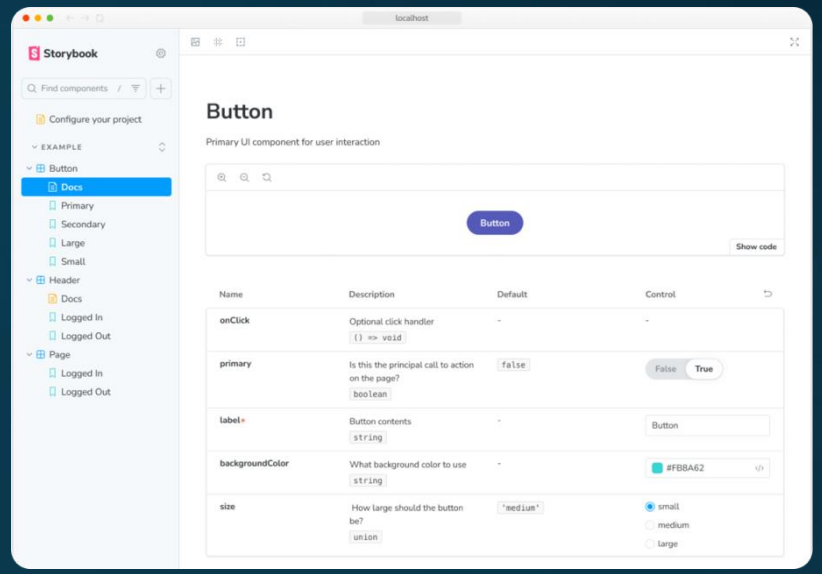
Synechron

Introducing **Storybook**

Sharing

Storybook helps us share our UI library easily across teams, tools, and applications.

Publish as static web app



Embed as iframe

```
<iframe
  src="https://your-storybook.com/?path=/story/button"
  width="800"
  height="260"
></iframe>
```

Design integrations

Figma

Zeplin

Zero height

UX Pin

In vision DSM

Adobe XD

Synechron

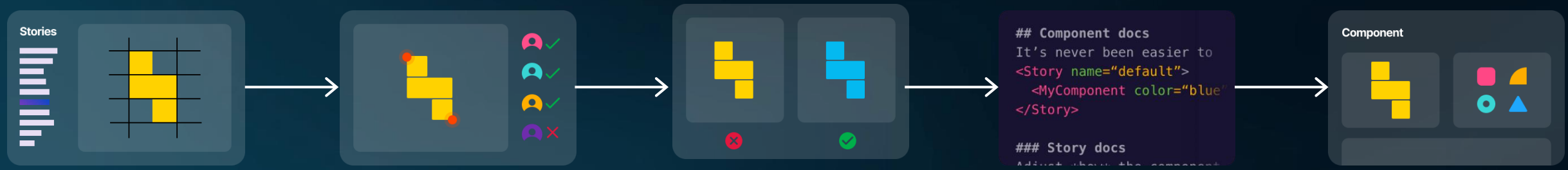
Synechron

Collaborate with Designers and Developers

Collaborate with Designers & **Developers**

Developer **Workflow**

Frontend Workflow



Build

- Review Approved components
- Build UI in isolation
- Lint and format
- Save UI states as stories

Review

- Auto-publish Storybook
- Review with Design team
- Get feedback, resolve issue

Test

- Visual test for UI bug
- Unit test for functional bugs
- Accept/Deny changes

Document

- Auto-generate docs
- Share with stakeholders

Distribute

- Package and version
- Distribute with NPM
- All channels pull updates

Synechron

Synechron

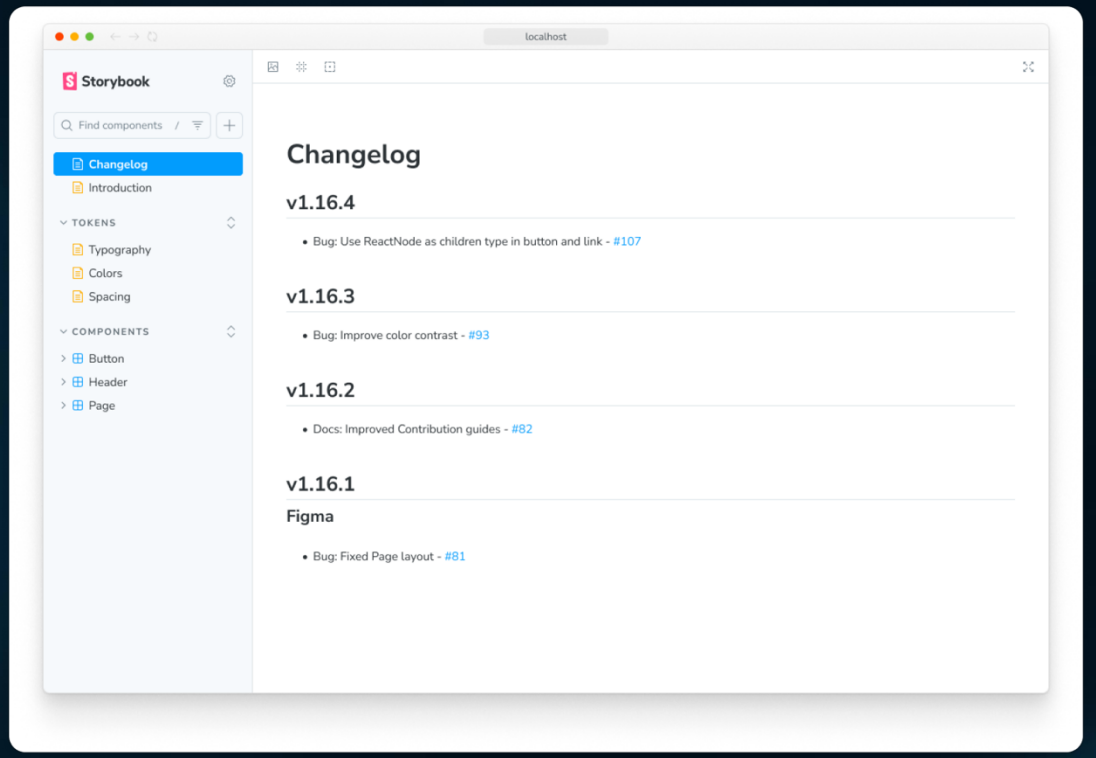
Consuming UI Library in App

Consuming UI Library

Versioning and updates

Versioning makes updates safe and predictable. You always know when a change is breaking and when it’s not.

- Semantic versioning (major.minor.patch)
- Clear release notes for every update
- Backward compatibility by default



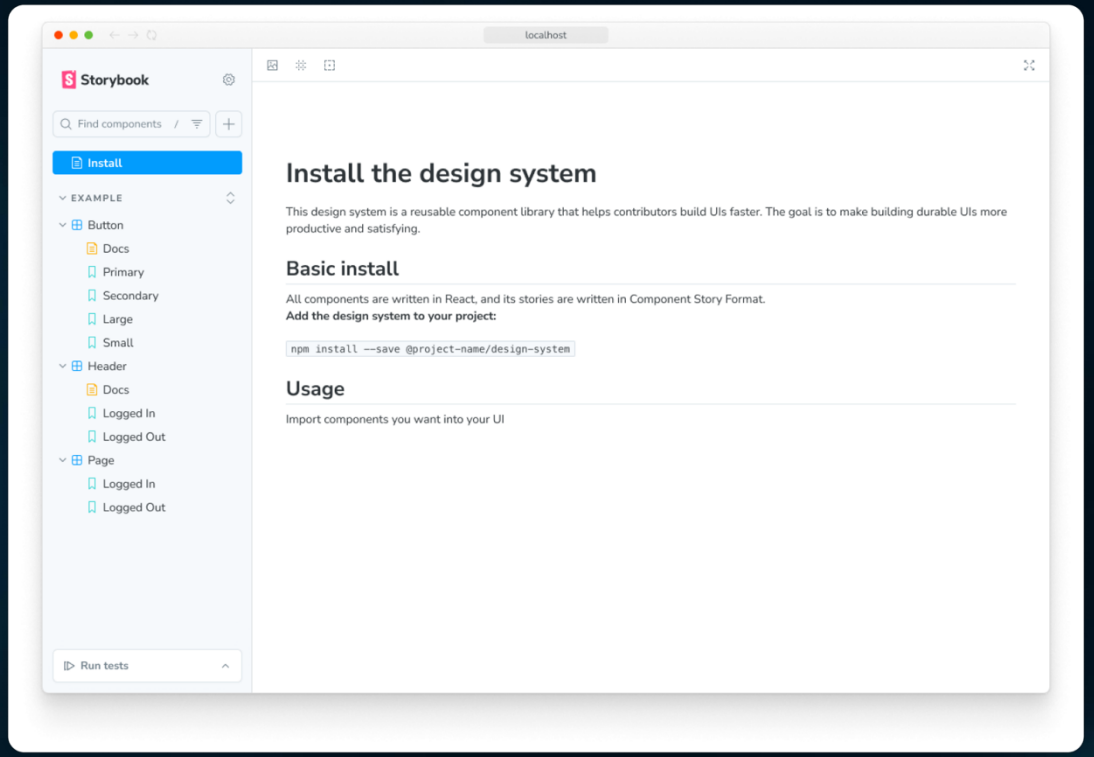
Synechron

Consuming UI Library

Simple **Installation**

The design system is installed just like any standard npm package. Teams can start using it immediately without extra setup.

- One fix benefits all applications
- Faster issue resolution
- Less duplicated effort across teams
- Higher confidence in shared UI



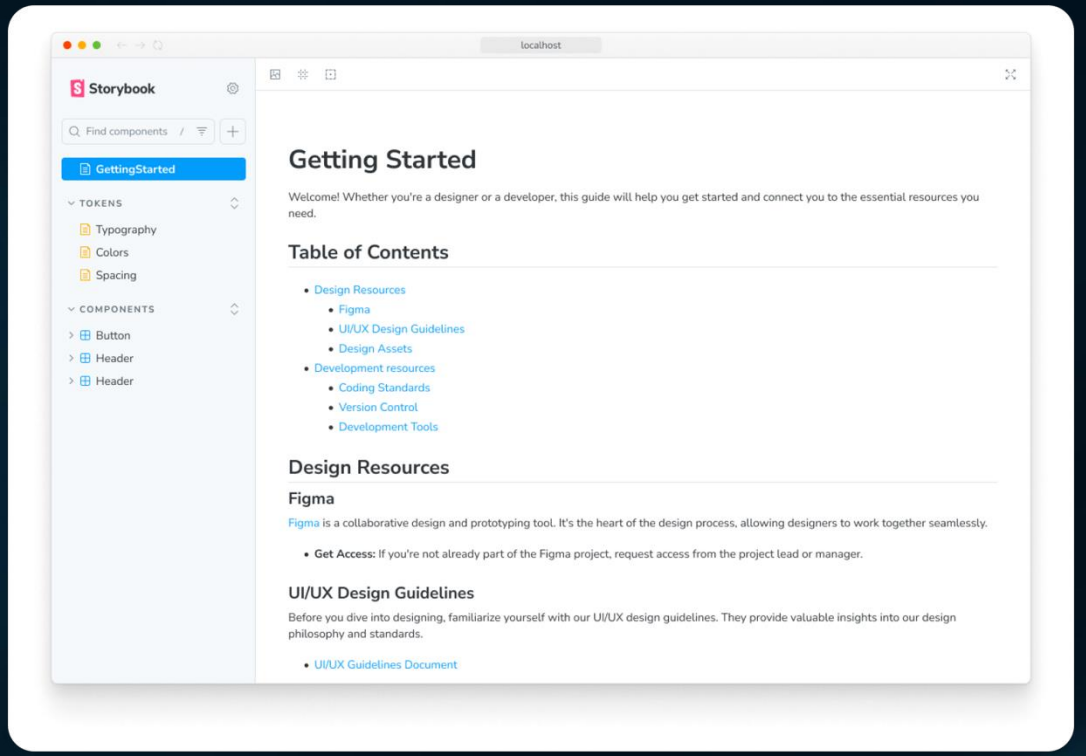
Synechron

Consuming UI Library

Onboarding and learning path

Teams learn faster, build independently, and rely less on hand-holding.

- Faster onboarding with clear examples
- Less reliance on senior engineers
- Documentation that supports self-service learning



Synechron

Consuming UI Library

Ease of Use & Composability

Teams can quickly assemble screens by combining small, reusable pieces.

- No custom styling needed
- Consistent behavior
- No layout hacks
- No repeated logic

Usage

```
<Card>
  <Text variant="heading">Profile</Text>
  <Input label="Email" />
  <Button>Save</Button>
</Card>

```

Small, consistent components combine to create real application screens quickly.

Synechron

Isolation from Business logic

UI components focus only on layout, styling, and accessibility. Business logic stays in the application, keeping both clean and easy to maintain.

- Components remain reusable across different apps and flows
- Changes in business logic do not require UI changes
- Easier testing and safer refactoring over time

Usage

```
function App() {
  function handleSave() {
    dispatch(saveUser());
  }

  return (
    <Button title="Save" onClick={handleSave} />
  );
}
```

Synechron

Evaluation and maintenance

Evaluation and maintenance

Design Audit

A design audit is like a magnifying glass for your brand's visual elements. It takes a hard look at everything your company puts out there, ensuring your brand's image, vibe, and messaging are rock-solid and consistent across the board.

Why should you conduct a design audit?

Complete overview

A design audit identifies inconsistencies, pinpoints channels that don't follow brand guidelines, and helps improve future branding enhancement.

Improved user experience

By identifying UI and UX issues, a design audit paves the way for smoother, more enjoyable user interactions with your products.

Unbiased perspective

Outsourcing the audit to an impartial third party provides an objective assessment, free from internal influences or biases.

Consistency

Consistent visual and textual communication makes a brand recognizable, reliable and trustworthy. It’s one of the foundations of successful branding.

Synechron

Evaluation and maintenance

Key Design Audit Checkpoints

These are a set of standardized evaluation criteria used to review UI designs produced by teams and ensure they align with the design system, brand guideline, and product quality expectations.

1

Design Tokens Consistency

Check out typography, colors, spacing, and elevation are applied using design tokens rather than hardcoded values.

2

Component Usage

Ensures components are configured with the correct variants and behaviors based on their intended use, align with Figma libraries & Storybook.

3

Layout, Grid & Spacing

Reviews structural consistency, alignment, spacing, and adherence to grid systems across the UI journey.

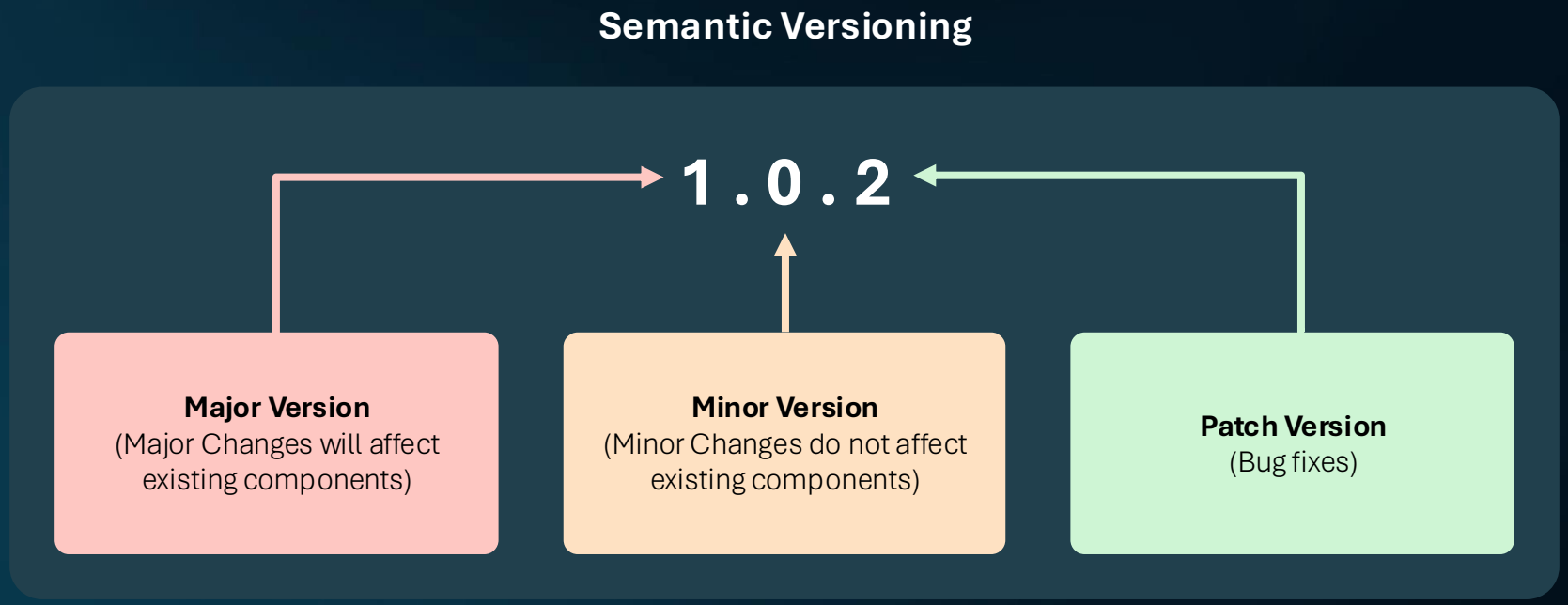
4

Content & UX Consistency

Checks consistency in copy, terminology, CTA hierarchy, and UX patterns across all screens in the journey.

Synechron

Versioning and Change Management



Semantic Versioning helps teams understand how risky an update is, just by looking at the version number.

Versioning and Change Management

Major changes

An enhancement towards an already existing design component that fundamentally changes every technical aspect of how it should work (backwards incompatible). Those changes being, but not limited to:

- Entirely different API call response
- Adjusting existing component's parameters/properties.
- Adjusting design tokens (values, deletions, etc)
- Deleting components
- Renaming / moving components

Minor changes

Newly added design elements / variants / properties towards an already existing design component that maintains the core fundamentals of the original component (backwards compatible). These new elements include:

- Adding design tokens (Colours, fonts swapping, spacings, etc...)
- Adding atomic component (Button, checkboxes, icons, etc...)
- Adding new molecule (combination of atoms) from new or existing atoms
- Extension of existing elements (new button type, new input type, new properties, etc...)

Bug fix

No impact on UI functionality, does not require rigorous testing including:

- Fixing responsiveness issue
- Fixing component properties conflict

Evaluation and maintenance

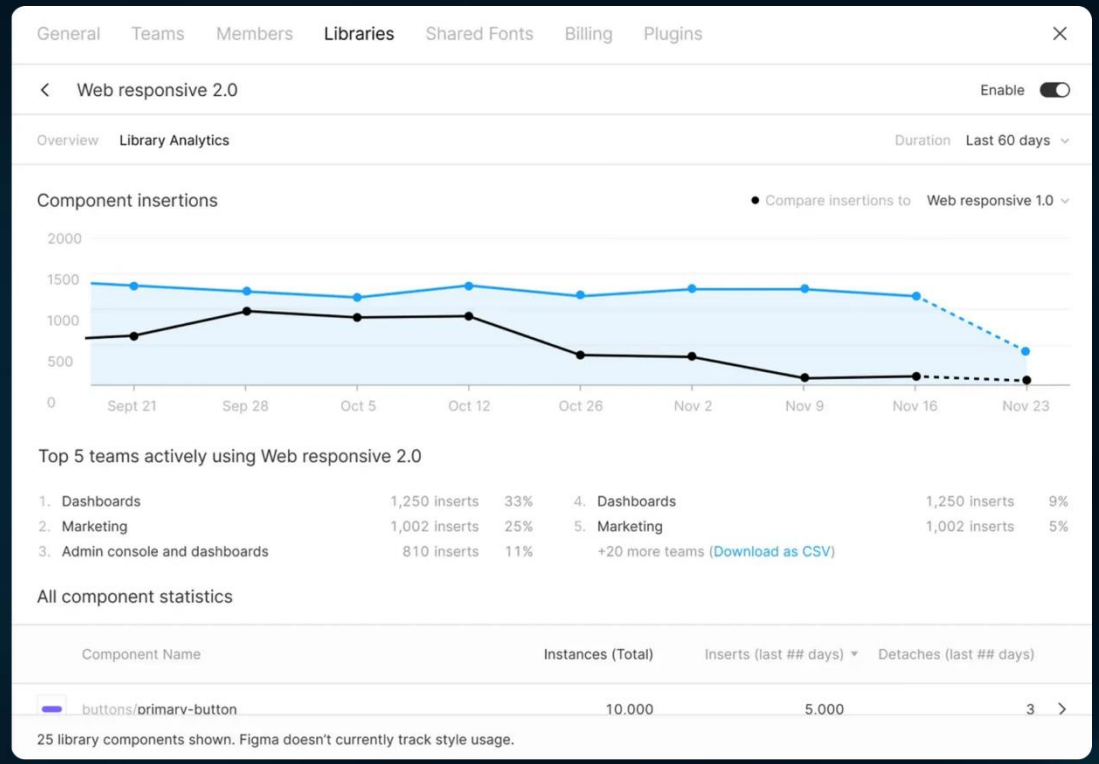
Measuring **adoption** in design

Using data-driven design to improve outcomes

Figma Library Analytics

Understanding how users interact with UI components is crucial for improving the overall user experience and ensuring that the design system serves its intended purpose. Getting insights in the design phase makes it easier to decide where to steer the wheel in the future.

Figma offers Library Analytics where you can view detailed analytics on how members are using components. Styles aren't included in library analytics.



Synechron

Evaluation and maintenance

Warning signs

When you spot a downward trend for these three metrics, slow down, reflect, and react.

Slow or low adoption

When you notice that there is low adoption of a specific UI component, you have to talk to your team.

- Do people know how to use our design system?
- How are our current users satisfied with our UI components? What is missing?
- How can we help them implement the design system in their workflow?
- When the system gets updated, where do people get notified?

Many detached components

If you notice that people detach components too much, it is time to update the component or change it.

A lot of bugs reported

When you start getting a lot of bugs, you need more stability. Probably you didn't test your components enough, or some other factors changed. No matter what, react fast.

Synechron

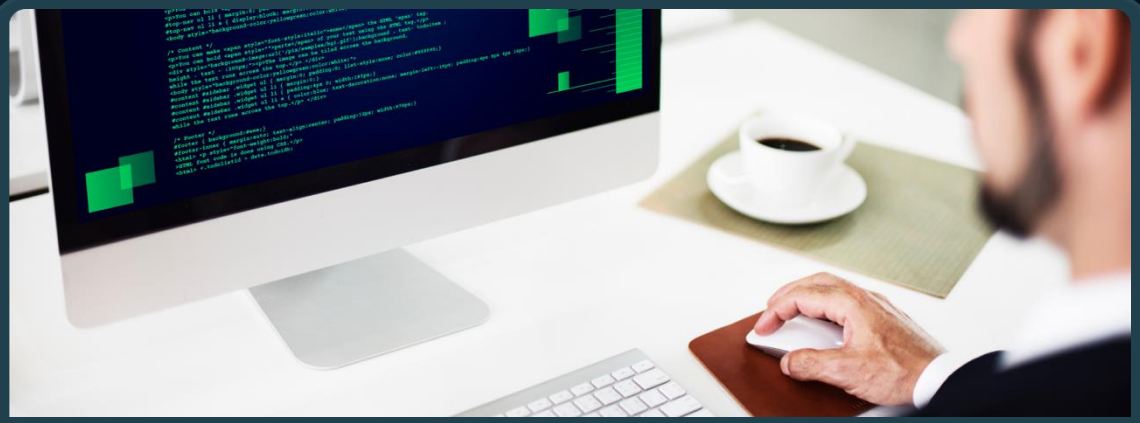
Evaluation and maintenance

Qualitative data

Besides tracking and comparing, you should also understand the subjective side of the design system. It’s time to talk and listen.

Feedback

Feedback is important to enhance your components and increase adoption rate. Constantly testing, iterate, and evolving is the key to maintain a good design system that everybody loves. Conduct 1on1 interviews, focus group discussion, or simply even online survey could help uncover hidden pain points on your design system.



Code Reviews

Conduct code reviews of design system components to ensure they are consistent with the design system's guidelines and best practices. When you analyze all the data, you will better understand how well your design system (and other parts of the organization) is performing from different perspectives.

Synechron

Synechron

The future of Design Systems

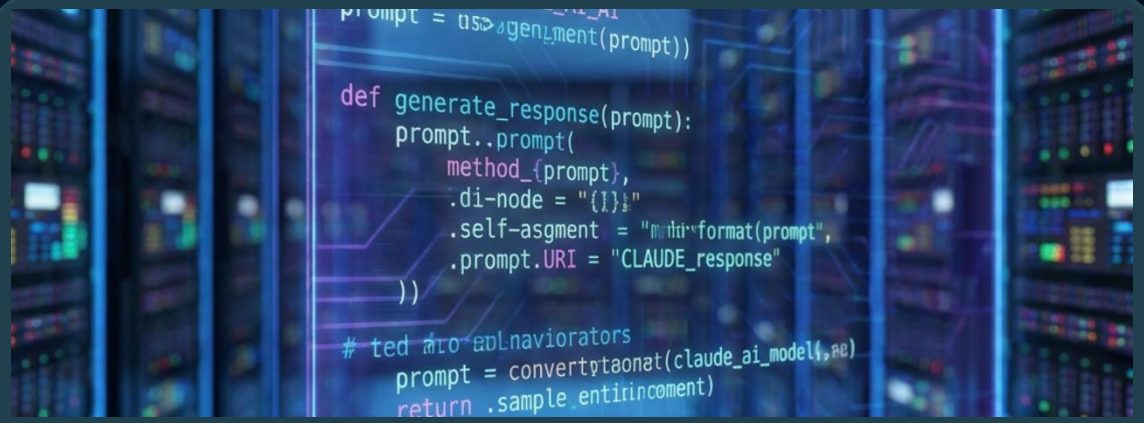
Future Design Systems

AI and **MCP Servers**

Future iterations and how AI plays a part in Design Systems and Development

Figma MCP Server

Figma MCP (Model Context Protocol) server connects Figma to AI systems, allowing models to securely access and understand live design data such as components, tokens, layouts, and documentation. By exposing Figma files through the MCP standard, teams can enable AI tools to interpret design systems directly.



AI – Claude Code

Claude Code is a powerful bridge between design and engineering by translating design specifications into production-ready code, generating components from design tokens, documenting libraries, and enforcing consistency across repositories.

Synechron

Synechron

Thank you