Cours transverse

Comprendre Figma

Hug × Hug

Une application réussie commence rarement dans le code. Elle commence sur un écran où l'on peut se tromper vite, à plusieurs, et sans que cela coûte cher.

Ce cours vous apprend à concevoir des interfaces et des prototypes avec Figma. Surtout, il vous apprend à travailler proprement : avec une charte graphique construite une seule fois, puis réutilisée partout.

- ✓ Passer du croquis au prototype cliquable
- ✓ Créer des mises en page qui s'adaptent
- ✓ Construire des composants réutilisables
- ✓ Comprendre pourquoi Figma s'est imposé
- ✓ Bannir les valeurs codées en dur
- ✓ Transmettre un fichier clair aux développeurs

Le plan, dans l'ordre de lecture

01 Pourquoi maquetter avant de coder ?
02 Le cimetière des outils
03 Pourquoi Figma ?
04 Premiers pas dans le fichier
05 L'auto layout
06 Le piège du codé en dur
07 Les variables
08 Texte, espacements et styles
09 Les composants
10 Prototyper
11 Du fichier au code
12 L'essentiel à retenir

Chapitre 1

Pourquoi maquetter avant de coder ?

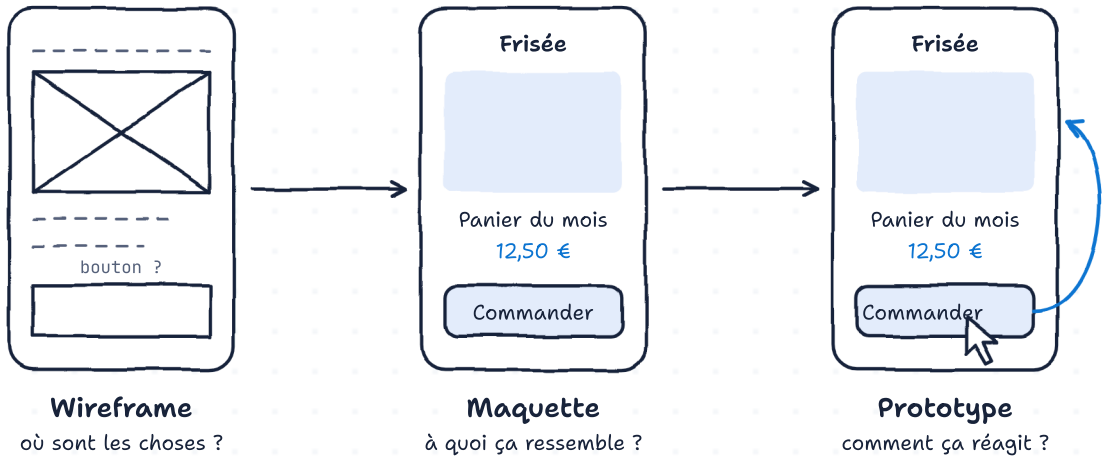
Imaginez que vous fassiez construire une maison. Personne ne coulerait les fondations sans plan, pour découvrir ensuite que la cuisine est trop petite. Sur un plan, déplacer un mur prend deux minutes. Une fois le mur construit, c'est une tout autre facture.

Une application fonctionne de la même façon. Modifier un écran dans le code demande du temps de développement, des tests, parfois une nouvelle mise en ligne. Modifier ce même écran sur une maquette prend quelques instants.

Maquetter sert donc à poser les bonnes questions au moment où elles coûtent le moins cher. Le client comprend-il ce qu'on lui propose ? L'utilisateur trouve-t-il le bouton ? Le parcours d'achat tient-il en trois écrans ?

Trois niveaux de précision

On ne dessine pas tout de suite un écran définitif. On progresse par étapes, et chaque étape répond à une question différente.



Le wireframe ignore volontairement les couleurs, pour que la discussion porte sur l'organisation de l'écran.

Le **wireframe**, ou maquette fil de fer, ressemble à un croquis. La **maquette** montre l'apparence finale. Le **prototype** relie les écrans entre eux : on clique, ça réagit, et on a l'impression d'utiliser la vraie application alors qu'aucune ligne de code n'existe.

Notre fil rouge

Tout au long du cours, nous allons concevoir **Frisée**, une application fictive de paniers de légumes locaux. Alex s'occupe du design, Sam du développement. Si vous avez suivi le cours sur Git, vous les connaissez déjà.

Point de vigilance

Une belle maquette peut donner l'illusion que le projet est presque terminé. Prévenez toujours votre client : ce qu'il voit est une image animée, et tout reste à développer.

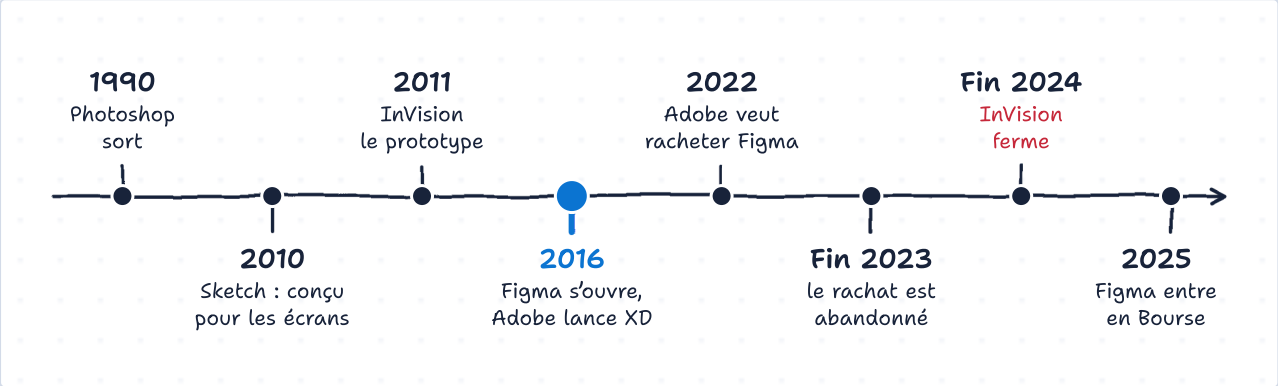
Pour produire ces maquettes, les professionnels ont changé plusieurs fois d'outil. Cette histoire, pleine de disparitions, a beaucoup à vous apprendre.

Chapitre 2

Le cimetière des outils

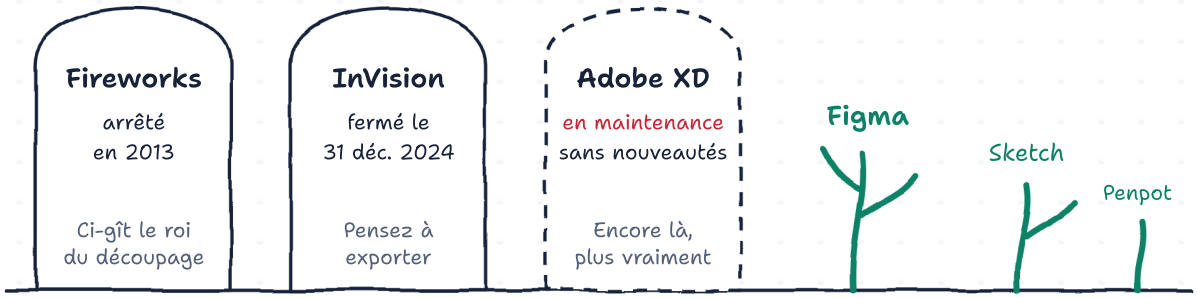
Au début du web, on dessinait les sites dans **Photoshop**. Le logiciel était pensé pour retoucher des photos, et chaque écran devenait un énorme fichier d'images. Changer la couleur d'un bouton répété sur vingt pages voulait dire ouvrir vingt fichiers. Vous commencez à voir venir le sujet de ce cours.

Des outils spécialisés sont alors apparus. Certains ont régné quelques années, puis ont disparu.



En 2010, **Sketch** change les habitudes sur Mac : un logiciel léger, entièrement conçu pour dessiner des écrans. Mais il produit des fichiers qu'il faut s'échanger. Pour montrer un parcours au client, on envoie ensuite les images dans **InVision**, qui permet de les rendre cliquables. Et pour renseigner les développeurs, on ajoute parfois un troisième outil.

Adobe réagit en 2016 avec **XD**, rapide et bien intégré à ses autres logiciels. Il fera longtemps figure de concurrent sérieux. Puis, en 2022, Adobe annonce vouloir racheter Figma pour environ 20 milliards de dollars. Les autorités de la concurrence s'inquiètent, et le projet est abandonné fin 2023, avec une indemnité d'un milliard de dollars versée à Figma.



Les tombes sont réelles, les épitaphes sont de nous.

Où en est-on aujourd'hui ?

Outil	État actuel	À savoir
Fireworks	Arrêté depuis 2013	Seuls de très vieux fichiers subsistent
InVision	Fermé le 31 décembre 2024	Documents supprimés après la fermeture
Adobe XD	En maintenance	Corrections seulement, plus aucune nouvelle fonction. N'investissez pas dedans
Sketch	Toujours actif	Édition réservée au Mac
Penpot	Actif, open source	Peut s'installer sur ses propres serveurs
Framer	Actif	Orienté publication de sites web
Figma	Référence du marché	Le sujet de ce cours

Conseil de pro

Le cas d'InVision est une leçon utile. Quand un outil en ligne ferme, vos documents peuvent partir avec lui. Exportez régulièrement les fichiers importants, et vérifiez ce que prévoit votre contrat avec le client sur leur conservation.

Si vous récupérez un jour un ancien projet XD, n'en faites pas un drame. Le fichier s'ouvre encore pour les personnes qui ont l'application, et rien ne presse de tout refaire. Mais tout nouveau projet doit partir ailleurs.

Comment choisir un outil qui dure ?

Vous devrez un jour recommander un outil à un client ou à une équipe. Quelques questions simples évitent les mauvaises surprises : le logiciel reçoit-il encore des nouveautés ? Peut-on en sortir ses fichiers dans un format réutilisable ? Combien de professionnels le maîtrisent, et pourra-t-on recruter ? Enfin, que se passe-t-il pour nos données si l'éditeur ferme ?

Gardez aussi en tête la vraie leçon de ce cimetière. Les logiciels passent, les compétences restent. Une personne qui sait structurer une charte, penser en composants et tester un parcours changera d'outil en quelques semaines.

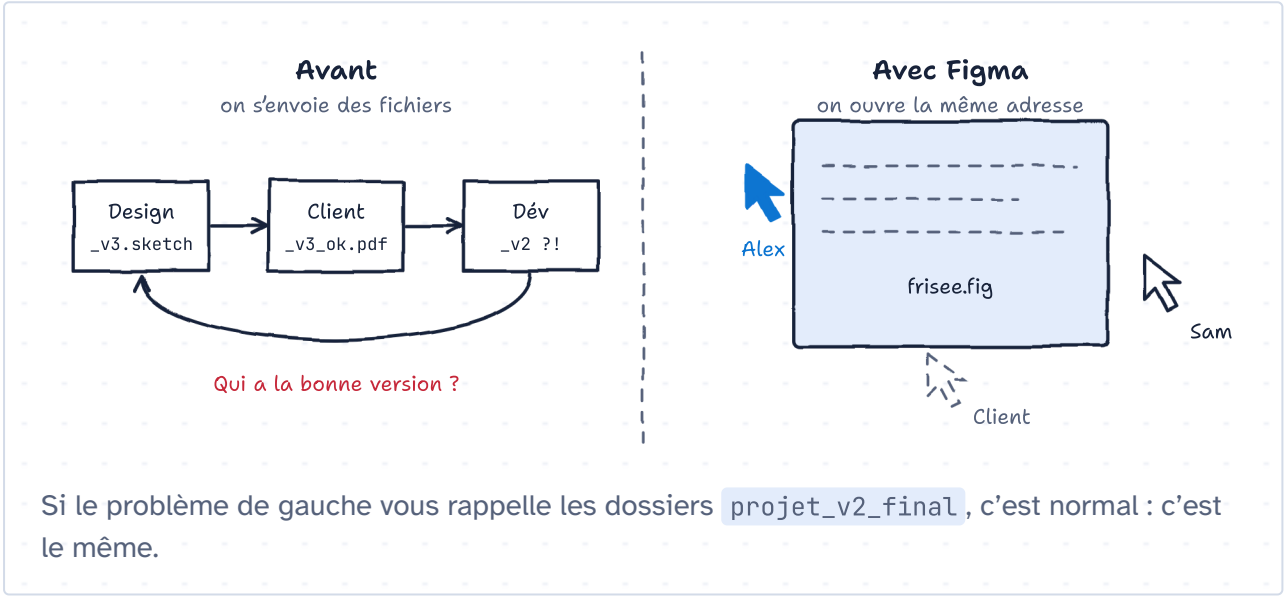
Reste une question : pourquoi est-ce Figma qui a survécu, alors qu'il arrivait après tout le monde ?

Chapitre 3

Pourquoi Figma ?

Figma est créé en 2012 par Dylan Field et Evan Wallace, et s'ouvre au public quelques années plus tard. Son pari semblait alors déraisonnable : faire tourner un outil de dessin vectoriel exigeant directement dans un navigateur.

Ce choix technique a tout changé, car il a transformé le fichier en une simple adresse web.



Si le problème de gauche vous rappelle les dossiers `projet_v2_final`, c'est normal : c'est le même.

Comme dans un document Google Docs, plusieurs personnes travaillent en même temps et voient leurs curseurs respectifs. Rien à installer pour consulter, aucune version à s'envoyer. Le client commente directement sur l'écran concerné, et le développeur lit les dimensions dans le même fichier.

Ce que vous gagnez au quotidien

- **Un seul outil** pour dessiner, prototyper, commenter et transmettre.
- **N'importe quel système** : Windows, Mac, Linux, il suffit d'un navigateur.
- **Un historique des versions** pour revenir en arrière.
- **Une communauté** qui partage modèles et extensions.
- **Un vrai système de variables et de composants**, que nous allons explorer en détail.

Point de vigilance

Figma reste un service en ligne, édité par une entreprise privée. Vos fichiers sont chez un prestataire, l'usage hors connexion est limité, et les conditions tarifaires évoluent. Sur un projet confidentiel, posez la question avant de créer le fichier. Penpot, open source, est une alternative à connaître lorsqu'il faut héberger ses données soi-même.

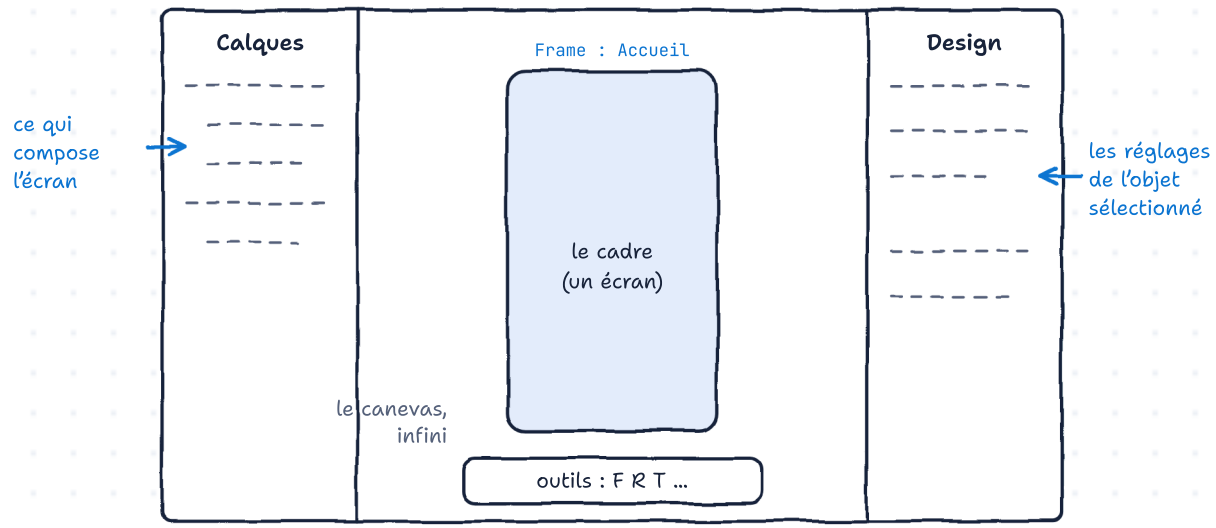
L'offre gratuite suffit pour apprendre. Les variables y sont disponibles, mais certaines fonctions, comme les modes multiples ou les bibliothèques partagées, demandent une offre payante. Si vous êtes en formation, regardez du côté de l'offre Education. Les détails bougent souvent : vérifiez sur la page des tarifs.

Vous êtes convaincu ? Ouvrons un fichier et faisons connaissance avec l'atelier.

Chapitre 4

Premiers pas dans le fichier

Quand on ouvre Figma pour la première fois, l'écran paraît chargé. En réalité, il se résume à quatre zones.



À droite, l'onglet Design règle l'apparence. L'onglet Prototype règle le comportement.

Le cadre, l'objet de base

Sur ce canevas infini, il faut délimiter vos écrans. C'est le rôle du **cadre** (*frame*). Vous appuyez sur `F`, vous choisissez un format de téléphone dans le panneau de droite, et votre première page existe.

Un cadre ressemble à une feuille posée sur une table, à ceci près qu'il peut contenir d'autres cadres. Une carte contient une image et un titre, un écran contient des cartes. C'est exactement la logique des blocs en HTML, et Sam retrouvera ses repères.

Conseil de pro

Nommez vos calques. `Rectangle 47` et `Frame 212` ne disent rien à la personne qui reprendra votre fichier. C'est un peu comme écrire `modif` dans un message de commit.

Les raccourcis à connaître

Défi de la semaine : gardez votre main gauche sur le clavier et essayez de ne pas toucher à la barre d'outils.

Touche	Ce qu'elle fait
`F`	Créer un cadre
`R` et `T`	Dessiner un rectangle, écrire un texte
`Maj + A`	Appliquer l'auto layout à la sélection
`Alt` + glisser	Dupliquer un objet
`Ctrl + /` (`Cmd + /` sur Mac)	Rechercher n'importe quelle commande

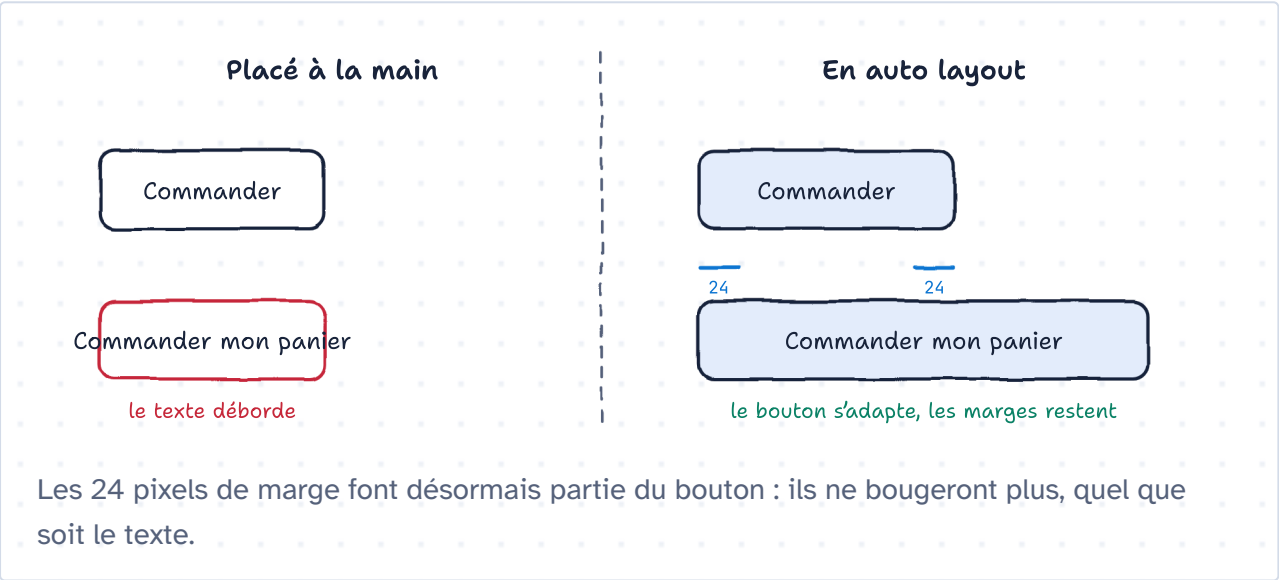
Vous savez poser des objets. Mais si vous placez chaque élément à la main, au pixel près, un simple changement de texte va tout dérégler. Voyons pourquoi, et comment l'éviter.

Chapitre 5

L'auto layout

Alex a dessiné un bouton : un rectangle vert, avec le texte « Commander » centré à la main. Le client demande maintenant « Commander mon panier ». Le texte déborde. Il faut agrandir le rectangle, recentrer le texte, et recommencer pour chaque bouton de l'application.

Un bouton réel, dans une page web, ne se comporte pas ainsi. Il s'adapte à son contenu. Figma permet de retrouver ce comportement avec l'**auto layout** : au lieu de placer des objets, vous décrivez une règle de disposition.



Les 24 pixels de marge font désormais partie du bouton : ils ne bougeront plus, quel que soit le texte.

Les trois réglages à comprendre

Sélectionnez des éléments, appuyez sur `Maj + A`, et Figma crée un conteneur qui les aligne avec un écart et des marges internes. Il reste à dire comment chaque élément gère sa taille.

Réglage	Comportement	Exemple
Hug (épouser)	Prend la taille de son contenu	Un bouton autour de son texte
Fill (remplir)	Occupe la place disponible	Un champ sur toute la largeur
Fixed (fixe)	Garde une taille imposée	Une icône de 24 pixels

Si vous connaissez le CSS, vous avez reconnu Flexbox. Cette ressemblance est précieuse, car ce qu'Alex dessine en auto layout, Sam peut le reproduire presque à l'identique.

Faites le test : sélectionnez votre écran, élargissez-le. Si tout se réorganise proprement, votre travail est solide. Si les éléments restent figés, c'est qu'ils ont été placés à la main.

Point de vigilance

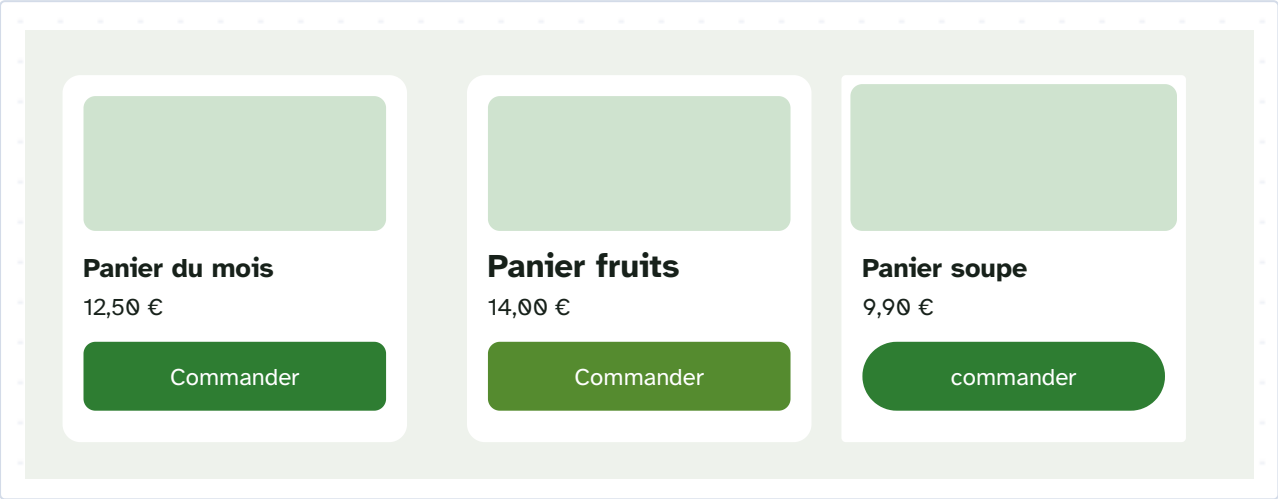
Le réflexe de débutant consiste à dessiner d'abord, puis à ajouter l'auto layout à la fin. Mettez-le en place dès le départ, en partant du plus petit élément vers le plus grand : le bouton, puis la carte, puis l'écran.

Vos éléments s'adaptent désormais à leur contenu. Reste un problème bien plus sournois, celui qui fait la différence entre un fichier amateur et un fichier professionnel.

Chapitre 6

Le piège du codé en dur

Commençons par un jeu. Alex a dessiné trois cartes censées être identiques, un vendredi à 18 heures. Saurez-vous trouver les 7 erreurs ?



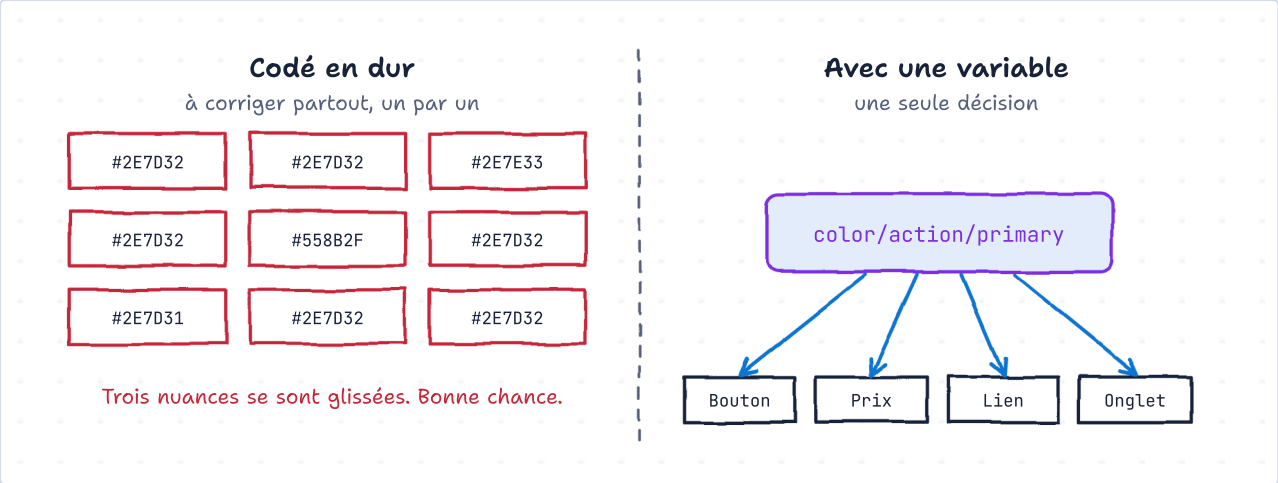
▼ Voir la solution

1. Le bouton du milieu a un vert différent.
2. Le titre du milieu est plus grand.
3. Les coins de la troisième carte sont moins arrondis.
4. Les marges de la troisième carte sont plus serrées autour de l'image.
5. Le bouton de droite a une forme de pilule.
6. Son libellé n'a pas de majuscule.
7. L'écart entre les cartes n'est pas le même à gauche et à droite.

Personne ne fait ces erreurs volontairement. Elles arrivent parce que chaque valeur a été saisie à la main : un vert copié d'un ancien écran, une taille de texte retapée de mémoire, un arrondi réglé à l'œil. On appelle cela des valeurs **codées en dur**.

Le coup de téléphone qui fait mal

L'application Frisée compte quarante écrans. Le client appelle : son nouveau logo est orange, et il faudrait que les boutons et les prix suivent. Dans un fichier codé en dur, Alex doit retrouver un par un tous les objets verts. Et parmi eux, lesquels sont des boutons, et lesquels sont simplement des feuilles de salade ?



La solution tient en une idée : **une décision de design doit exister à un seul endroit**. Tout le reste doit y faire référence. Les développeurs connaissent ce principe depuis longtemps, car personne ne recopie une même valeur à cent endroits dans un programme.

Dans un fichier sain, quand vous sélectionnez un objet, le panneau de droite affiche le nom d'une variable à la place d'un code hexadécimal. Ce sera désormais votre critère de qualité.

L'astuce pour auditer un fichier

Sélectionnez un écran entier et regardez la section *Selection colors* du panneau de droite. Elle liste toutes les couleurs présentes. Chaque code hexadécimal affiché est une valeur qui a échappé à votre charte.

Voyons maintenant comment construire cette source unique, avec l'outil prévu pour cela.

Chapitre 7

Les variables

Une charte graphique est un ensemble de décisions : notre vert, notre titre, nos arrondis. Pendant longtemps, elle vivait dans un PDF que tout le monde oubliait d'ouvrir. Les **variables** la font entrer dans le fichier de travail.

Une variable est une valeur à laquelle on donne un nom, comme une étiquette collée sur un bocal. Vous utilisez le bocal « sel ». Le jour où vous changez de marque de sel, la recette ne change pas.

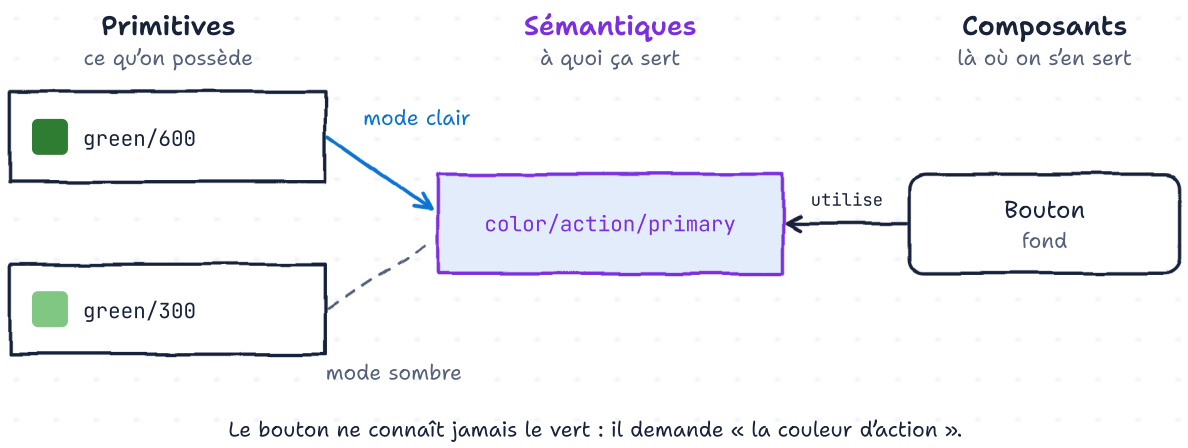
Ce qu'une variable peut contenir

Type	Exemple de valeur	On l'applique à
Couleur	`#2E7D32`	Remplissages, contours, ombres
Nombre	`16`	Marges, arrondis, tailles de texte
Chaîne	`Commander`	Textes, polices, graisses
Booléen	`vrai` ou `faux`	Afficher ou masquer un calque

Figma propose aussi des variables de durée et de courbe, utiles pour les animations. Vous les découvrirez plus tard. On les range dans des **collections**, créées depuis l'icône des réglages du panneau de droite, lorsqu'aucun objet n'est sélectionné.

La bonne architecture : deux étages

Votre premier réflexe sera sans doute de créer une variable `vert` et de l'appliquer à tous les boutons. C'est mieux qu'un code en dur, mais si la marque passe à l'orange, vous aurez une variable `vert` qui contient de l'orange. Les professionnels séparent donc deux questions : *quelles couleurs possédons-nous ?* et *à quoi servent-elles ?*



Le bouton ne connaît jamais le vert : il demande « la couleur d'action ».

Une variable qui pointe vers une autre s'appelle un alias. Changez la primitive, et tout ce qui en dépend suit.

Les **primitives** forment votre nuancier : `green/600`, `grey/900`, `white`. Les **variables sémantiques** décrivent un rôle : `color/action/primary`, `color/text/default`, `color/surface/card`. Chaque sémantique est un **alias**, c'est-à-dire un renvoi vers une primitive.

La règle est simple : **les maquettes n'utilisent que les sémantiques**. Pour la faire respecter, activez l'option *Hide from publishing* sur vos primitives, afin qu'elles n'apparaissent pas dans les choix proposés à l'équipe.

Conseil de pro

La barre oblique crée des dossiers : `color/text/default` se range tout seul. Autre réglage utile, la portée (*scope*) limite une variable à certaines propriétés. Une variable de fond n'a aucune raison d'être proposée pour un texte.

Un clic pour passer en mode sombre

C'est ici que l'architecture révèle sa puissance. Une variable peut stocker plusieurs valeurs, une par **mode**. Ajoutez une colonne « Sombre » à votre collection, et pointez chaque sémantique vers la bonne primitive.

Variable	Clair	Sombre
`color/action/primary`	green/600	green/300
`color/text/default`	grey/900	grey/50
`color/text/muted`	grey/600	grey/300
`color/surface/card`	white	grey/800
`color/surface/page`	grey/50	grey/900

Voici à quoi ressemble la collection de Frisée. Vous remarquerez qu'aucun nom de variable ne contient le mot « vert ».

Sélectionnez ensuite un cadre, et choisissez le mode voulu dans la section *Appearance*. Tout ce qu'il contient bascule, puisque les éléments héritent du mode de leur parent. Le même principe sert à gérer plusieurs marques, ou des textes en plusieurs langues.

La couleur n'est qu'une partie de la charte. Appliquons la même discipline au texte et aux espacements.

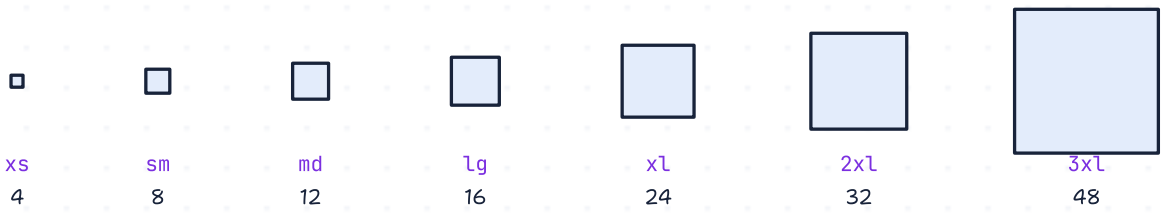
Chapitre 8

Texte, espacements et styles

Dans un fichier amateur, chaque texte est réglé à la main : Inter, 17 pixels, gras, gris foncé. On recommence au texte suivant, et on tape 16 par distraction. Pourtant une application n'a besoin que d'une poignée de niveaux de texte : un grand titre, un sous-titre, le texte courant, une légende.

Les espacements : une échelle, pas des valeurs

Commençons par le plus simple. Pourquoi une marge de 13 pixels ici, et de 15 pixels là ? Pour éviter ces hésitations, on se fixe une échelle, le plus souvent multiple de 4 ou de 8, et on la stocke dans des variables de type nombre.



Dessiné au double de la taille réelle. Avec sept valeurs, on couvre presque tous les besoins d'une interface.

Créez `space/sm`, `space/md`, `radius/card`, `radius/button`, et appliquez-les aux marges de vos auto layouts et à vos arrondis. Le rythme de vos écrans devient régulier, et il faut un seul geste pour tout aérer.

Le texte : les styles entrent en jeu

Un texte réunit plusieurs réglages : police, taille, graisse, interligne. Or une variable ne contient qu'une seule valeur. Figma propose donc un second outil, le **style de texte**, qui rassemble tous ces réglages sous un seul nom. Et la bonne pratique consiste à combiner les deux.



Les variables portent les valeurs, le style les regroupe, le calque n'a plus qu'un nom à connaître.

Sur chaque calque de texte, vous choisissez un style. La couleur, elle, reste liée à une variable sémantique comme `color/text/default`. Vous n'avez plus jamais à taper une taille de police à la main.

L'échelle typographique de Frisée

Voici un jeu complet pour une application mobile. Remarquez que les noms décrivent un rôle dans la page. Le jour où le « titre de page » passe de 28 à 32 pixels, son nom reste juste.

Style	Taille / interligne	Graisse	Usage
`title/page`	28 / 34	Bold	Le titre principal d'un écran
`title/section`	20 / 26	Semi Bold	Un bloc dans la page
`body/md`	16 / 24	Regular	Le texte courant
`label/button`	16 / 20	Semi Bold	Les libellés de boutons
`caption`	13 / 18	Regular	Mentions, dates, aides

Variable ou style : comment choisir ?

Figma le dit lui-même : les variables ne remplacent pas les styles, elles les complètent. Retenez ce tableau.

Besoin	Le bon outil	Pourquoi
Une couleur, une marge, un arrondi	**Variable**	Une seule valeur, gère les alias et les modes
Un niveau de texte complet	**Style de texte**	Regroupe plusieurs réglages
Une ombre portée	**Style d'effet**	Regroupe décalage, flou et couleur
Un dégradé	**Style de couleur**	Une combinaison de plusieurs couleurs

Point de vigilance

Commencez petit. Inutile de créer deux cents variables le premier jour. Une dizaine de couleurs sémantiques, une échelle d'espacement et quatre ou cinq styles de texte suffisent pour démarrer. Vous ajouterez le reste quand un vrai besoin apparaîtra.

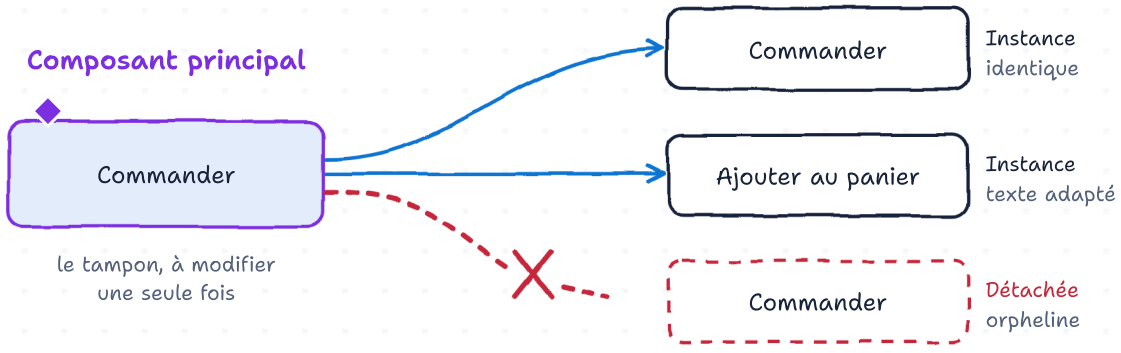
Les valeurs sont maintenant centralisées. Mais Alex redessine encore le même bouton sur chaque écran. Passons à l'étape suivante : réutiliser les objets eux-mêmes.

Chapitre 9

Les composants

Si les variables centralisent les valeurs, les **composants** centralisent les objets. Un bouton, un champ de saisie, une carte de produit apparaissent des dizaines de fois dans une application. Il n'y a aucune raison de les dessiner plus d'une fois.

Pensez à un tampon encreur. Vous le fabriquez une seule fois, puis vous imprimez autant de copies que nécessaire. Figma ajoute une touche de magie : si vous modifiez le tampon, toutes les copies déjà imprimées se mettent à jour.



Une instance peut personnaliser son texte sans perdre le lien. Une instance détachée redevient un dessin ordinaire.

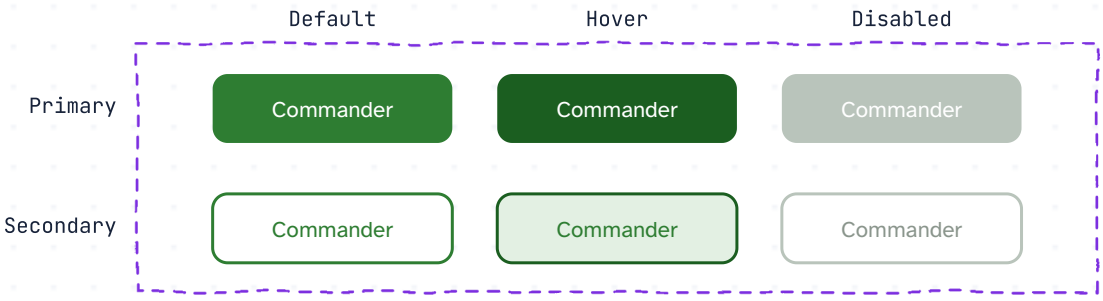
Sélectionnez votre bouton et faites `Ctrl + Alt + K` (`Cmd + Option + K` sur Mac). L'objet devient un composant principal, reconnaissable à son losange violet. Chaque copie que vous ferez sera une **instance**, qui reste reliée à son modèle.

Point de vigilance

La commande *Detach instance* semble bien pratique quand un composant vous résiste. Résistez. Chaque instance détachée est un morceau de votre écran qui ne recevra plus aucune mise à jour. Si vous en avez souvent besoin, c'est le composant qui doit évoluer.

Un composant, plusieurs états

Un bouton existe en version principale ou secondaire. Il peut être normal, survolé ou désactivé. Plutôt que de créer six composants sans lien, on les réunit dans un **ensemble de variantes**.



Deux propriétés, Type et State, décrivent six variantes. Dans Figma, on les choisit dans le panneau de droite.

Les propriétés : un tableau de bord pour l'instance

Vous pouvez décider ce que l'utilisateur du composant a le droit de modifier. Ces réglages, appelés **propriétés de composant**, apparaissent dans le panneau de droite.

Propriété	Rôle	Exemple sur Frisée
Variant	Choisir une version	Bouton primaire ou secondaire
Text	Modifier un texte	« Commander », « Annuler »
Boolean	Afficher ou masquer un calque	L'icône du bouton
Instance swap	Remplacer un élément imbriqué	Changer d'icône
Slot	Ouvrir une zone libre	Le contenu d'une fenêtre modale

Les slots sont un ajout récent et très attendu : on peut enfin glisser le contenu de son choix dans une zone prévue sans détacher le composant.

Tout s'emboîte

Le composant bouton utilise des variables. La carte de produit contient le bouton. L'écran contient des cartes. Remontons la chaîne le jour où le client demande de

l'orange : Alex modifie une primitive, ou repointe une sémantique, et les quarante écrans suivent. Lorsque cette organisation est partagée, documentée et publiée, on parle de **design system**.

> **Conseil de pro**
>
> Créez vos composants quand un élément se répète, pas avant. Et dans le composant principal, n'utilisez que des variables et des styles. Un composant rempli de valeurs en dur reproduit simplement le problème à plus grande échelle.

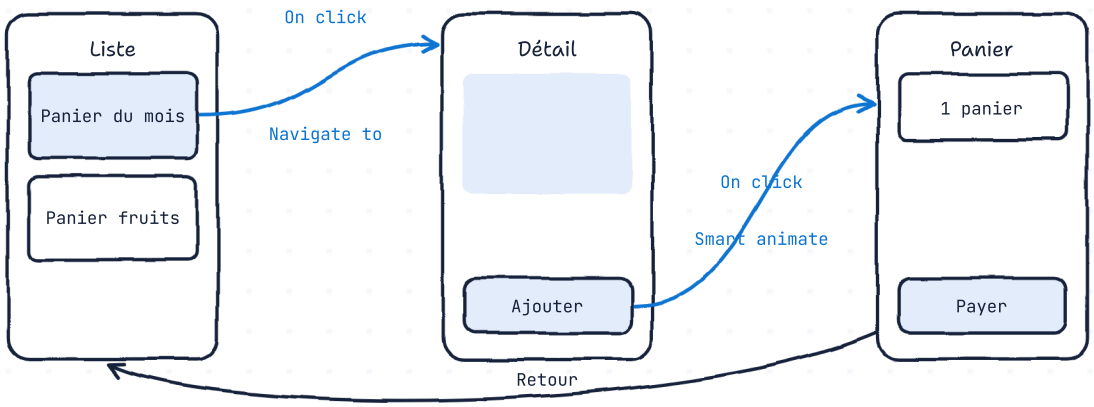
Votre interface est propre et cohérente. Il est temps de lui donner vie.

Chapitre 10

Prototyper

Une maquette statique ne dit rien du parcours. L'utilisateur comprend-il qu'il faut balayer l'écran ? Le récapitulatif arrive-t-il au bon moment ? Avant de mobiliser Sam pendant des semaines, Alex veut pouvoir faire tester l'application pour de vrai.

Passez sur l'onglet *Prototype* du panneau de droite. Un petit rond apparaît au survol d'un objet. Tirez-le vers un autre écran : vous venez de créer une **connexion**, ce que les designers appellent affectueusement une nouille.



Chaque nouille répond à trois questions : quand ça se déclenche, ce qui se passe, et avec quelle animation.

La grammaire d'une interaction

Élément	La question	Exemples
Déclencheur (*trigger*)	Quand ?	*On click, While hovering, After delay*
Action	Quoi ?	*Navigate to, Open overlay, Back, Set variable*
Animation	Comment ?	*Instant, Dissolve, Smart animate*

Parmi les animations, **Smart animate** vaut le détour. Figma compare deux écrans et anime les calques portant le même nom. Une image qui grandit entre la liste et le détail ne demande aucun réglage particulier, et vos calques bien nommés sont enfin récompensés.

Les composants interactifs

Faut-il relier à la main chaque case à cocher de chaque écran ? Non. On peut créer la connexion entre les variantes, à l'intérieur de l'ensemble : *Default* vers *Hover* au survol. Toutes les instances de tous les écrans réagissent alors sans effort supplémentaire.

Un prototype qui se souvient

Les variables ne servent pas seulement à l'apparence. L'action *Set variable* peut modifier une valeur pendant le test. Créez une variable numérique `cart/count`, reliez-la au texte de la pastille du panier, ajoutez 1 à chaque clic sur « Ajouter » : votre maquette compte vraiment les articles. Avec des conditions, on peut même afficher un message lorsque le panier est vide.

Faire tester

Le prototype n'a d'intérêt que si quelqu'un l'utilise. Partagez le lien, donnez une mission à un testeur (« commandez un panier pour mardi ») et taisez-vous. Le plus difficile est de ne pas aider. Chaque hésitation que vous observez est un problème que vous n'aurez pas à corriger dans le code.

> **Point de vigilance**
>
> Ne cherchez pas à tout simuler. Un prototype qui reproduit chaque cas d'erreur devient aussi long à construire que l'application. Prototypez le parcours que vous voulez valider, et rien de plus.

Le client a validé, les tests sont concluants. Il reste à transmettre tout cela à Sam, sans perte en route.

Chapitre 11

Du fichier au code

Autrefois, le designer livrait une image, et le développeur mesurait à la règle sur son écran. Le résultat s'éloignait toujours un peu de la maquette, et chacun accusait l'autre. Votre travail sur les variables va permettre d'éviter tout cela.

Partager la charte : les bibliothèques

Une charte a vocation à servir plusieurs projets. Créez un fichier dédié, publiez-le comme **bibliothèque**, et les autres fichiers de l'équipe pourront utiliser ses variables, styles et composants. Quand vous publiez une mise à jour, chaque fichier abonné reçoit une notification. Cette fonction demande une offre payante, mais on peut très bien tout apprendre au sein d'un seul fichier.

Une même langue pour Alex et Sam

Regardez ce qui se passe lorsque les noms sont bien choisis. Alex a créé une variable sémantique dans Figma, et Sam déclare exactement la même chose en CSS.

Ce que voit Alex dans Figma

```
color/action/primary     Clair : green/600     Sombre : green/300
space/md                 16
radius/button            8
```

Ce qu'écrit Sam dans son CSS

```
:root {
  --color-action-primary: var(--green-600);
  --space-md: 16px;
  --radius-button: 8px;
}

[data-theme="dark"] {
  --color-action-primary: var(--green-300);
}

.button {
  background: var(--color-action-primary);
  padding: var(--space-md);
  border-radius: var(--radius-button);
}
```

Lorsqu'une valeur nommée circule ainsi entre le design et le code, on parle de **design token**. La discussion devient limpide. Personne ne dit plus « tu as mis le mauvais vert ». On dit « ce bouton devrait utiliser `action/primary` ».

Sam n'est même plus obligé de recopier ces valeurs. Des extensions comme Tokens Studio permettent d'exporter les variables dans un fichier JSON, qui alimente ensuite le code. Un format commun, porté par un groupe communautaire du W3C, facilite ces échanges.

Dev Mode

Figma propose aussi un espace dédié aux développeurs, le **Dev Mode**. Sam y lit les mesures, les extraits de code, les changements depuis la dernière version, et surtout le nom des variables utilisées. Vous pouvez d'ailleurs renseigner pour chaque variable sa syntaxe de code, afin que l'inspecteur affiche directement `var(--color-action-primary)`.

Marquez un écran *Ready for dev* quand il est prêt : Sam saura ce qu'il peut commencer, sans deviner parmi vos brouillons.

La liste à vérifier avant de transmettre

- ☐ Aucun code hexadécimal ne traîne dans *Selection colors*.
- ☐ Les textes utilisent tous un style.
- ☐ Les marges et les arrondis viennent de variables.
- ☐ Aucune instance n'a été détachée sans raison.
- ☐ Les calques et les pages ont des noms lisibles.
- ☐ Les états sont prévus : vide, chargement, erreur.

Et l'intelligence artificielle ?

Figma et d'autres proposent désormais de générer des écrans ou un premier code à partir d'une description. Des assistants de code peuvent aussi lire un fichier Figma. Le résultat dépend directement de la qualité du fichier : avec des variables et des composants bien nommés, un assistant comprend ce qu'il manipule. Avec des `Frame 212` remplis de codes hexadécimaux, il devine. Votre rigueur vaut donc aussi pour les machines.

Vous avez fait le tour. Rassemblons l'essentiel, et mettons-le à l'épreuve.

Chapitre 12

L’essentiel à retenir

Si vous ne deviez garder qu’une phrase de ce cours :

Une décision de design ne doit exister qu’à un seul endroit. Tout le reste y fait référence.

Le jeu : variable, style ou composant ?

Pour chaque situation, choisissez l’outil le plus adapté, puis vérifiez votre réponse.

▼ La couleur de fond des cartes doit changer en mode sombre.

Variable, avec deux modes.

▼ Le même en-tête revient sur tous les écrans.

Composant, sans hésiter.

▼ Tous les titres de niveau 2 doivent partager police, taille et interligne.

Style de texte, dont les réglages utilisent des variables.

▼ L’arrondi de tous les boutons doit passer de 8 à 12 pixels.

Variable de type nombre, utilisée dans le composant bouton.

▼ Une ombre portée revient sur toutes les cartes.

Style d’effet.

▼ Le bouton existe en version normale, survolée et désactivée.

Composant avec des variantes.

▼ Le panier de votre prototype doit compter les articles ajoutés.

Variable numérique modifiée par l’action *Set variable*.

Le vocabulaire de l'équipe

Le mot	Ce qu'il veut dire
Frame (cadre)	Un conteneur, souvent un écran ou un bloc
Auto layout	Une règle de disposition qui s'adapte au contenu
Codé en dur	Une valeur saisie directement, sans référence
Primitive	Une valeur brute du nuancier
Sémantique	Une variable nommée d'après son rôle
Alias	Une variable qui pointe vers une autre
Mode	Un jeu de valeurs : clair, sombre, marque
Instance	Une copie reliée à son composant principal
Variante	Une version d'un composant
Design token	Une valeur nommée partagée entre design et code
Design system	La charte, les composants et leurs règles d'usage

Votre défi : l'application Frisée

Comptez une demi-journée, et dessinez l'application du fil rouge en suivant cet ordre.

1. Créez deux collections : `Primitives` et `Sémantiques`.
2. Définissez une dizaine de sémantiques en clair et une échelle d'espacements.
3. Créez quatre styles de texte reliés à des variables.
4. Construisez un bouton en auto layout, avec ses variantes.
5. Construisez une carte de produit qui contient ce bouton.
6. Assemblez trois écrans : liste, détail, panier.
7. Reliez-les en prototype, avec un compteur de panier.
8. Le test final : changez une seule primitive.

Vous saurez que c'est réussi si, à l'étape 8, l'ensemble de vos écrans change de couleur et qu'aucun élément ne résiste. S'il en reste un, vous venez de trouver une valeur codée en dur.

Pour aller plus loin

- Le guide officiel des variables, en anglais.
- La documentation des propriétés de composants.
- Les design tokens, expliqués par Figma.
- Penpot, pour comparer avec un outil open source.