Wireframing Academy
by balsamiq

INTRODUCTION to USER INTERFACE DESIGN through WIREFRAMES

balsamiq.com/learn

Introduction to User Interface Design Through Wireframes

This easy-to-follow guide will give you the confidence to talk about, review, and start designing user interfaces.

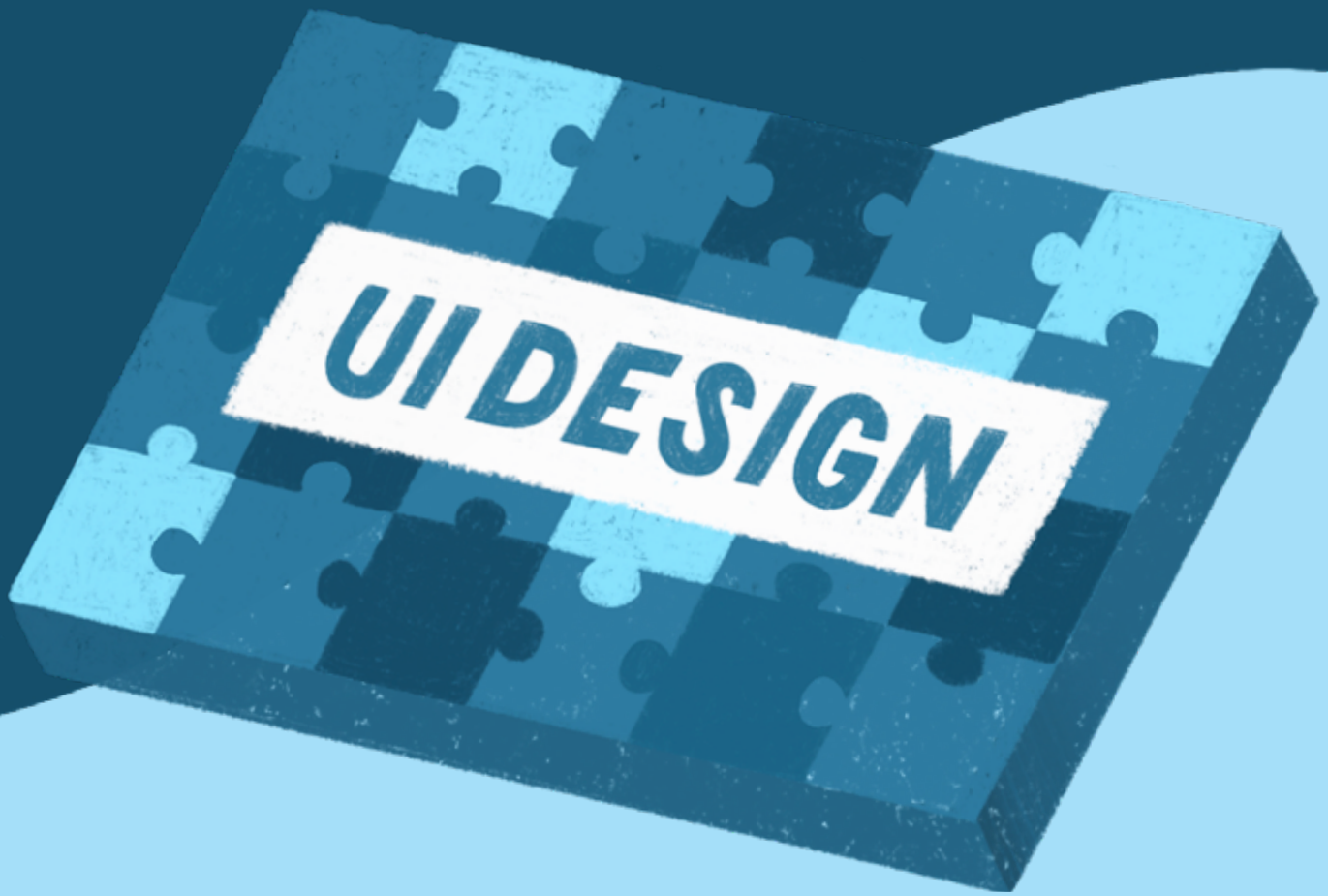
It teaches foundational UI concepts and shows you how UI professionals create effective wireframes.

It is intended for Product Managers, Entrepreneurs, and other non-designers who want to feel comfortable in the world of UI design.

-The Balsamiq Education Team

Table of Contents

1 What is UI Design



What is UI Design?

User Interface Design is the craft and process of designing what a user interacts with when communicating with software. We'll look at common aspects of this process, including how to apply design principles and interface conventions.

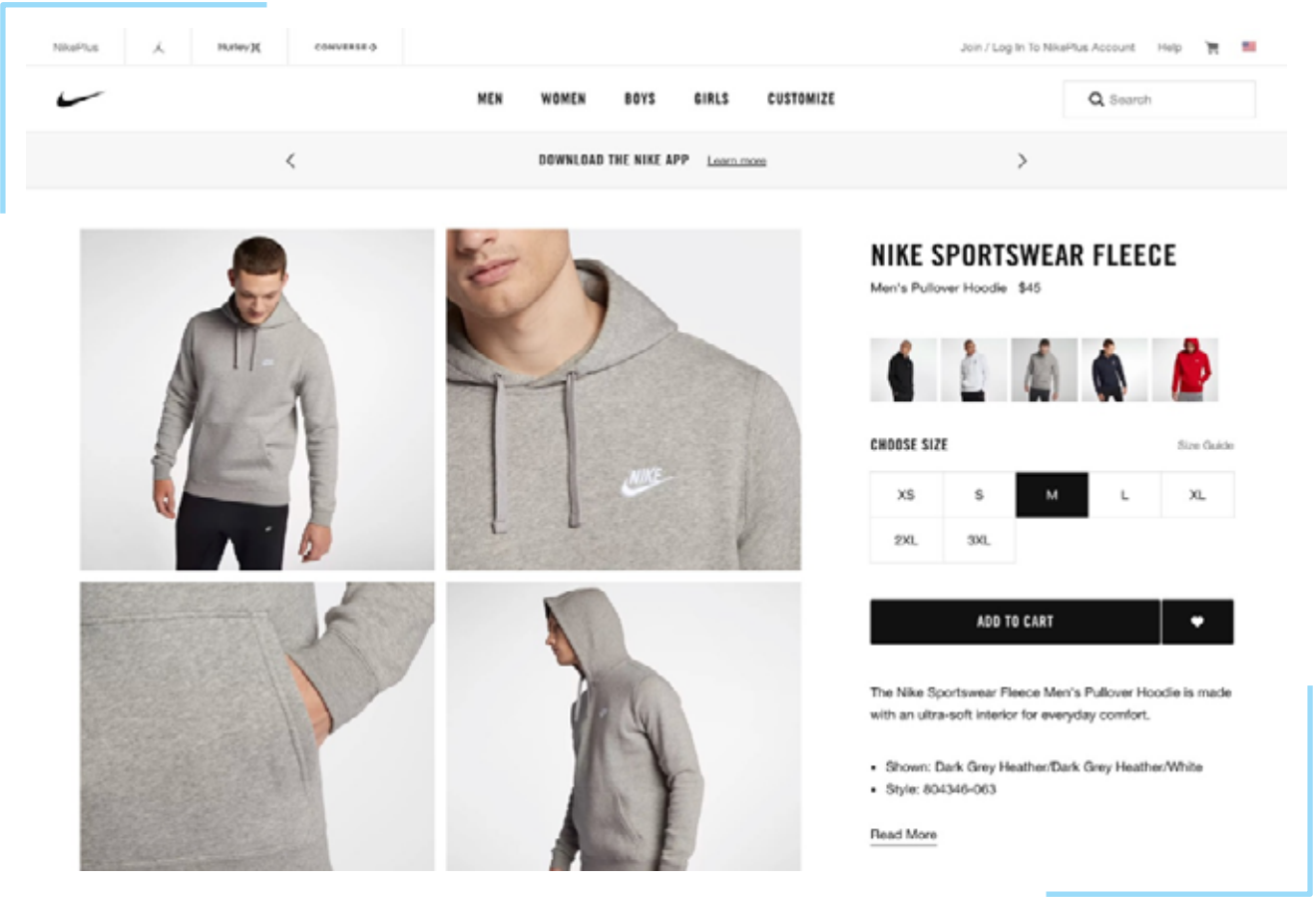
User interface design is surprisingly hard to define. The crowd-sourced (i.e., Wikipedia) definition seems to essentially say "user interface design is the design of software user interfaces," which isn't very enlightening.

A better way to define it is through the process of **deconstructing a user interface into the areas that a UI designer is concerned with**.

As we'll see, good UI design doesn't happen by accident.

There are actually multiple layers of UI design and multiple lenses that a UI designer looks through when creating a user interface.

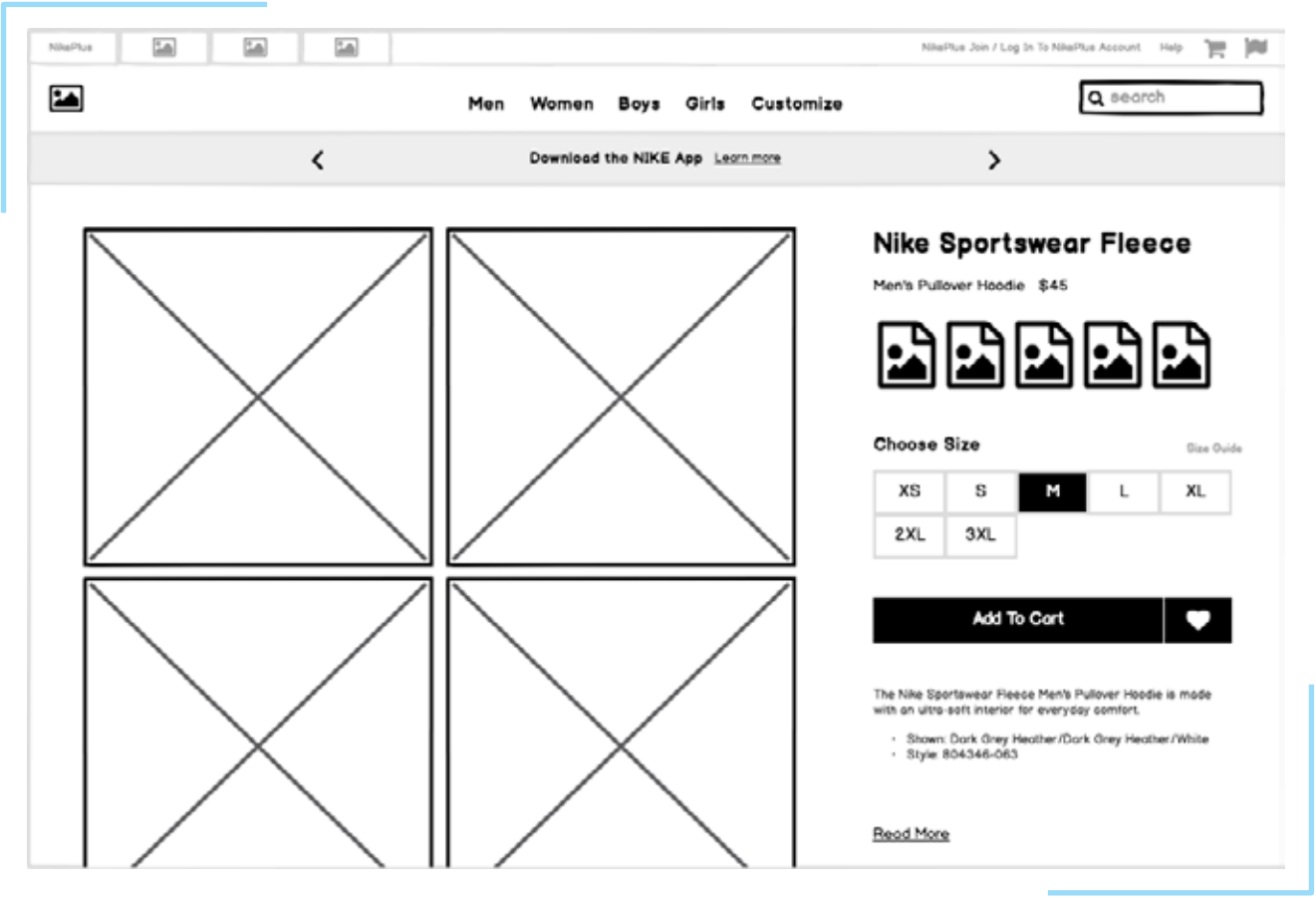
Here is a typical user interface on a web site. There are slightly different UI considerations for web, mobile, desktop and other types of software, but, generally speaking, they are all software, and the following concepts apply to all of them.



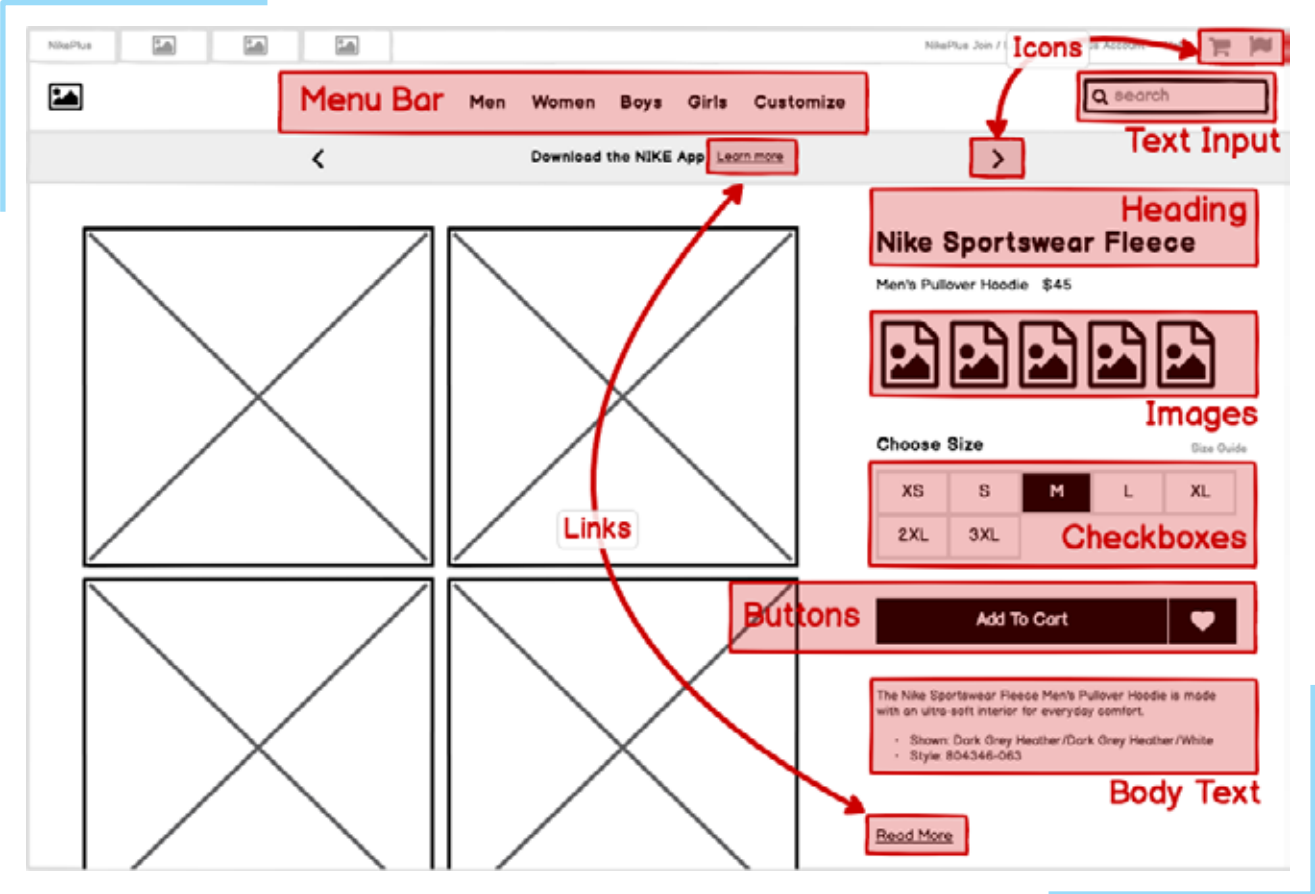
So, let's begin by stepping into the shoes of a UI designer to see how they might approach this website UI.

UI design layer 1: Controls

Before diving into the Controls layer, let's simplify the page above by viewing it as a wireframe.



Now let's look at some of the UI Controls that were used to build this page.



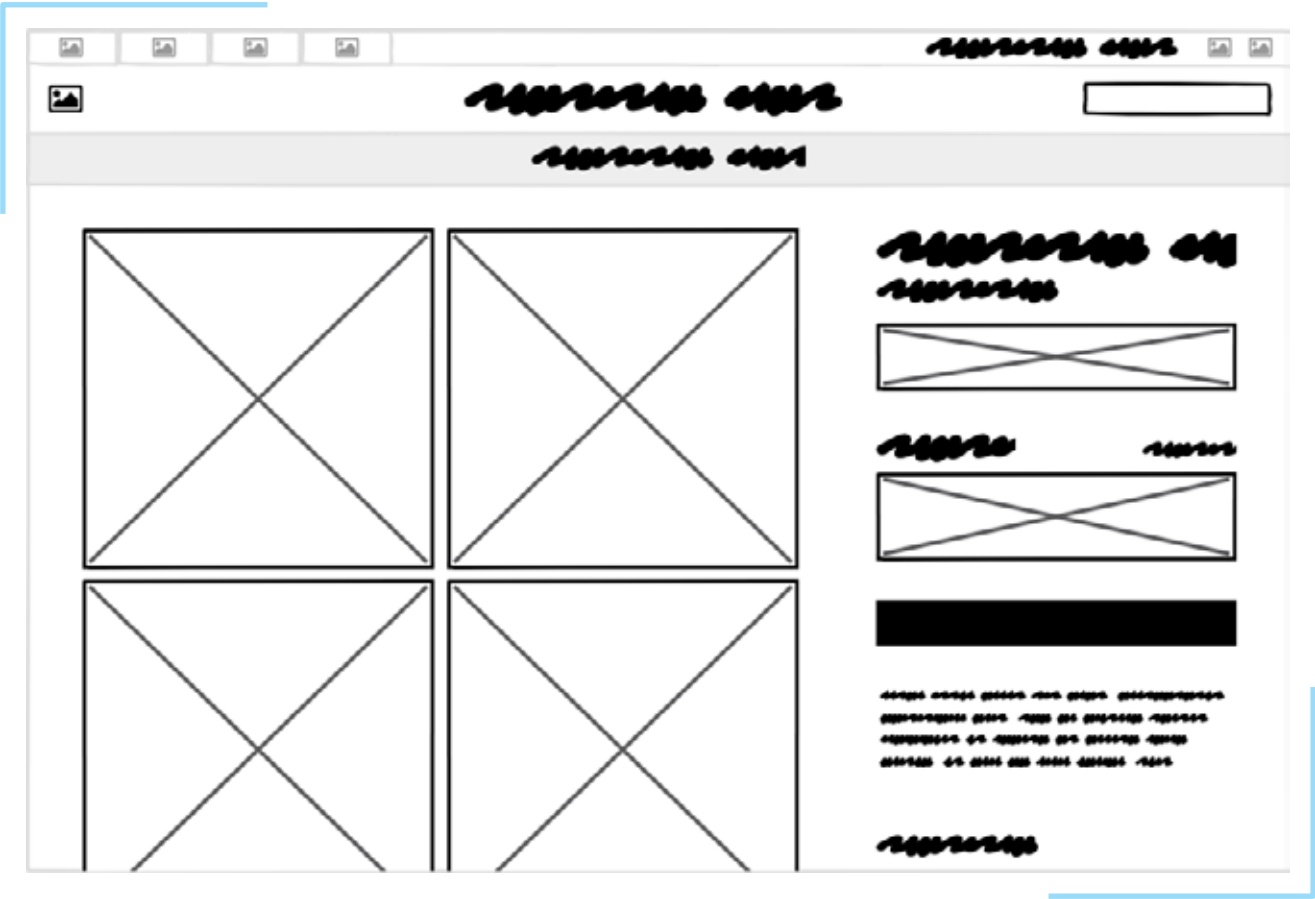
User interface controls (also known as elements, components, and "widgets") are individual pieces of a user interface that perform a single function. Some examples are links, buttons, and icons. Even plain text can be considered a control, since its function is to describe or label something within the user interface.

Each one of these controls was selected for a specific reason.
UI design is concerned with the process and rationale of choosing controls.

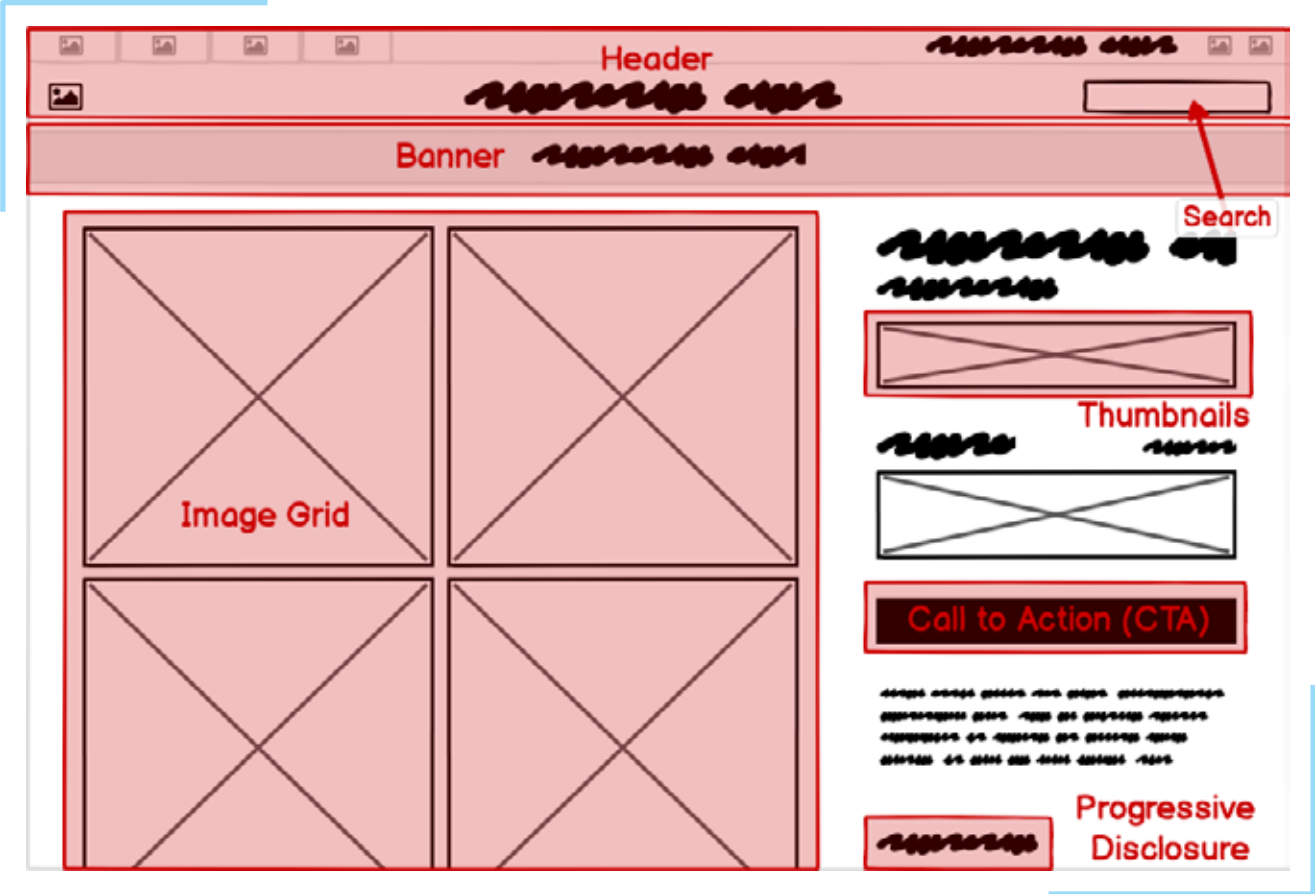
Read more about controls in Intro to UI Controls.

UI design layer 2: Patterns

We can further simplify this page by decreasing the fidelity of our wireframe to abstract away the individual controls, like this:



Now let's think about the *groups of controls* and what purpose they serve as units within the page. A UI Pattern is a group of controls that function to solve a particular problem. Let's look at some of the patterns on this page.



It can be useful to consider this layer of UI design even before moving on to the level of controls, as each pattern can meet its goals in different ways and using different controls.

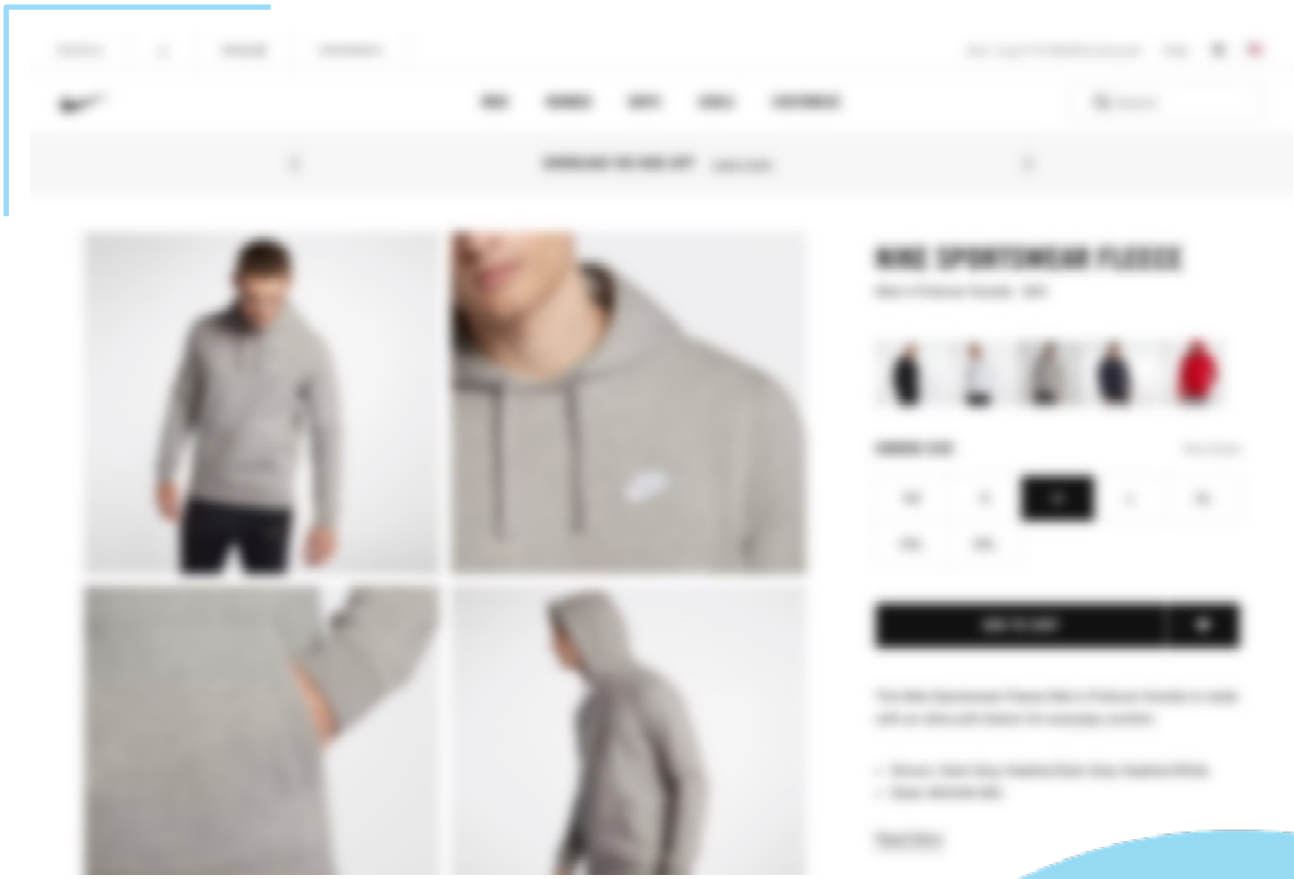
Read more about patterns in Intro to UI Design Patterns.

UI design layer 3: Design principles

The most commonly understood definition of UI design is the visual design layer. But even this is more purposeful than most people understand. Visual design isn't merely "making it look pretty." A better way to think of it is as the application of established **visual design principles**, many of which are rooted in scientific psychological, neurological, or physiological understanding.

The specific principles we'll cover in this course are **Contrast**, **Hierarchy**, **Proximity**, and **Alignment**.

One way that UI designers evaluate design principles is using the "squint test", which helps to further abstract the design into its visual principles. An alternative is to blur the screen.



Either way, the goal is to take your attention away from the content in order to **focus on the visual effects and techniques**.

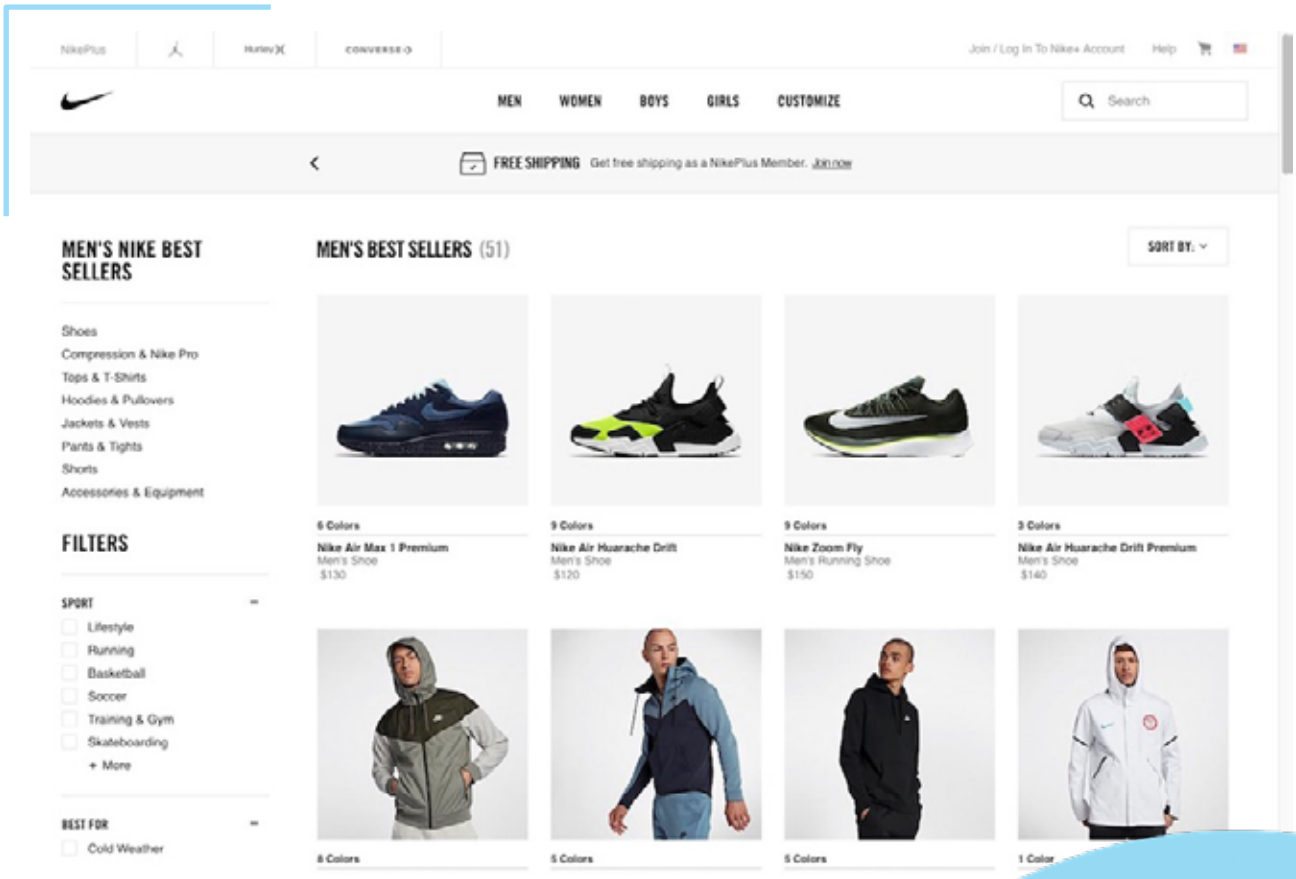
Read more about design principles in Visual Design Principles.

UI design layer 4: Templates

Finally, looking at this site as a whole, we can view this page as an instance of a **template** that can be reused across the site, rather than a single page that was designed for this particular article of clothing. For a site or product that can have dozens, or even thousands, of screens, it is useful both from the designer/developer and the end-user perspectives to have screens that **behave predictably and look similar across the entire application**.

The example we've been looking at so far could be described as a "product detail view" template that would look very similar when any other product is viewed.

Another UI template is the category template, shown here:



Other templates for this site might include one for checking out and purchasing, and another for search results. While every product may use different types of templates, all software types can benefit from a template-based approach to design.

Read more about templates in UI Design Templates.

The cooking analogy

In cooking there's a term called *mise en place*, which refers to the practice of putting "everything in its place" so that when the time comes to cook, you can automatically find what you need in the place you expect it to be.

The same will be true of the controls and patterns you use once you've familiarized yourself with them. In Balsamiq, you'll find the ingredients (the controls) in your UI Library. You may find the recipes (design patterns) in Wireframes to Go, or you'll build your own patterns for your ecosystem.

Once we've learned how to put together the pieces, we can create more complete solutions (using design principles and templates). We have some good awareness of ingredients, we have an understanding of our recipes. These fast become a toolset for our culinary work.

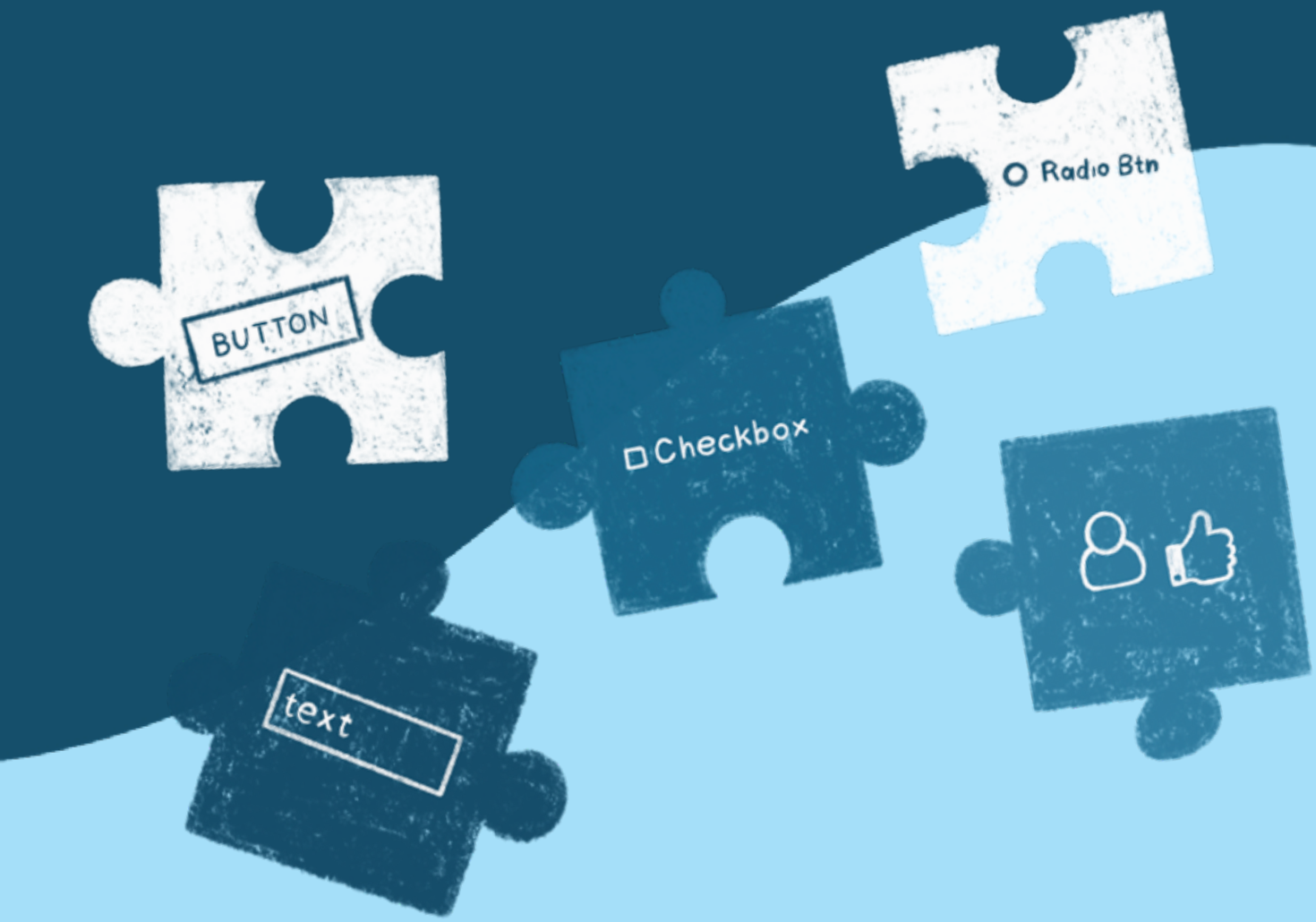
Finally, for a restaurant to be successful, it must figure out how to prepare food that comes out on time, in the right order, at the right temperature, and that is consistent across visits (the process).

Learn about the UI design process in our Opinionated Guide to Creating Software.

There's a lot to learn, but we're not going to go too deep in this course. If successful, it will give you *just enough* to feel comfortable designing or reviewing user interfaces in your own work. And hopefully have a little fun along the way!

Ready? Let's start with the ingredients (the UI controls)!

2 Intro to UI Controls



Intro to UI Controls

User interface (UI) controls are the building blocks of any software interface. Using them intelligently can guide users through your product as you intend, by making it feel familiar and learnable even if they have never used it before.

Becoming familiar with them as a user interface designer is essential for a good user experience. Many new designers get stuck trying to decide whether to use a checkbox or a radio button on a form, for example, or how many navigation tabs on a screen is too many. That's what this section is all about.

Fortunately, best practices and guidelines for user interface controls are well established, through years (often decades) of research and practice. In the next few articles we'll introduce the **most common user interface control types**, describe **when and how to use them**, and show **examples** and **variations** that will make you feel comfortable choosing and using them in your own designs.

UI controls are like the ingredients in a recipe. Learn their unique flavors and characteristics and you can improvise, customize, and substitute to meet your needs (or those of your specific users). Get to know them well enough and you can start creating your own recipes (design patterns) from scratch.

Below is the list of UI controls that we'll be learning about. Roughly speaking, they can be grouped as follows: those that accept **input**, such as the controls you'd find on a form; **navigation** controls, which allow users to move around in your app or site; and **output** controls, which communicate information to the user.

You can also come back to these later and jump to learning about UI Design Patterns next.

Button Guidelines

A button is used to execute an action. Designing buttons may seem obvious, but they are surprisingly complex and there are a few tricky things to look out for.

Text Input Guidelines

Text input fields allow keyboard input from the user. They are frequently used with other types of input controls in a form, but can be used on their own.

Dropdown Menu (Combo Box) Guidelines

A dropdown menu gives you a list of items to select from. Whether you call it a Dropdown menu, Combo Box, Pull Down menu, or Picker, you use them every day.

Radio Button and Checkbox Guidelines

Radio button and checkbox controls each allow users to select items from a list, but with different uses: when only a single selection is valid, or zero to more.

Link Guidelines

Links are the original way of navigating pages on the web and are so common that not adhering to best practices can break the usability of your site or app.

Tab Guidelines

Tabs can be a smart way to break up content into sections, but their use is a double-edged sword because they bury other content, making users guess where it is.

Breadcrumb Guidelines

Breadcrumbs are a compact, unobtrusive way to show navigation hierarchy. They show users where they are and provide an easy way to navigate up multiple levels.

Vertical Navigation Guidelines

Vertical (a.k.a. 'Sidebar') navigation is a way of showing a site or app structure along one side of them. Very common on the web, almost standard on mobile.

Menu Bar Guidelines

Menu bars allow users to navigate using categories and sub-categories. They are persistent, unchanging across the app, and used exclusively for primary navigation.

Accordion Guidelines

Accordions are stacked containers with nested items that expand and collapse when clicked or tapped. They can be used either for navigation or for content.

Validation Guidelines

Validation is a great way to present feedback or guidance with limited interruption. It provides in-context error messages and suggests ideas for improvement.

Tooltip Guidelines

Tooltips are a common form of contextual help and supplementary information. They should not be treated as a primary means of helping users understand the system.

Alert Guidelines

Any UI control that captures the user's attention can be thought of as an alert. They should be used wisely and sparingly so they are not overwhelming.

Data Table Guidelines

Data tables use columns and rows to display related information in a grid. They are best used for numerical data and lists of objects of the same type.

Icon Guidelines

With icons, the power of an image helps users identify things quickly and accurately. Useful for internationalization and for concepts hard to describe in words.

Button Guidelines

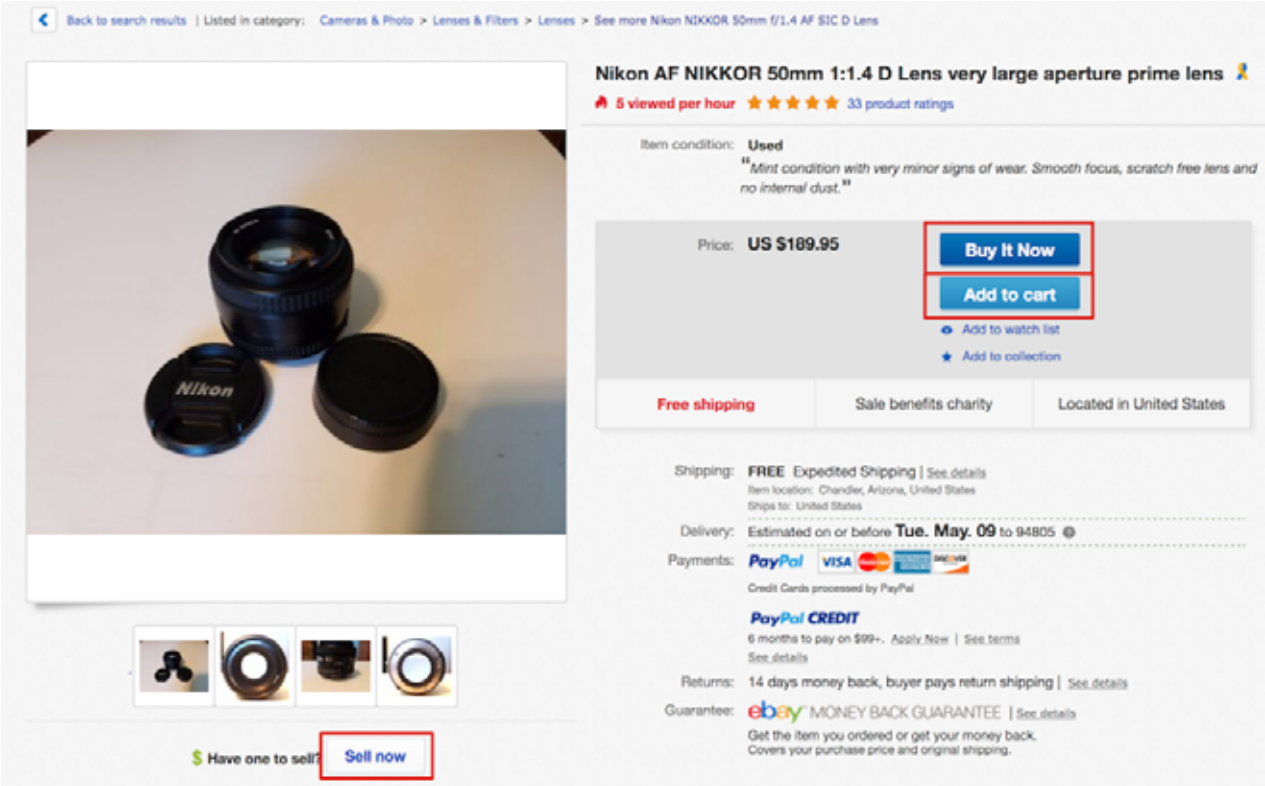
A button is a control used to execute an action, sending an email for example. Designing buttons may seem obvious, but they are surprisingly complex and there are a few tricky things to look out for.

When to use buttons

According to the book "Designing Interfaces", buttons should be "big, readable, obvious, and extremely easy to use for even the most inexperienced computer users". They are best used **for important actions**.

The challenge with buttons is that the more of them you add, the less obvious and easy to use each one becomes. So, use them wisely and deliberately.

Let's look at an example:



There are many elements on this page, including several actions the user can perform. But there are only 3 buttons, used for the most *consequential* actions: Buy It Now, Add to Cart, and Sell now.

How to use buttons

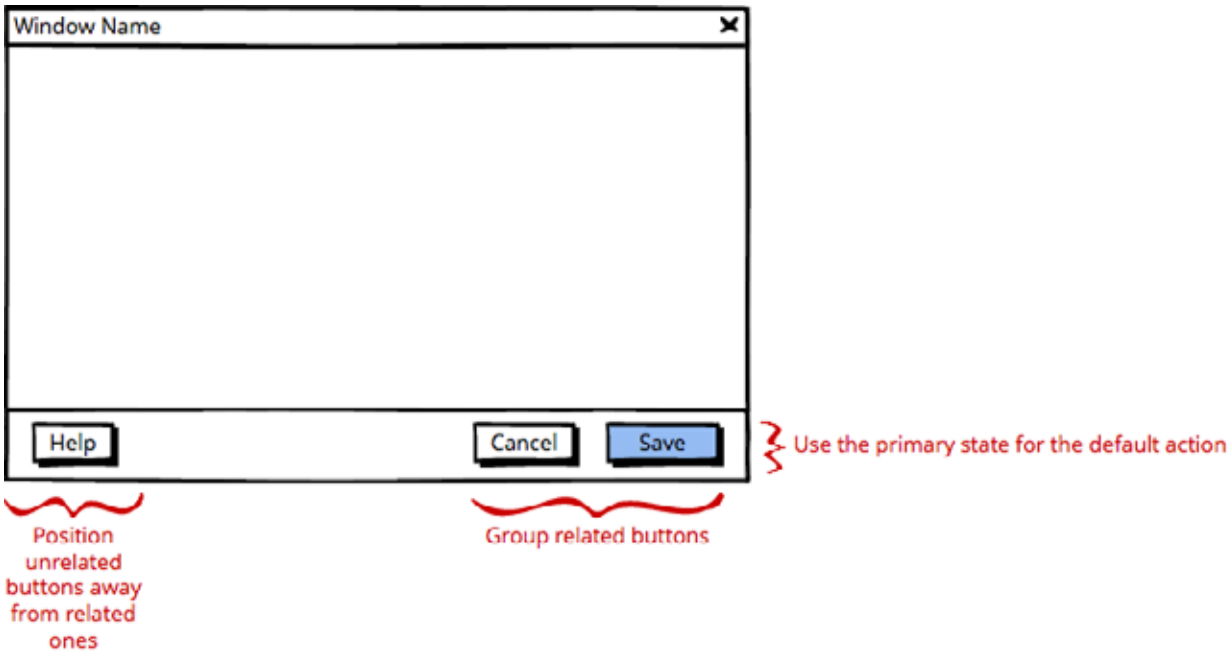
- Set the button that a user is highly likely to select as the default (primary).[1]
- Avoid using a button to mimic the behavior of other controls.[1]
- Use enough space between buttons so that users can click a specific one easily.[1]
- Avoid displaying an image in a standard button.[1]

- Use a verb or verb phrase and title-style capitalization for the button text.[1]
- Add an ellipsis to the title if the button immediately opens another window, dialog, or app to perform its action.[1]
- Separate destructive buttons from nondestructive controls.[1]
- Action should be immediate following a button press.[1]
- When several buttons are placed next to each other, ensure that they have the same width. This is particularly important for pairs of Cancel and OK buttons.[2]
- If pressing a button by mistake could cause a loss of data, do not set a default button.[2]
- Keep labels short, so they don't cause a button to use too much space. It is also important to consider how labels will change length when localized.[2]

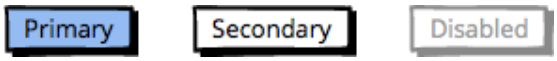
References

1. Apple macOS Human Interface Guidelines
2. GNOME Human Interface Guidelines

Basic usage



States



Variations

- **Icon buttons** - Buttons that have an icon accompanying, or in place of, the text.
- **Split Menu buttons** - Buttons that can be activated by pressing, but also offer a secondary menu of options in a drop-down menu.

Note: The Balsamiq examples shown above are available to import or download from Wireframes to Go. Read the instructions for using Wireframes to Go.

Related controls

- Links

Further reading

- Designing for Action: Best Practices for Effective Buttons
- “Designing Interfaces” by Jenifer Tidwell
- Google Material Design button guidelines
- Button UX Design: Best Practices, Types and States (UX Planet)
- Buttons in Design Systems (EightShapes)

Text Input Guidelines

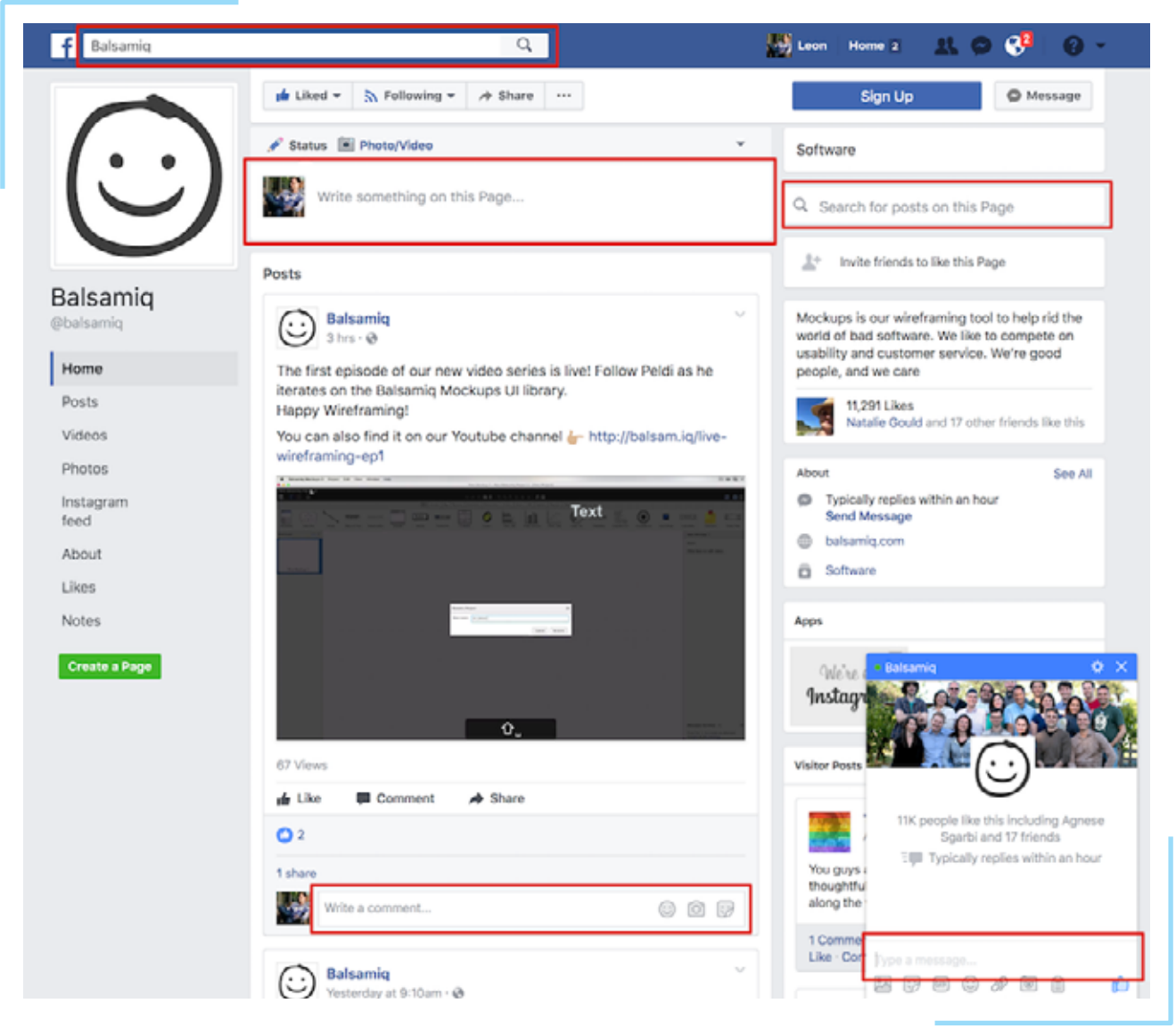
Text input fields allow keyboard input from the user. They are not as simple as they seem.

They are frequently used in combination with other types of input controls in a form, but can be used on their own.

When to use text input

Text input fields are used when requesting free-form input from a user, such as a username, phone number, password, or comment. They are one of the main components in a form. They are also frequently used for search, comments and chat.

Here is an example:



It is arguably **more important to know when *not* to use them**. Following the usability heuristic "recognition rather than recall", if you know in advance the list of possible, valid entries, it is better to use a dropdown menu (combo box) control or other constrained input control to reduce errors and facilitate entry. When asking for a country name, salutation, or payment method, for example.

The U.S. Web Design System suggests using them only when "you can't reasonably predict a user's answer to a prompt and there might be wide variability in users' answers."

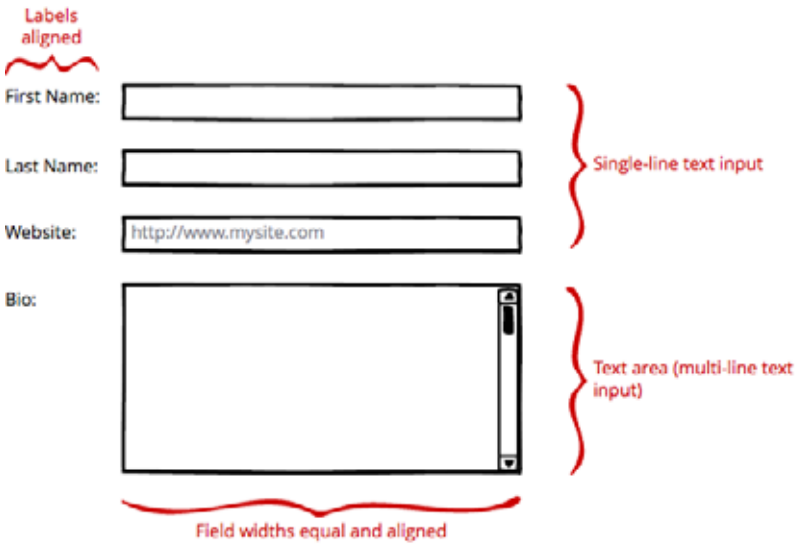
How to use text input

- Ensure that the length of a text input field comfortably accommodates the length of the expected input.[1]
- Text fields should indicate their state — whether enabled or disabled, empty or filled, valid or invalid — with a clear label, input, and assistive text.[2]
- Label text should be aligned with the input line, and always visible.[2]
- Label text shouldn't take up multiple lines.[2]
- Placeholder text (also called hint text) can be used inside the input field (see the "Website" field below), but it shouldn't take the place of a label in a form, since it should disappear when the user starts typing. (Also see this note about accessibility of placeholder text.)
- Avoid breaking numbers with distinct sections (such as phone numbers, Social Security Numbers, or credit card numbers) into separate input fields.[3] (Try using Input masks or flexible inputs instead.)
- For longer text, use a text area control (also called multi-line entry field), rather than a single line control.

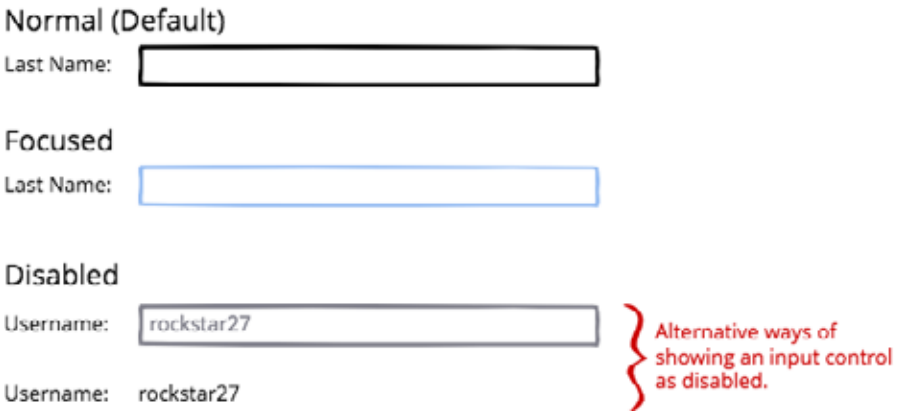
References

1. macOS Human Interface Guidelines
2. Google Material Design guidelines
3. U.S. Web Design System

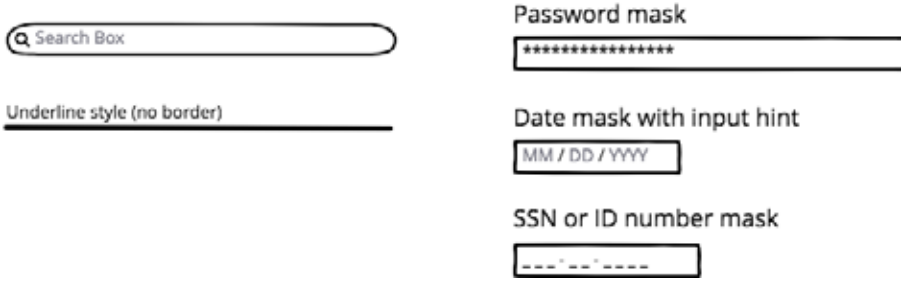
Basic usage



States



Variations



Related controls

- Dropdown Menu (Combo Box)

Further reading

- Jakob Nielsen's 10 Usability Heuristics for User Interface Design
- U.S. Web Design System Form Controls and Form Templates
- Forgiving Format Design Pattern
- "Web Form Design: Filling in the Blanks" by Luke Wroblewski
- Four Simple Rules for Effective Website Forms

Dropdown Menu (Combo Box) Guidelines

The control with many names! A dropdown menu gives you a list of items to select from. It is versatile and familiar.

Whether you call it a Dropdown menu, Combo Box (or Combobox), Pull Down menu (or Pull-down menu), Picker, Select menu, or something else, you use them every day.

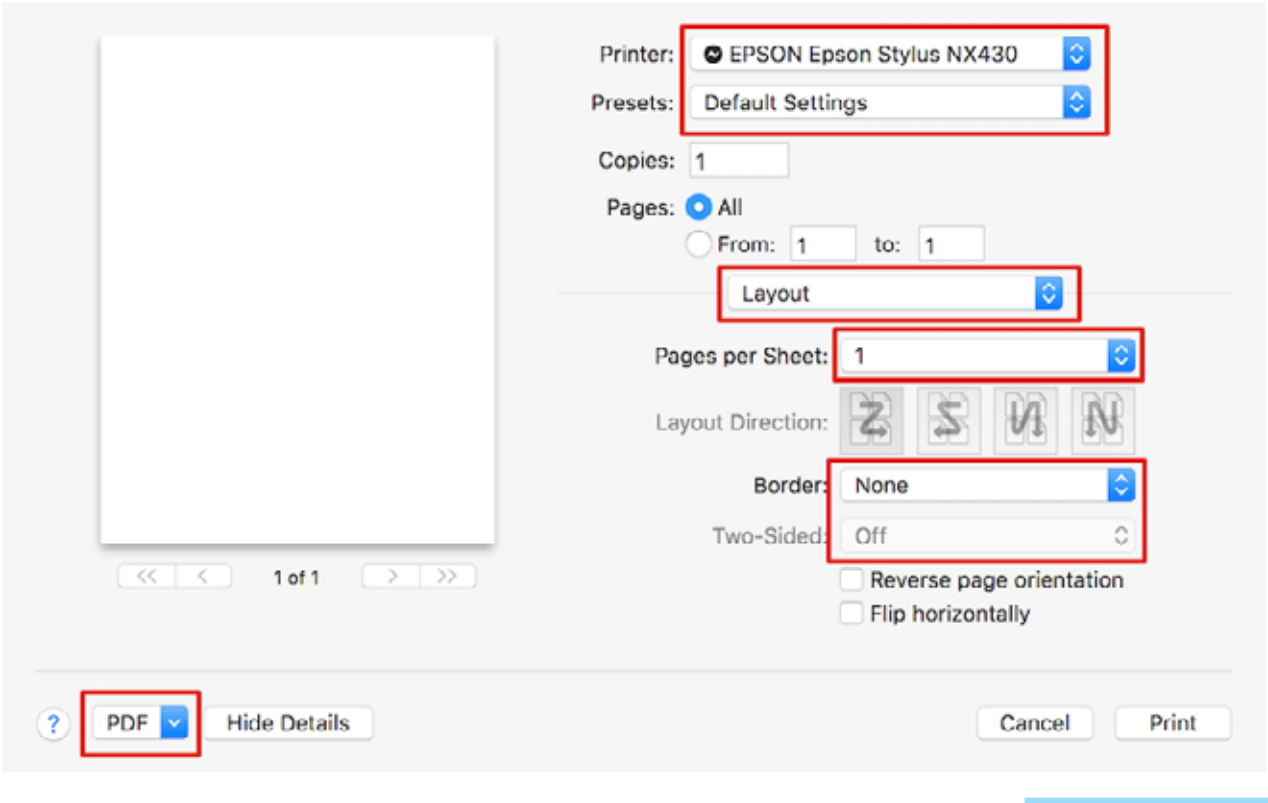
Technically, a dropdown menu is different from a combo box. A combo box is a *combination* of a dropdown menu and text input (confined to a set list of values). It is arguably more usable because you can type all or part of the value you want to select without having to scroll through the entire list of values. However, the dual nature of combo boxes is not obvious, so many users don't know that they can type into them.

When to use dropdown menus

Dropdown menus are a great way to **present a large number of options without taking up much space** on the screen.

They can also reduce errors, when compared to text input fields, because the input is constrained to the options available. Welie.com writes "The user may be familiar with the data but may not know the exact required syntax."

Some drawbacks of dropdown menus are that users can't see all the options at once and it can take time and dexterity to scroll. If you have a very long list, you can use a combo box or autocomplete text input field instead.



According to the U.S. Web Design Standards, the optimal number of items in a dropdown menu is between **7 and 15**. It suggests using radio buttons or checkboxes for shorter lists.

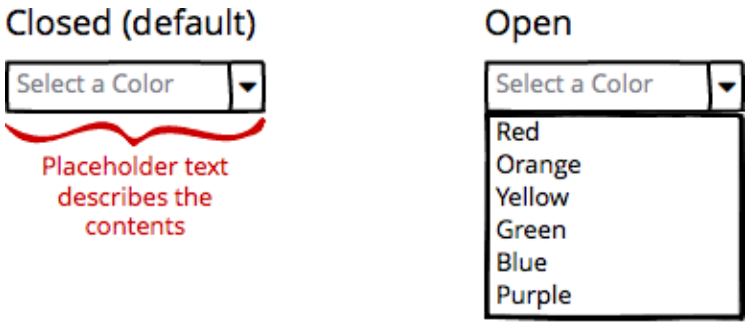
An exception is when the list is familiar so that users can expect to know the contents before opening it, such as days or months of the year, states/provinces, or countries.

The GNOME Human Interface Guidelines use the following example: "if you have an option menu labelled 'Month:' with the item 'January' selected, the user might reasonably infer that the menu contains the 12 months of the year without having to look."

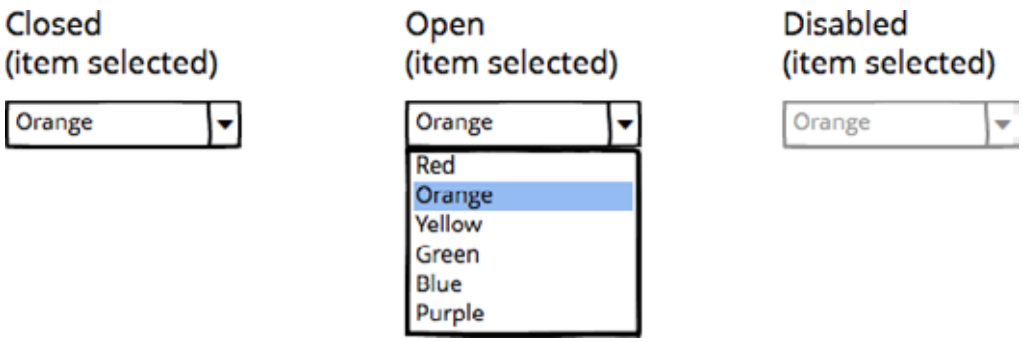
How to use dropdown menus

- Order the items logically (e.g., sequential for dates/numbers, alphabetical for countries).
- Display a meaningful default selection.1 *(Note: Pre-selecting an item can be dangerous, however, since you can't verify whether the user chose it deliberately. When in doubt, default to no selection.)*
- Provide relevant choices. Longer lists require more time to scan, so don't add options you don't expect users to select.[1]
- Avoid making options in one dropdown menu change based on the input to another. Users often don't understand how selecting an item in one impacts another.[2]
- Allow users to click anywhere on the control to open it, rather than only the arrow.
- Use grouping or categorization when it makes sense (see categorized dropdown variation below). Group headings or separators should not be selectable, however. (*In HTML, this can be achieved using the <optgroup> tag.*)

Basic usage



States



Variations

Related controls

- Text Input

Further reading

- Welie.com interaction design pattern library
- Autocomplete design pattern (UI Patterns)
- Dropdowns: Design Guidelines (Nielsen Norman Group)
- HTML Element Reference

References

1. macOS Human Interface Guidelines
2. U.S. Web Design System

Radio Button and Checkbox Guidelines

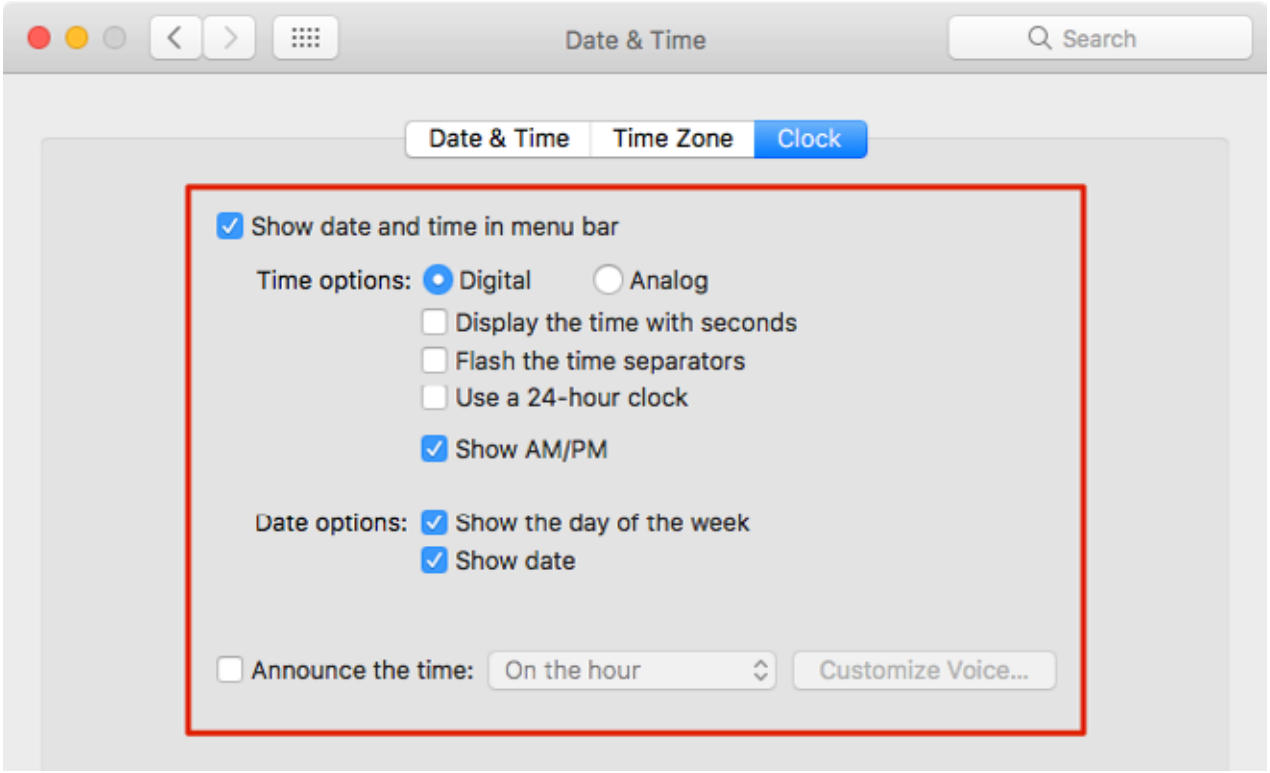
Radio button and checkbox controls each allow users to select items from a list. Despite that, they have different uses and guidelines.

Deciding when to use which one can be challenging, so this article explains the differences between them and when and how to use each.

When to use radio buttons and checkboxes

Like dropdown menus, radio buttons and checkboxes are appropriate when there is a predefined range of selection options. The difference between them is that radio buttons are for cases where **only a single selection is valid** (marital status, gender, etc.), whereas checkboxes support **zero or more selections** (preferred activities or interests, for example).

Unlike dropdown menus, radio buttons and checkboxes let **users see all options at once**. This can make selection faster. Their primary limitation is the amount of space they take up. Because of this, the GNOME Human Interface Guidelines recommend no more than "about 8" choices for a single group.



The most difficult scenario for choosing between radio buttons and checkboxes is when there is a binary (e.g., yes/no) choice, since either control could be used. The deciding factor should be whether the *second choice becomes obvious from the first one*. In the example above, radio buttons are used for Digital vs. Analog in the time options. These alternatives are familiar, but there could still be some doubt if only one were used with a checkbox. The alternative to "Show AM/PM", when using a checkbox, becomes *don't show* AM/PM, which is unambiguous.

How to use radio buttons and checkboxes

Guidelines for both

- Use a label to describe the group of choices *and* a label for each option within it, unless it is a single checkbox (see examples below).
- Vertical alignment is easier to read and parse. Use horizontal or rectangular alignments only if they greatly improve the layout of the window.[1]
- These controls shouldn't initiate actions on their own.[2] Use a button instead.
- Users should be able to click/tap the button/checkbox *or* its label to activate it.
- Be considerate when setting the default selection. Avoid dark patterns.

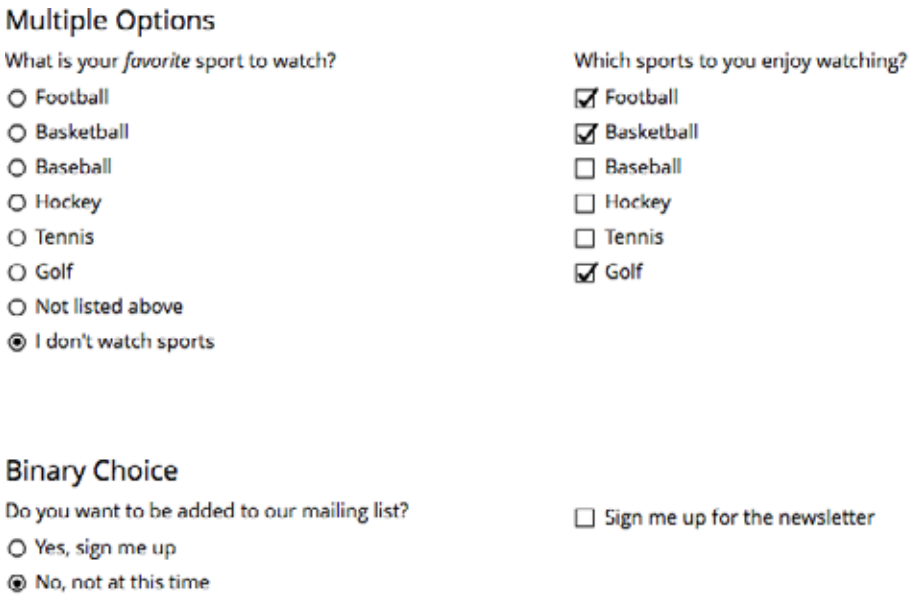
Guidelines for radio buttons

- Always use at least 2 radio buttons together. It doesn't make sense to use only one.
- Some guidelines state that one option in a radio button group should always be selected by default. If unsure which option that should be, add an explicit no choice option (such as "Unsure", "None", "Decline to State").

Guidelines for checkboxes

- Avoid using negative language in labels as they can be counter-intuitive, e.g., "Don't sign me up".[3]

Basic usage

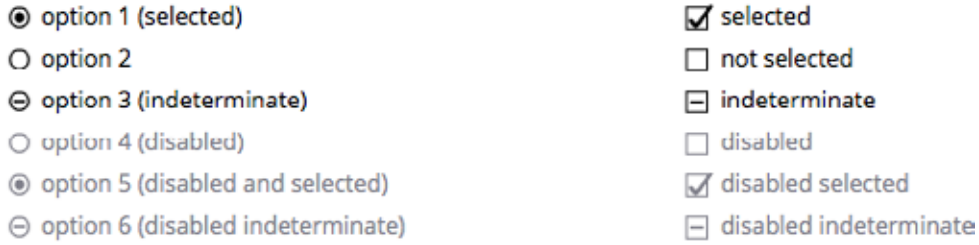


States

As with most form controls, radio buttons and checkboxes can be disabled as needed. One state that is unique to radio buttons and checkboxes is the non-binary **indeterminate** (also called mixed) state (neither on nor off).

The indeterminate / mixed state should *only* be used "to indicate that an option is set for some, but not all, child objects. [It] must not be used to represent a third state."[4]

The example below shows all states:



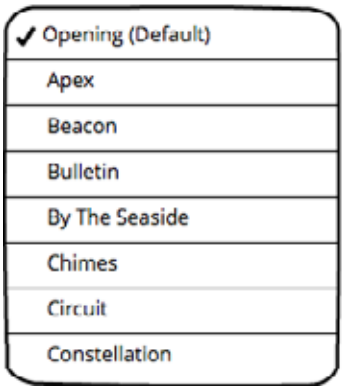
Variations

A **scrolling checkbox** group can be used when the number of items isn't known in advance or can be customized by the user and space is limited.

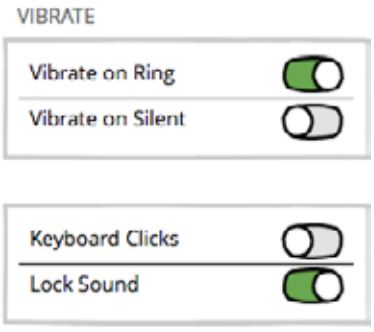
Scrolling checkbox group

On **mobile**, radio buttons and checkboxes can look different in order to be optimized for touch. On iOS, for example, check marks can be used for mutually exclusive options instead of radio buttons. And checkbox functionality may be indicated by a toggle switch instead.

iOS radio button



iOS checkbox



Related controls

- Dropdown menus

Further reading

- Dark Patterns

References

1. GNOME Human Interface Guidelines
2. macOS Human Interface Guidelines
3. U.S. Web Design System
4. KDE Human Interface Guidelines

Link Guidelines

Hyperlinks continue to be central to navigation on the web, such that not adhering to best practices can break the usability of your site or app.

When to use links

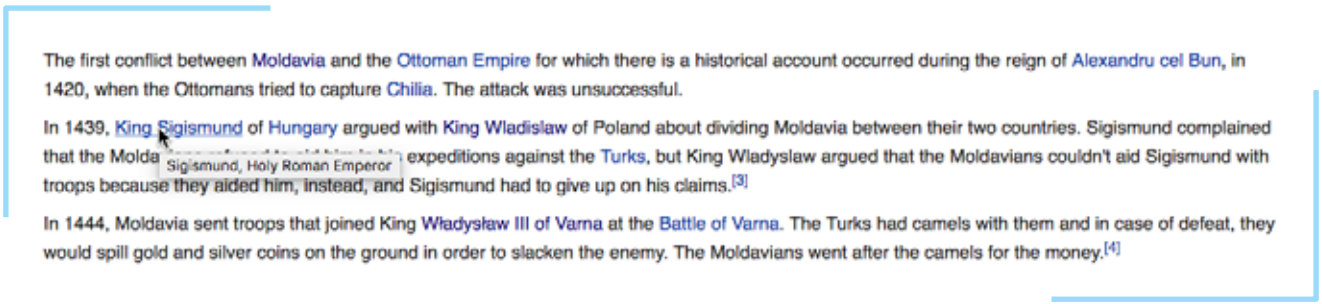
Links (originally called "hypertext links", then shortened to "hyperlinks", and now typically referred to just as "links") are the original way of **navigating from one page to another on the web**. They are ubiquitous on the web and common in web applications and web-like desktop apps.

Their strength lies in their simplicity. They can be embedded within blocks of regular text, allowing them to be read in context without interrupting the user's flow, while also indicating that content related to the linked text is available.

Additionally, as long as they are implemented correctly, they offer the advantage that they don't need much decoration to invite action (like buttons do).

This is because they are so standard that users expect to be able to click on text that has a distinct color and/or is underlined.

Wikipedia is a showcase for the use of links:



Links can be used within chunks of text to indicate the presence of related content, but can also stand on their own to attract more attention, such as for primary or secondary navigation, as in bread-crumbs, menu bars or vertical navigation.

Hyperlinks, when used to navigate between or within pages or screens, are familiar and easy to use (with a mouse or keyboard, at least), so they can be used liberally. Although having too many redundant links on a page can reduce usability.

How to use links

- Links should be visually distinct from other items on the page (preferably using color and underlining; see exceptions to this rule) and don't use that style for non-clickable items.[1]
- More than one link style on a page is OK, but should be done purposefully (e.g., one style for body text, another for header or footer text).
- Use the correct link markup for text that links to another page (i.e., the HTML <a> tag). Don't use javascript to perform an action on click.
- Use a different color for visited links.[2]
- Link text should make sense on its own (e.g., don't write "click here").[3]
- Link text should be as succinct as possible.[4]
- Use link titles (via the HTML "title" attribute) to help users predict where a link will lead before they click it, unless it's obvious.[5] On the web, these can function as tooltips.
- Consider using an outgoing (external) link indicator when linking to pages outside of your domain (see variations below for an example).[6]
- Avoid using text links to trigger actions.[7] They should be primarily used for navigation.
- Ensure sufficient contrast between link colors and their background.[8]

Basic usage

States

- **Default** - A normal, unvisited link.
- **Hover** - Occurs when the user mouses over it.
- **Active** - Occurs the moment it is clicked.
- **Visited** - A link the user has visited.

As described above, it is important to have a distinct default link style and a separate style for visited links.[9] The other link states are “active” and “hover”. But note that hover is not available on mobile devices, so don’t rely on it.

Variations

It can be helpful to indicate links that go to external sites using a small icon, such as the one below.

Related controls

- Buttons
- Breadcrumbs
- Tooltips

References

1. Nielsen Norman Group
2. Nielsen Norman Group
3. Nielsen Norman Group
4. Moz
5. Nielsen Norman Group
6. Welie.com
7. KDE Human Interface Guidelines
8. U.S. Web Design System
9. CSS Link styles

Further reading

- The Same Link Twice on the Same Page: Do Duplicates Help or Hurt? (Nielsen Norman Group)

Tab Guidelines

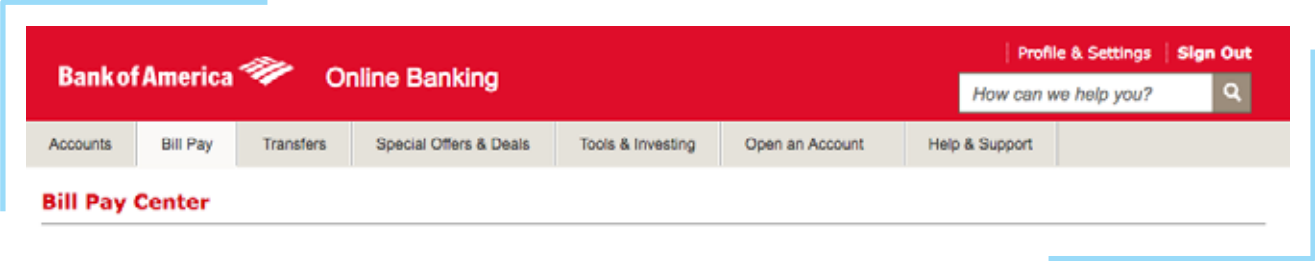
Tabs can be a smart way to break up content into sections, but their use is a double-edged sword.

On the one hand, **it focuses attention on a subset of content** to make the page easier to take in. On the other hand, **it buries other content**, making users guess where it is, or if it even exists. It is an example of the Master/Detail screen pattern.

In "About Face 3: The Essentials of Interaction Design", Alan Cooper says "navigation is excise" — meaning that any time a user is required to jump from one page to another, it adds a cognitive cost to their experience. The authors write: "the work that users are forced to do to get around in software and on Web sites is seldom aligned with their needs, goals, and desires" and urge designers to minimize the amount of navigation required.

When to use tabs

Tabs are one of the most popular navigation patterns (along with menu bars and vertical navigation). The biggest advantage of tabs is that they are familiar and often persistent, so that even when a user has navigated around in a site or application, they don't feel lost.



There are a few considerations to keep in mind when using tabs, however. Such as:

- Only use tabs when there is a limited number of navigation options available (up to 5 on mobile[1], less than 7 on desktop[2]).
- Tab width is usually determined by the text in each tab, so consider the impact of localization and font size adjustments.
- Only use tabs "to present closely related peer areas of content."[3] Content separated by tabs should be related in some way and exist at the same level in a hierarchy.
- Avoid using tabs for sequential tasks or "wizards" — tabs should be able to be used independently from each other.[4]

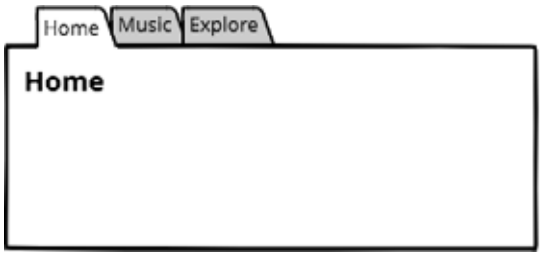
How to use tabs

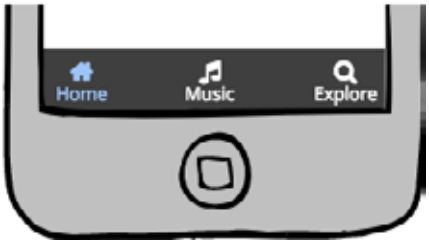
- Avoid using multiple sets of tabs.
- If you do, make sure to visually distinguish the second set from the first (see variations below)
- Put the most important content in the first tab.
- Make sure the controls within a pane (the area that the tab applies to) only affect content in the same pane.[3]

- You should never have only one tab.[3]
- Don't wrap tabs to a new line. If the tabs won't fit, consider using scrolling or drop-down tabs, as shown in the variations below.
- Be wary of using icons alone for tabs. Adding text above or below is recommended. "[F]or most icons, text labels are necessary to communicate meaning and reduce ambiguity."[1]

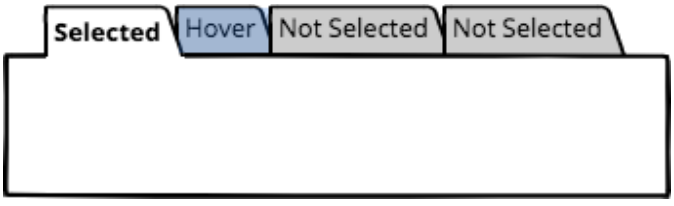
- Consider vertical tabs when the number of horizontal tabs would be too many (or use a different kind of navigation entirely).

Basic usage





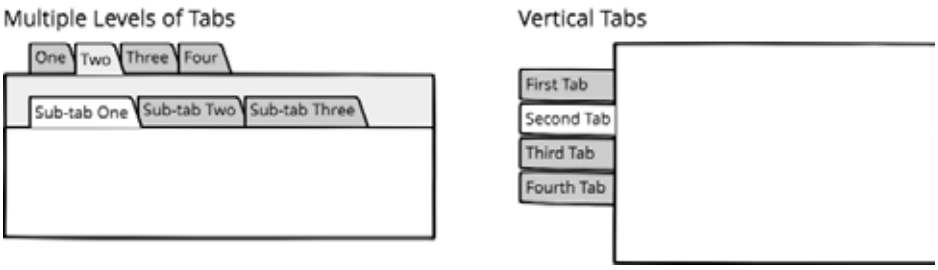
States



Tabs should have 2 primary states: **selected** and **non-selected**. A hover state can also be used to invite action, similar to a button. As shown above, the selected tab should be visually distinct from the non-selected tabs, with the selected tab more prominent (higher contrast) than the others. Bold text can be used to emphasize the selected tab.

Variations

Here are some common tab variations.



Tabs often have borders around the content area they refer to, but they aren't required.

Related controls

- Menu Bars
- Vertical Navigation

Further reading

- 12 Standard Screen Patterns
- “About Face 3: The Essentials of Interaction Design” by Alan Cooper

References

1. Smashing Magazine
2. KDE Human Interface Guidelines
3. macOS Human Interface Guidelines
4. Microsoft Windows Desktop Guidelines

Breadcrumb Guidelines

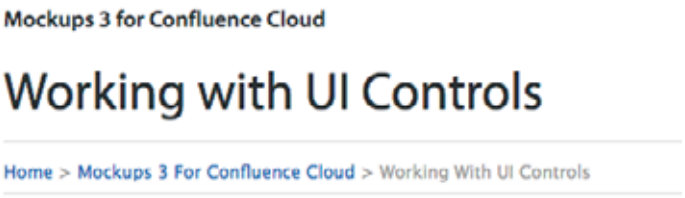
Breadcrumbs are a compact way to show navigation hierarchy. They are unobtrusive and not distracting.

They not only show users where they are, but provide an easy way to allow them to navigate up multiple levels.

They only require a small amount of space and are very familiar to most users. The Nielsen Norman Group writes that "user testing shows **many benefits and no downsides to breadcrumbs** for secondary navigation."

When to use breadcrumbs

As stated above, breadcrumbs are considered *secondary* navigation, meaning that they **shouldn't be provided as the only way for users to navigate.**[1] This is because they are not as obvious or noticeable as other navigation methods, such as Tabs.



You can use breadcrumbs when there is no visible way to navigate back to the parent page. Breadcrumbs are not needed, for example, with hierarchical vertical navigation, such as a tree control, because the navigation path is always visible.

Breadcrumbs can be preferable to other hierarchical navigation controls when space (especially in the horizontal direction) is constrained, such as on mobile. They might not be ideal for very deep hierarchies, however, where they can become very long. See the variations below for how they can be condensed in these cases.

How to use breadcrumbs

- Include the name of the current page as the last item in the breadcrumb, but don't link it.[2] It is generally good practice not to include links to the current page.
- Even though the title of the page may be indicated in the breadcrumb, it is good practice to repeat it below, as the breadcrumb itself is often small.
- Use a single character to separate the links. The most common separators for breadcrumbs are the ">" and "/" characters.
- There is some debate about whether breadcrumbs should show the site/application hierarchy or the path that the user has taken (i.e., more akin to the origin of the name "breadcrumb"). However, most guidelines recommend the former, where the links show the site hierarchy, rather than user's path.[2]

- Place breadcrumbs above the content, but not above any primary navigation (such as a horizontal or header menu).[1]
- Avoid using multiple sets of breadcrumbs on one page.

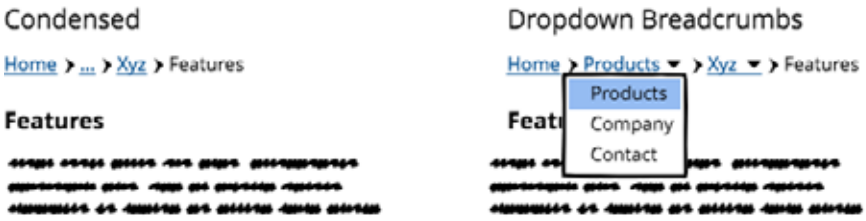
States

Breadcrumbs should be constructed from standard links and text, and should inherit the same states. As with standard links, breadcrumb links may have normal, hover, active, and visited states.

Variations

Condensed Breadcrumbs - You may use this pattern when the number of items exceeds about 5 or as space requires. Clicking on the "..." can expand the entire list, or only the last few items.

Dropdown Breadcrumbs - This is a less common pattern that combines breadcrumbs with a vertical menu to allow users to navigate non-linearly. It is not standard and should be used sparingly. See examples and guidelines for this pattern.



Related controls

- Dropdown Menu (Combo Box)
- Links

Further reading

- “Scotch Egg” Navigation

References

1. KDE Human Interface Guidelines
2. Nielsen Norman Group

Vertical Navigation Guidelines

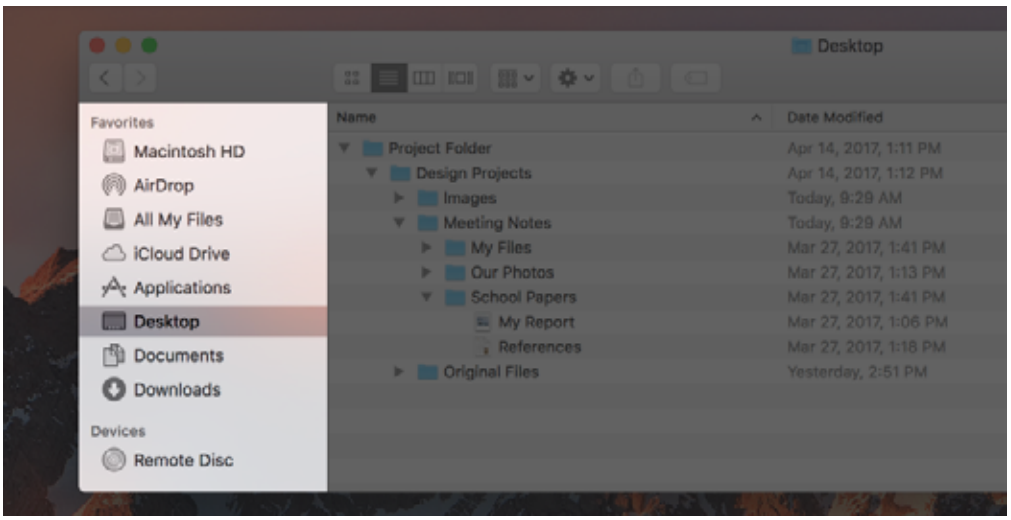
Vertical (a.k.a. “Sidebar”) navigation is a way of showing a persistent site or application structure along one side of the product.

The macOS Human Interface Guidelines define it by saying “a sidebar typically consists of a table view or outline view that lets people navigate and select items to act upon in the main portion of the window.”

Vertical navigation is used extensively on the web and is becoming much more common, almost standard, on mobile via the slide-out “navigation drawer” pattern.

When to use vertical navigation

Vertical navigation, like tabs, is a member of Master/Detail family of patterns, which is described as “ideal for creating an efficient user experience by allowing the user to stay in the same screen while navigating between items.”



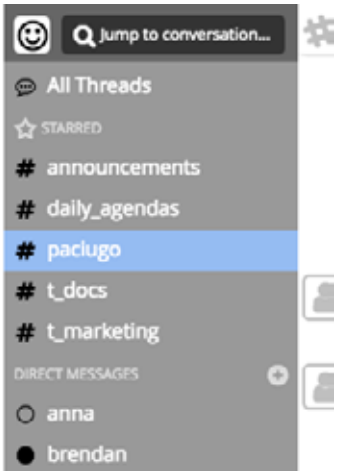
Unlike tabs, vertical navigation is appropriate **when the number of categories is not small, or is user-customizable** (such as folders or tags in an email client). It is considered a "safe" navigation pattern because it is familiar, flexible, and doesn't take up much space. It is often used when there is no other obvious choice.

One primary reason not to use it is when horizontal space is constrained. This is why many websites (including this page) remove the vertical navigation for small screen sizes, replacing it with either breadcrumbs or the popular "hamburger" menu navigation pattern. The Material Design guidelines differentiate between 4 distinct variants, based on screen size and content: **standard, modal, permanently visible, and dismissible**.

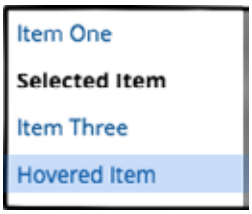
How to use vertical navigation

- Highlight the selected page/item in the list. It should not be styled as clickable like the other items (even if it behaves the same).
- Use titles to form logical groupings of related items.[1]
- Clicking or tapping on category labels should expand or collapse that category rather than linking to its own page.
- Keep the navigation link names short. They can be shorter derivatives of page titles themselves.[2]
- Order the list according to what is most useful for the users of your application.[3]
- Having hierarchical data doesn't mean that you must use a tree view. Very often a list view is a simpler, yet more powerful choice.[4]
- In general, refrain from exposing more than 2 levels of hierarchy within a sidebar.[1]
- The sidebar can be on the left or right side of the page, but it should be consistent across the application.
- Consider what happens when the sidebar content is longer (taller) than the page content. Ensure that users can still access the entire list (i.e., scroll beyond the page contents).
- Consider replacing the navigation panel with a slide-out panel on small screens or breadcrumbs on desktop displays.

Basic usage



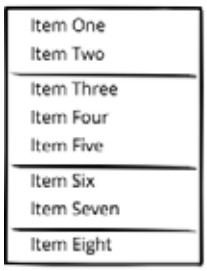
States



Like tabs, vertical navigation should have a clear selected state, usually achieved by making the selected item bold and/or highlighted. A hover state can also be useful.

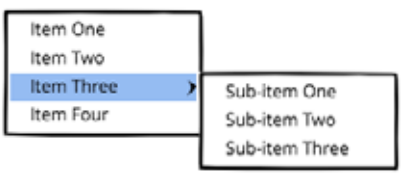
Variations

Grouped (no labels)

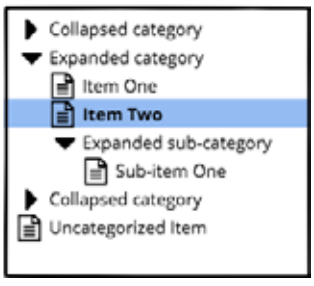


With Icons

Sub-menus



Hierarchical



Related controls

- Tabs
- Menu Bars

Further reading

- 12 Standard Screen Patterns
- 10 Sites Doing Vertical Navigation Right

References

1. macOS Human Interface Guidelines
2. U.S. Web Design System
3. GNOME Human Interface Guidelines
4. Microsoft Windows Desktop Guidelines

Menu Bar Guidelines

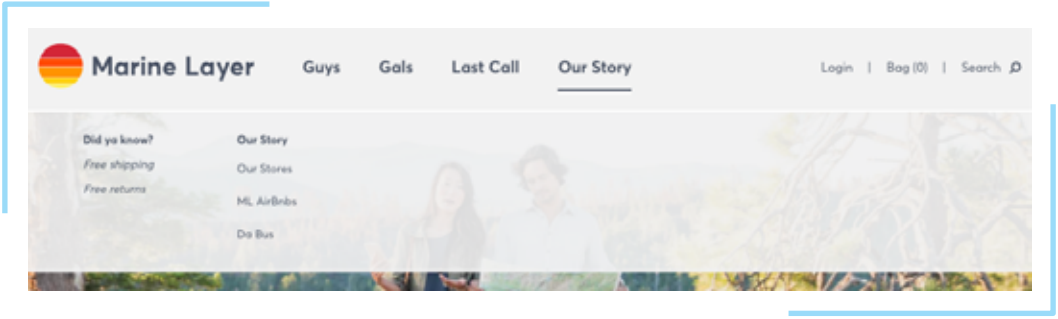
Menu bars allow users to navigate using categories and sub-categories. They are persistent and unchanging across the app.

In a desktop application, menu bars are the items across the top of the application, a.k.a. the File menu. On the web, they reside in the header/top of the page. In both cases, they can support nested menus or work as stand-alone categories (also called "monocline grouping" or flat structure).

When to use menu bars

Menu bars are used exclusively for **primary navigation** (unlike vertical navigation or tabs, which can also be secondary navigation).

They should be used for categories of the application or site that are **meaningful across its entire use**, provided there are not too many categories. If there are too many categories to fit across the page, consider vertical navigation instead.



The main advantage of menu bars is their persistence, i.e., the fact that they are always accessible. This is highlighted in "About Face: The Essentials of Interaction Design":

> *Well-designed Web sites make careful use of persistent objects that remain constant throughout the shopping experience, especially the top-level navigation bar along the top of the page. Not only do these areas provide clear navigational options, but their consistent presence and layout also help orient customers.*

Menu bars can contain sub-menus (as shown above), but going any deeper than one level should be avoided if possible because it pushes users' mental models, whereas "organizing things into a single layer of groups is extremely common and can be found everywhere in your home and office."

How to use menu bars

The macOS Human Interface Guidelines[1] provide a comprehensive guide to using menus in desktop applications (many of which apply to websites as well). Some highlights:

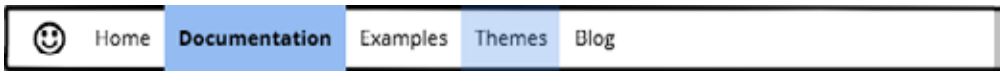
- Make menu titles as short as possible without sacrificing clarity. Ideally they should be limited to one word.
- Use text, not icons, for menu titles. (One exception is that it is generally acceptable to use a logo or graphic in place of "home" text on the web, as shown in the variations below.)
- Disable, don't hide, unavailable menu items.
- Limit the use, depth, and length of submenus.
- Assign keyboard shortcuts and display them next to their associated menu items.
- Use separator lines to create visually distinct groups of related menu items.
- In general, place the most frequently used items at the top of a menu.
- Group menu items that initiate related actions.
- Use a checkmark to indicate that something is currently active.

Other important menu bar guidelines:

- Don't model your navigation after your agency's org structure (or your application's architectural model). Instead, **structure it according to the tasks and information your users most frequently need to access.**[2]
- Include skip navigation links to allow users with screen readers to bypass long navigation lists.[2]
- Consider using a "mega menu" if you have more than 6 links or menu items within a menu. (See the variations below for more on mega menus.)[2]
- Use 9 or fewer top-level categories.[3]
- For mobile sites and applications, menu bars can be collapsed to a slide-out or expandable "hamburger" menu.
- Make sure that the menu items look interactive and have enough visual weight. They should invite action.[4]
- Make menu links big enough to be easily tapped or clicked. Provide enough padding and spacing around them so that users don't accidentally click the wrong one.[4]
- Choose consistency over "wow" factor.[4] Deviate from standards at your own risk.

Basic usage

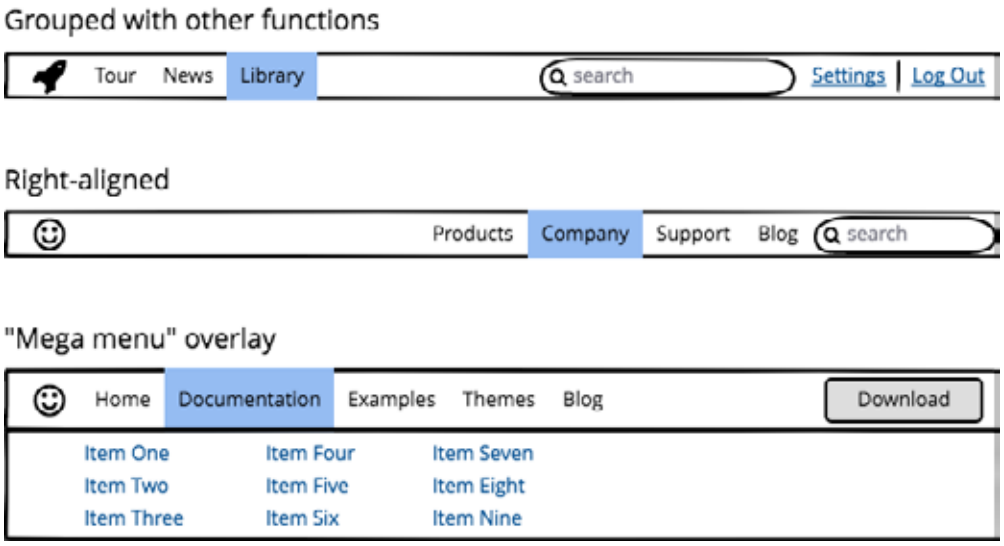
States



Like tabs and vertical navigation, menu bars should have a clear selected state, usually indicated by high-contrast text (or inverting the unselected style) and possibly making it bold.
A hover state should also be used.

Variations

To keep the header compact, other functions can be merged into the menu bar area, such as a search box or important actions (e.g., sign in/out). One term for this is "extended header".



For more complex hierarchies, a "mega menu" overlay can be used, showing multiple columns of sub-categories (or even an additional level of nesting). When using them, pay close attention to the implementation details, especially around the timing of when the menu is triggered and hidden.

Related controls

- Tabs
- Vertical Navigation

Further reading

- “About Face: The Essentials of Interaction Design”
- Mega Menus Work Well for Site Navigation (Nielsen Norman Group)

References

1. macOS Human Interface Guidelines
2. U.S. Web Design System
3. KDE Human Interface Guidelines
4. Nielsen Norman Group

Accordion Guidelines

Accordions are stacked containers with nested items that expand and collapse when clicked or tapped.

Accordions are an attractive control because **they allow a lot of links to be shown in a compact space**. But they can be less familiar or even intuitive, so be careful not to overuse them. They can be used either for navigation or for content, in contrast to vertical navigation.

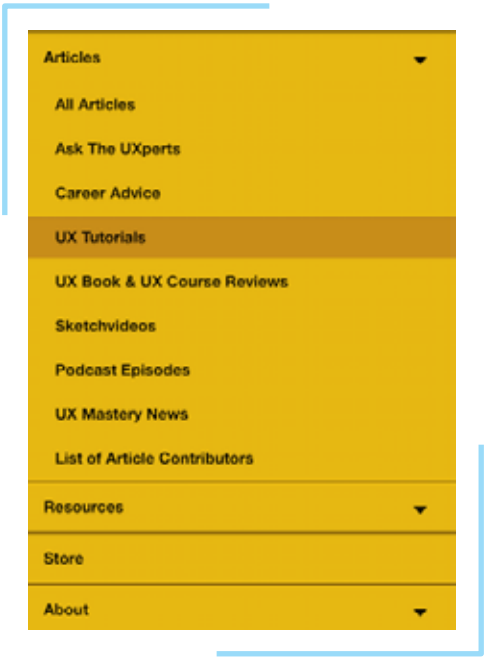
Other names include expansion panel and expand/collapse control.

When to use accordions

The book "Designing Web Interfaces" says that accordions are "good for collapsed modules of content. However, they should be used sparingly, as they have a strong visual style."

Part of that visual style frequently involves animating the panels as they open and close, which can seem like a "cool" effect, but often adds unnecessary interaction cost that can be avoided by using a simpler control. In that way they are similar to carousel controls, which have fallen out of favor for that reason.

While accordions are often *not* the best choice for desktop screens, they are **more likely to be appropriate on mobile**[1], like this mobile navigation menu from UX Mastery:

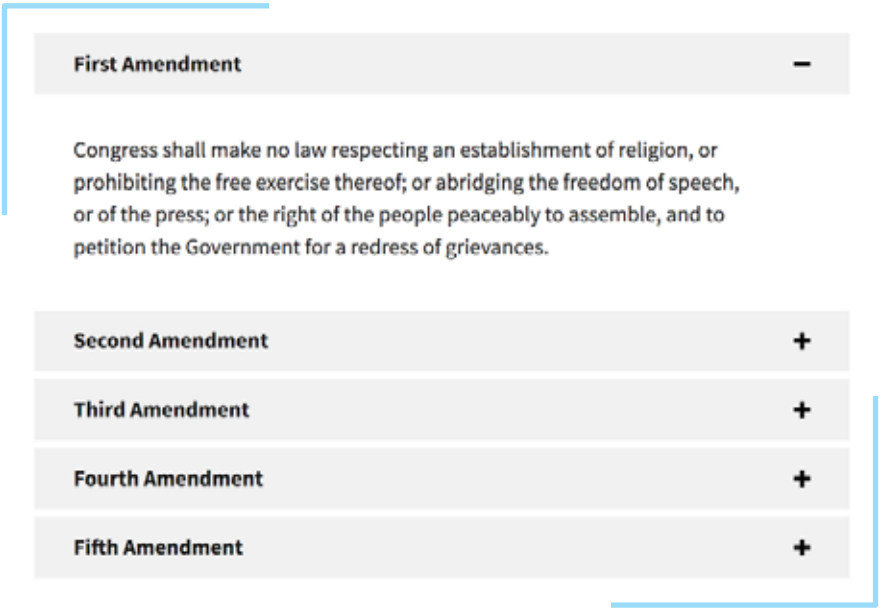


A good reason to use accordions for navigation is when the user may want to see the whole, as well as the parts, of the navigation hierarchy. One use case is where users may want to explore the site content before committing to a particular section or link. In that way, they are similar to hierarchical vertical navigation.

Accordion (😉 get it?) to the site UI Patterns, accordions should only be used for navigation when there are:

- More than 2 main sections on a website each with 2 or more subsections.
- Less than 10 main sections.
- Only have two levels to show in the main navigation.

As for using accordions for in-page content, the U.S. Web Design System suggests that if there is not a lot of content, or if users will want to see all content at once, *don't* use an accordion control. One valid use case example is a FAQ help page. However, don't use them *only* because there is a lot of content. The idea that desktop users don't like long pages is considered a myth by many.[2]



In all cases, avoid horizontal accordions (where panels are "stacked" horizontally). They are unfamiliar and almost always inferior to another kind of control.

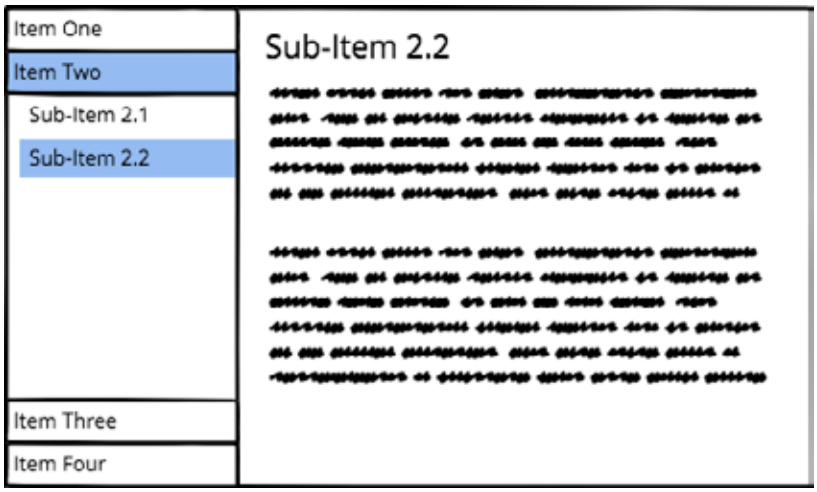
How to use accordions

The following are general guidelines that apply to most cases.

- When one panel is clicked it is expanded, while other panels are collapsed. Only one panel should be open at a time.[3]

- Don't open accordion containers on hover. Use click or touch behavior instead.[4]
- Allow users to click anywhere in the header area to expand or collapse the content; a larger target is easier to manipulate.[5]
- Start with the first section open. Don't default all to closed.
- Highlight the current panel so the user can distinguish open panel headers form closed panel headers.[6]
- If animating panels, "the animation should be subtle which means that it should last no more than 250ms."[6]
- Do not mix the accordion with other types of navigation on the same level.[7]
- If the content of a section won't fit on the page vertically, scroll within the panel rather than scrolling the entire control.[7]

Basic usage

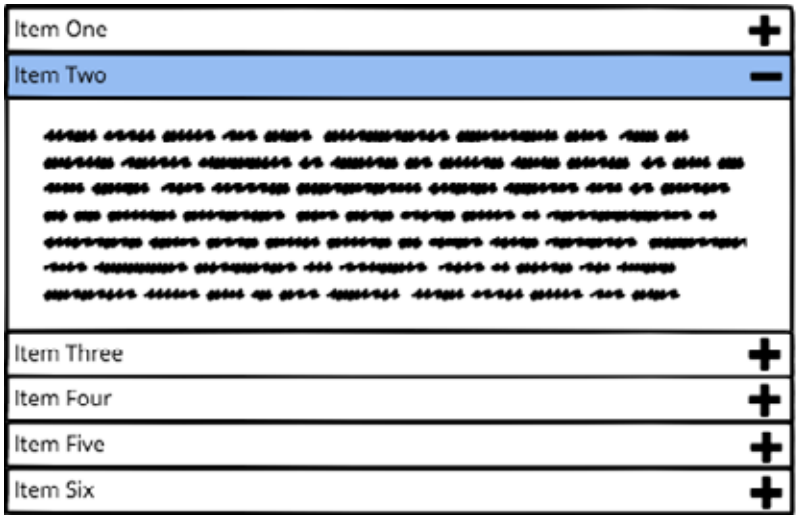


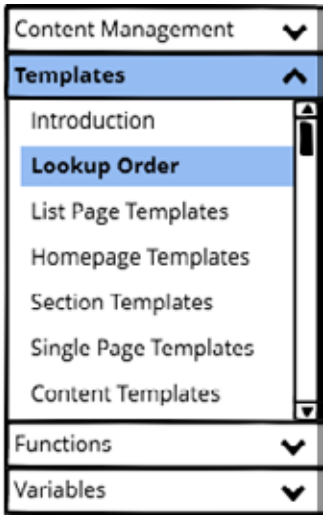
States

Accordions should have a clear selected state for the open panel and, if applicable, for the selected item within. A hover state should also be applied to both.

Variations

Variations should be minimized in order to keep consistent with standards. One common option is the addition of icons. The + and - icons are the most identifiable and familiar. Pointed triangles or chevrons can also be used.





The icon states should tell the user what will happen when they click.[8] I.e., the plus or expand icon should be shown on collapsed panels rather than indicating that an open panel has been expanded.

Related controls

- Tabs
- Vertical Navigation

Further reading

- "Designing Web Interfaces" by Bill Scott and Theresa Neil
- Interaction Cost (Nielsen Norman Group)
- Accordions Are Not Always the Answer for Complex Content on Desktops (Nielsen Norman Group)

References

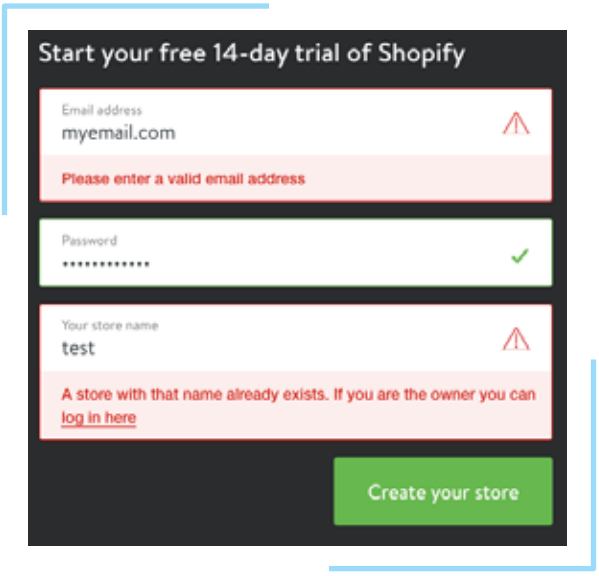
1. Nielsen Norman Group
2. Nielsen Norman Group
3. UI Patterns
4. Designing Web Interfaces
5. U.S. Web Design System
6. Welie.com
7. KDE Human Interface Guidelines
8. Smashing Magazine

Validation Guidelines

Validation is often used as a light-weight alternative to alerts. It is a great way to present feedback or guidance with limited interruption.

When to use validation

Validation is sometimes used in place of, or in addition to, alerts **to provide in-context error messages**. But it's not just for telling users what they have done wrong. It can also be used to tell users when they are doing something right, or to suggest ideas for improvement (such as password strength).

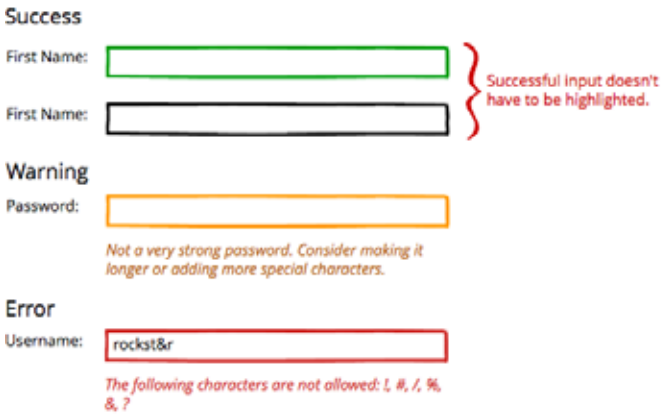


Validation is often used **to help users recover from errors**. But even better than providing good validation is preventing the need for it in the first place. Following text input guidelines can minimize the need for validation.

How to use validation

- Only show error validation messages or styles after a user has interacted with a particular field.[1]
- Try to validate “on-the-fly”, before the form is submitted. But if you can’t, consider adding a notification to summarize feedback at the top of the page when it reloads.
- Don’t clear invalid input data unless users aren’t able to correct errors easily. Doing so allows users to correct mistakes without starting over.[2]
- Provide guidance on how to fix any errors, don’t just tell users what they did wrong.
- Follow voice and tone guidelines, if you have them. (If you don’t have your own, you can take inspiration from these great examples of voice, tone, and content guides.)

Basic usage



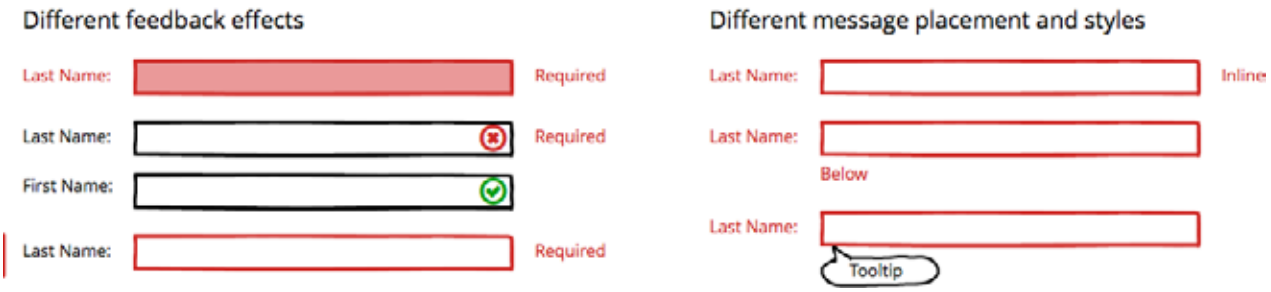
States

There are generally 3 states for validation components (shown above):

1. **Success** - Tells users that their entry or selection is valid. Not required, but can be helpful for new or novice users.
2. **Warning** - Usually indicates that an entry or selection is valid, but not recommended. A weak password, for example.
3. **Error** - The most common form of validation. Alerts users to an invalid entry and, preferably, suggests how they can fix it.

Variations

The primary way in which validation components vary is in their presentation or styling. Color is frequently used, although in different ways. It should not be the only indicator, however, because it may not be discernible by color blind users. An icon and/or help text is often used in addition.



Related controls

- Alerts
- Text Input

Further reading

- Preventing User Errors: Avoiding Unconscious Slips (Nielsen Norman Group)
- Voice, Tone & Content Guides (by John Moore Williams)
- “Web Form Design: Filling in the Blanks” by Luke Wroblewski

References

1. U.S. Web Design System
2. KDE Human Interface Guidelines

Tooltip Guidelines

Tooltips are a common form of contextual help that leverage the “details on demand” UX pattern.

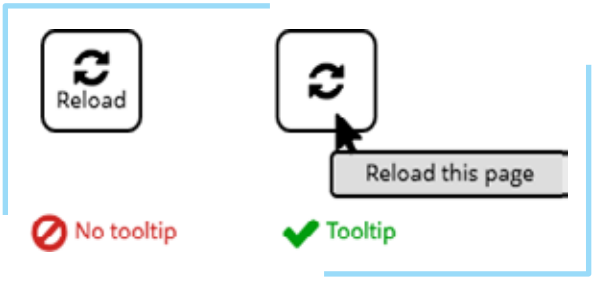
They are a great way to help novices without disrupting experienced users. A classic use case for tooltips is to show keyboard shortcuts when a user hovers over an action button.

In “About Face: The Essentials of Interaction Design”, Alan Cooper calls tooltips “one of the cleverest and most effective user-interface idioms we’ve ever seen.”

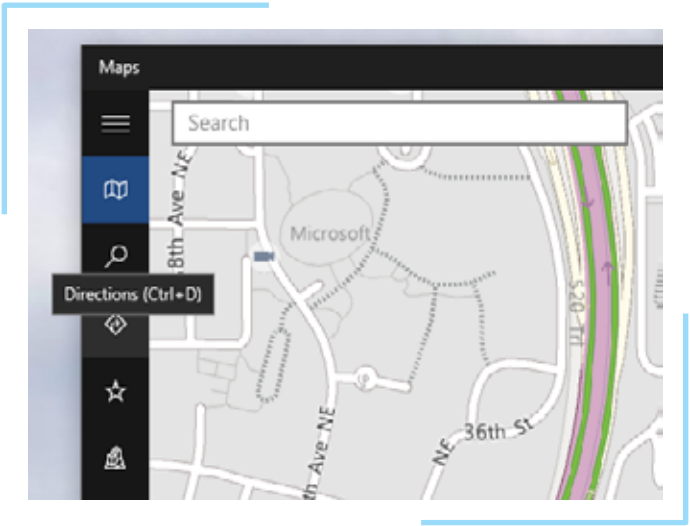
When to use tooltips

Tooltips are generally under-used, so when in doubt, use them (appropriately of course; the implementation details are critical, so make sure to read the next section).

The Microsoft UWP Guidelines have a nice rule of thumb for tooltips: **“if the information is available elsewhere in the same experience, you do not need a tooltip.”** This means that a button that contains an icon and text shouldn’t have a tooltip, but a button with an icon alone should, as in the example below.



Here is a typical tooltip example:



Tooltips are also handy when a longer explanation can be useful, but is impractical for space reasons. Another simple heuristic for using tooltips is when a control "benefits from a supplemental description or further information".[1] However, tooltips should never be used as a fallback for unclear icons or labels. Tooltips are supplementary information and should not be treated as a primary means of helping users understand the system.

Finally, tooltips are often not shown on mobile devices, so don't rely on them or assume that your user will read them.

How to use tooltips

- Display tooltips only as the result of user interaction, never display them on their own.[2]
- Only show plain text in a tooltip. Avoid formatted text or pictures.[3]
- Don't use the HTML "alt" attribute for tooltips. This should only be used as alternative text for accessibility purposes. Use the "title" attribute instead.
- Focus on the action a control initiates. Start with a verb to tell users what will happen when they click.[4]
- Keep them short. The macOS Human Interface Guidelines4 suggest a maximum of 60-75 characters.
- Tooltips should be placed near the object being hovered, but should never be placed in a way that interferes with what the user is doing by obscuring the object of interest.[1]
- If you want them to be available on mobile, consider adding small informational buttons for touch screen use (see variations below).[5]
- Timing is critical. If you have control over it, follow these guidelines:
    - Delay the start of the tooltip so that they aren't constantly popping up as the user moves their mouse across the screen. Wait until the user has stopped moving their cursor for about a second.[6]
    - Show the tooltip for about 10 seconds or until the pointer moves away from the control.[4]
    - Fade tooltips in and out over ~150ms.[3]

- Most operating systems and platforms have built-in tooltip controls; use them rather than defining your own.

Basic usage

States

Tooltip states are simple; they are either **on** or **off**. If they transition between those states, it should happen quickly (150ms or less).

Variations

Tooltip styles vary greatly, although functional variations should be limited, since users expect them to behave in a standard way.

Some simple variations include:

- Adding a directional arrow pointing to the object the tooltip is describing.
- Using light formatting or visual elements for additional information.
- Adding an explicit tooltip icon or button. This can be helpful for complex or novel interfaces where you expect users to need help, or for adding touch behavior on mobile.

Related controls

- Validation

Further reading

- HTML “alt” and “title” attributes.

References

1. UX Planet
2. Microsoft UWP Guidelines
3. Google Material Design guidelines
4. macOS Human Interface Guidelines
5. KDE Human Interface Guidelines
6. “About Face: The Essentials of Interaction Design”

Alert Guidelines

Any UI control that captures the user's attention can be thought of as an alert. They should be used wisely so they aren't overwhelming.

For the purposes of this guide, alerts are characterized by being interruptive and **requiring action to proceed**, unlike notifications or validation messages.

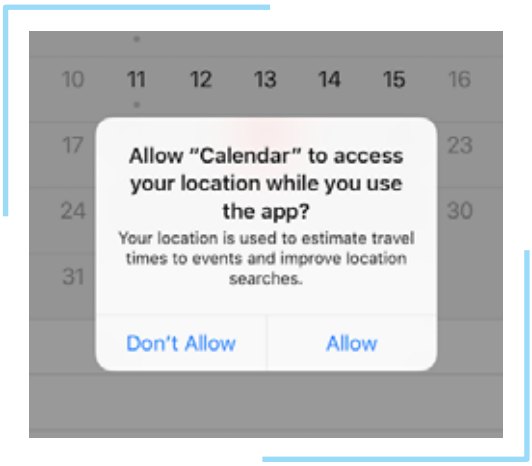
When to use alerts

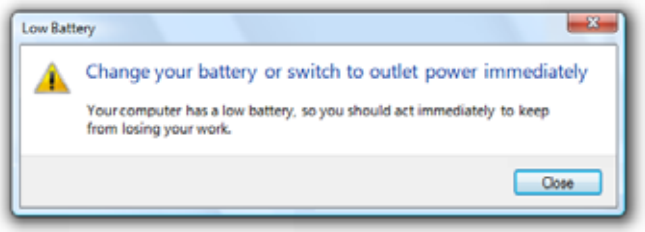
Every alert guideline says to **use alerts sparingly**. Overwhelming users with alerts dilutes their importance and annoys users. The Microsoft Windows Application Design Guidelines suggest when to use alerts this way: "Don't overwarn. Limit warnings to conditions that involve risk and are immediately relevant, actionable, not obvious, and infrequent. Otherwise, remove or rephrase the message."

It lists specific reasons for using alerts, such as:

- Awareness
- Error prevention
- Imminent problem
- Risky action confirmation
- Unintended consequence confirmations
- Clarifications

Here are some examples:





The U.S. Web Design System suggests that, when designing forms, consider using inline validation instead of (or in addition to) alerts.

How to use alerts

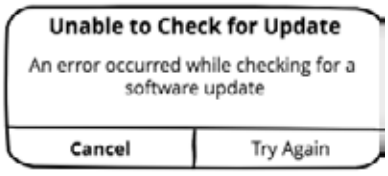
The macOS Human Interface Guidelines are succinct in their guiding principle: "When an alert is necessary, your most important job is to **explain the situation clearly and give users a way to handle it.**"

More specifically:

- Create specific, actionable, user-centered error messages. Users should either perform an action or change their behavior as the result of the message.[1]
- Use pre-defined or system styles when possible. Don't deviate from familiar.

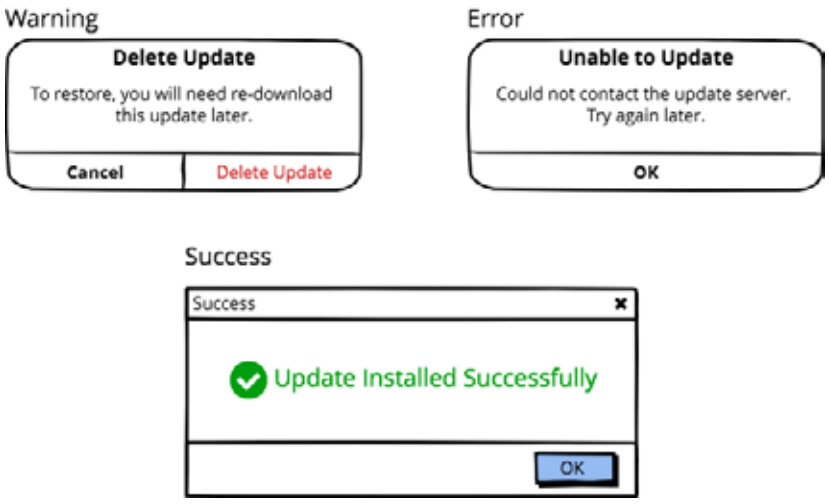
- Generally, use 2-button alerts (for dialogs overlays).[2]
- Express everything in the user's vocabulary. Use clear language, free from jargon.[3]
- The default button name should correspond to the action you describe. In particular, it's a good idea to avoid using OK for the default button.[3]
- They should be easy to dismiss when appropriate (e.g., responding to Escape key or the Close button in a dialog, in addition to an explicit dismiss button or link).
- Refer to your existing style and tone guide if you have one (you can find some inspiration in these voice, tone, and content guides, if you don't). As an example, the Microsoft Windows Style and Tone guidelines writes this about using "please" and "sorry":
  - "Use please whenever its absence would be considered curt"
  - "Use sorry only in error messages that result in serious problems for the user"

Basic usage



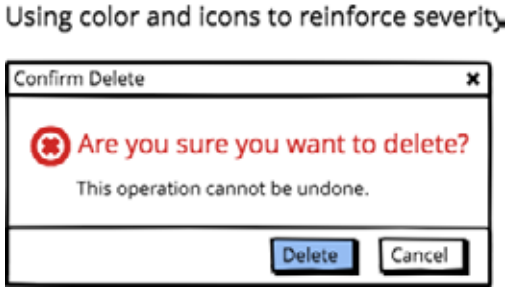
States

Similar to validation messages, alerts can be used to **communicate errors or provide warnings** about potentially destructive actions. Unlike validation, however, alerts should generally not be used for success messages, unless it is to notify that an important action has completed.



Variations

As shown above, alerts vary by operating system and platform. Most software platforms provide built-in alert components. It is best to use default system styles for alerts. Some operating systems allow for icons and/or color to differentiate states or types of alerts.



Related controls

- Validation

Further reading

- Voice, Tone & Content Guides (by John Moore Williams)
- Microsoft Windows Style and Tone guidelines

References

1. KDE Human Interface Guidelines
2. iOS Human Interface Guidelines
3. macOS Human Interface Guidelines

Data Table Guidelines

Data tables, also called table views, tables, and data grids use columns and rows to display related information in a grid.

this	is	a	very
basic	HTML	data	table

Tables are readable and familiar when designed appropriately.

For very simple tables the guidelines are easy to follow. The challenge comes as the data becomes too much to parse at a glance. When you start adding ways to filter, sort, search, and act on the data the waters get a lot muddier.

When to use data tables

Data tables are best used for **numerical data and lists of objects of the same type**.

Here is a typical numerical data table:

PASSING STATS													
SEASON	TEAM	GP	CMP	ATT	CMP%	YDS	AVG	TD	LNG	INT	FUM	QBR	RAT
2005	GB	3	9	16	56.3	65	4.06	0	16	1	2	--	39.8
2006	GB	2	6	15	40.0	46	3.07	0	16	0	1	7.8	48.2
2007	GB	2	20	28	71.4	218	7.79	1	43	0	0	78.0	106.0
2008	GB	16	341	536	63.6	4,038	7.53	28	71	13	6	64.4	93.8
2009	GB	16	350	541	64.7	4,434	8.20	30	83	7	8	70.7	103.2
2010	GB	15	312	475	65.7	3,922	8.26	28	86	11	2	69.2	101.2
2011	GB	15	343	502	68.3	4,643	9.25	45	93	6	4	84.5	122.5
2012	GB	16	371	552	67.2	4,295	7.78	39	73	8	5	71.2	108.0
2013	GB	9	193	290	66.6	2,536	8.75	17	83	6	3	60.6	104.9
2014	GB	16	341	520	65.6	4,381	8.43	38	80	5	7	78.3	112.2
2015	GB	16	347	572	60.7	3,821	6.68	31	65	8	8	60.3	92.7
2016	GB	16	401	610	65.7	4,428	7.26	40	66	7	4	73.8	104.2
2017	GB	6	128	193	66.3	1,385	7.18	13	72	3	1	62.6	103.2
Career		148	3,162	4,850	65.2	38,212	7.88	310	93	75	51	--	104.1

Tables, like the one below, with actions, links, and buttons are common in enterprise, CRM, and other business applications:

10 items Download Upload

Column One	Column Two	Column Three	Column Four	Column Five	Column Six
Alexander Stokes	as8du0as65sl	9	Approved	09/15/2016	09/15/2016
Tom Jenkins	983ukasdh283	15	Denied	11/06/2016	Resubmit
Floyd Shaw	asdkjjkajhsd86	22	Denied	06/27/2016	Resubmit
Garrett Davis	asdHJKH8346	35	Approved	12/15/2016	09/11/2016
Maud Banks	asdhjajks6753	12	Denied	01/08/2016	Resubmit
Dominic Cobb	asdjhjak67a5s	22	Denied	10/06/2016	Resubmit
Curtis Holmes	asdjhha86725	17	Approved	01/20/2016	09/08/2016
Etta Guzman	asdhja862362	30	Approved	08/24/2016	09/01/2016
Gilbert Olson	fkljsdkh82638	15	Approved	07/03/2016	08/28/2016
Francisco Holloway	askhdja672352	42	Approved	05/27/2016	08/15/2016

FIRST PREV 1 NEXT LAST Sho 10

The Google Material Design guidelines suggest that tables should only be used when there are **3 or more columns**. Side-by-side lists or any other text control can be used for a simple 2-column comparison.

Apple mentions that an outline (a.k.a. Tree) view should be used instead of a table view for hierarchical data. One variation for this scenario is a tree table (described below).

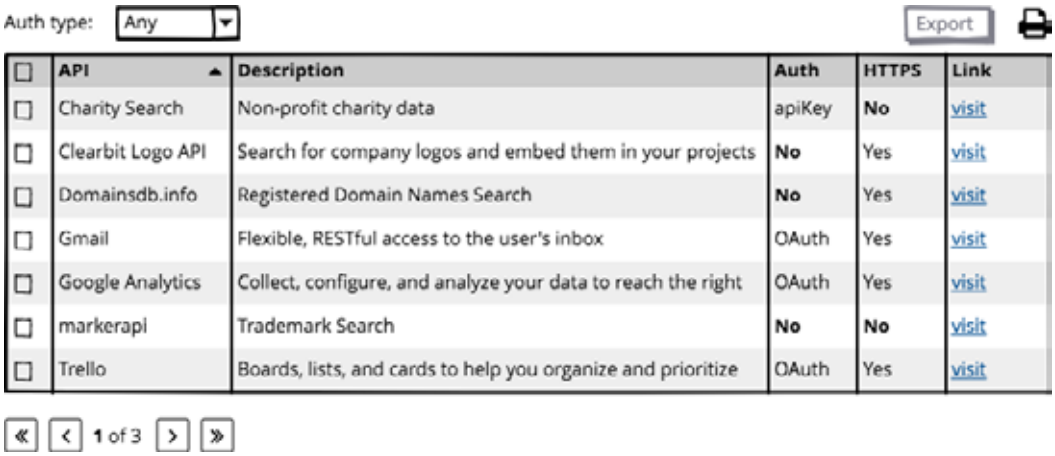
How to use data tables

- Use a header row with descriptive titles.[1]
- Let people click column headings to sort a table view if it provides value.[1]
- Clicking once on a column heading should apply "natural" sort order (e.g., alphabetical, smallest number first, earliest date first, checked items first) and show a *down* arrow. Clicking again should reverse the order and show an up arrow.[2]
- Alternate row colors for large tables.[1] This is often referred to as "zebra striping".
- Checkboxes should accompany each row if the user needs to select or manipulate data.[3]
- For actions that can only be applied to one row at a time (e.g., edit, view details), standard practice is to provide a link or icon in the last column of the table.
- Use the <THEAD> and <TH> HTML tags for header rows for accessibility and easier visual styling.
- Left-align text and right-align numbers.

- If you have numeric data, keep the level of precision appropriate. The fewer decimals, the less time it takes to scan and understand the data.
- Show only the information that users really need to see, but provide the ability to dig deeper into details if needed. Use Inlays, Overlays, and tooltips for showing details on the same page with the table to maintain user's flow.
- Consider using floating (i.e., "sticky") header rows for long tables.
- Provide the ability to search within long tables.[2]
- Pay attention to display density. Having the data too close together makes it hard to read, yet using too much spacing means more scrolling (and more time to find what the reader is looking for). You can give users the option to change the display density (as shown here), but don't use this as an excuse not to pick a good default.
- Pagination is useful for large data sets, but don't immediately assume that it's needed. Studies have shown that users don't mind scrolling[4] in many cases and showing all the information on one page means that users can search and sort more easily to find what they're looking for.
- If you use pagination, make sure to show the total number of pages or results. The ability to choose how many rows to show on a page is also useful.
- Actions that users can perform on the data should be placed around the edges of the table (as shown in the example below). Pagination is usually placed at the bottom, while filtering, sorting, and multi-row actions are often placed in the top-left and/or top-right.

- If you provide a way to filter or otherwise limit the data, make sure to clearly indicate that a subset of the data is being shown.
- Use color and decoration sparingly. It can be useful to highlight unexpected data (such as extreme/outlying values or failed actions), but can detract from overall readability.[5]

Basic usage



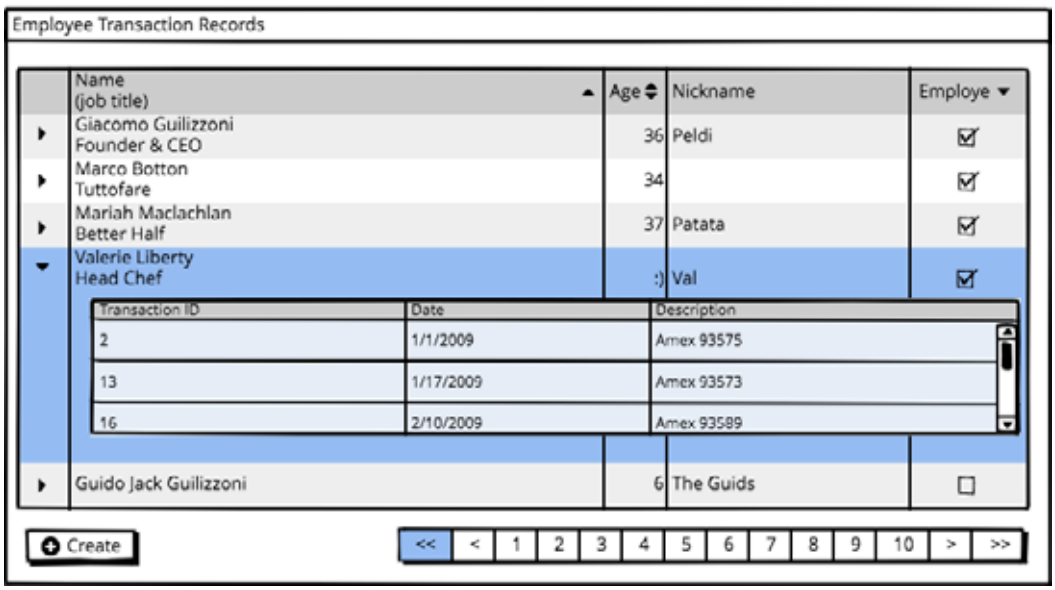
States

Tables don't have states like other UI controls, but some have cells which can be changed from read-only to editable. In the case of editable cells, they should be visually distinct through a lowered bevel or thicker border.[6]

Variations

One common variation is the expandable hybrid table/tree view. This is used to avoid forcing the user to visit a new page and losing context. See the further reading section below for more table variation examples.

Tree Table



These details-on-demand variations should be reserved for complex data tables. It is overkill for cases where the user is primarily browsing. A better choice is to carefully consider which information the user needs to see and provide only that data.

Further reading

- <THEAD> and <TH> HTML tags
- Design better data tables by Andrew Coyle
- Web Typography: Designing Tables to be Read, Not Looked At (A List Apart)
- Table Filter Design Pattern

References

1. macOS Human Interface Guidelines
2. GNOME Human Interface Guidelines
3. Google Material Design guidelines
4. UX Myths
5. A List Apart
6. KDE Human Interface Guidelines

Icon Guidelines

Icons are everywhere, both in software and outside of it. The power of an image helps users identify things quickly and accurately.

The KDE visual design group say that icons “convey meaning that users perceive almost instantaneously.” They can be also useful for internationalization and when concepts are hard to describe in words.

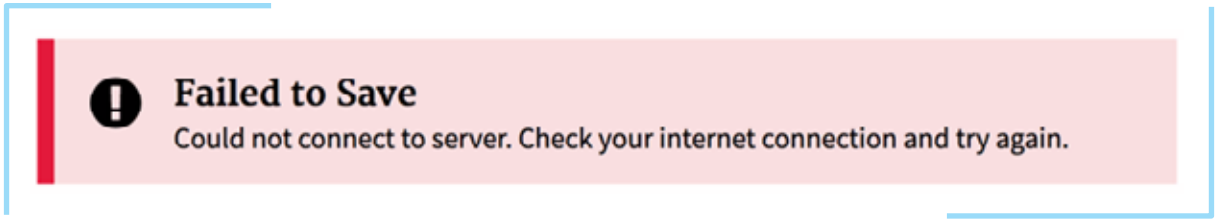
When to use icons

It is rare that icon use is actively discouraged. The biggest danger when using icons is the use of ambiguous or unclear icons, which can either mislead the user or conflict with their adjacent label, resulting in a slower and/or more frustrating experience with your product. In short, bad icons are costly.

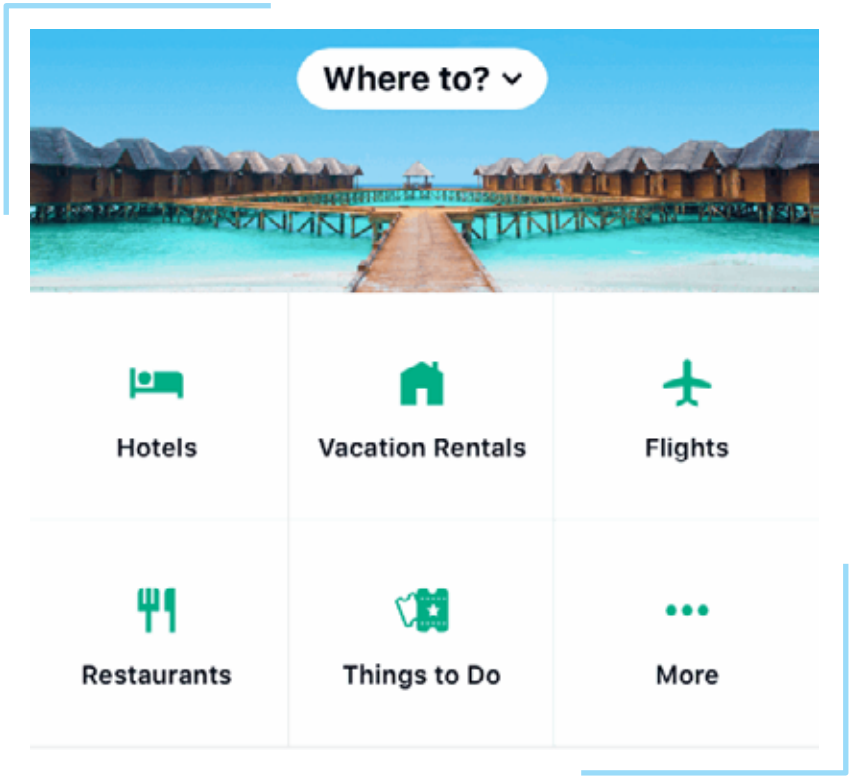
The GNOME developer guidelines state that choosing the correct icon for each purpose is “vital to making sure that your application is usable.” They go on to encourage their use, though, calling them “an essential part of any application” and “a crucial part of its identity.”

Icons are also a great way to provide redundancy, especially for important messages.

In the following example, there are 3 ways that the error state is conveyed: the text itself, the color, and the icon. The icon reinforces the severity of the message.



When used correctly, icons can speed up a user's interaction with your product, enhance its usability, and reinforce your brand identity.



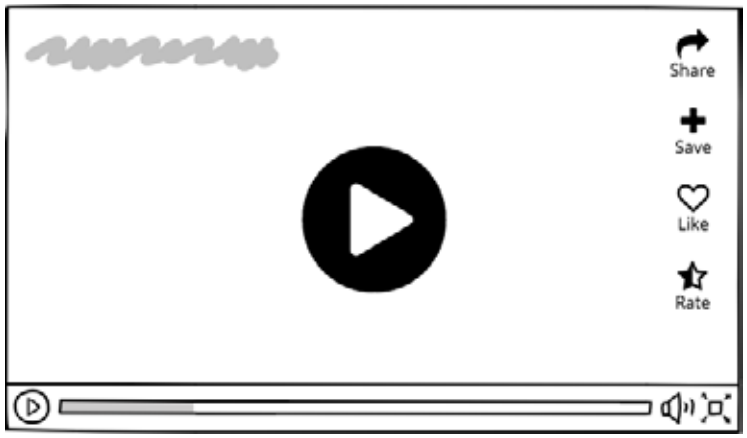
How to use icons

Note: The Google Material Design Guidelines differentiate between **product icons** and **system icons**. Product icons are primarily for branding and visual identity purposes. An example of a product icon is an application icon on the task bar or home screen. The guidelines below apply mostly to system icons, which are identifiers for actions or commands.

- Adjust the degree of abstractness according to familiarity of the metaphor.[1]
- Apply metaphors only once (e.g. do not use a brush twice for different options).[1]
- Use arrows only if they can easily be related to spatial features such as Previous/Next in a sequence or Up/Down in a hierarchy. Avoid using arrows metaphorically (such as for Reply/Forward or Undo/Redo).[1]
- Use consistent rules for placing text next to icons.[1]
- Icons that lose details when shrunk may need a special version that preserves meaning even at smaller sizes. Critical details may become unrecognizable when scaled down.[1]
- Avoid using text in icon designs; it may not scale well to smaller sizes.[1]
- Follow conventions. Don't create new metaphors when familiar ones exist.[2]

- Always accompany icons with text labels unless the icon metaphors are nearly universally recognized (e.g., home, print, search), which is rare.[3]
- Don't use icons that are commonly associated with a different meaning.[4]
- Establish color and design rules for icons and apply them consistently. See the Google Material Design guidelines for a good example.
- Apply tooltips for additional information or when labels aren't used.

Basic usage



States

Icons often have "on" and "off" variants that are used to indicate states when used as buttons. A common example is switching between "liking" and "unliking" something. Badges can also be overlaid on top of icons to indicate a more complex state.[5]

Variations

Aside from a simpler version of icons, called badges[5], most variations in icons come from their look and feel. Some are monochromatic and simple, others are bright and intricate. They can use a "filled" or "outline/hollow" style.[6] The most important thing is that they are consistently styled and quick to grasp.

Related controls

- Buttons
- Tooltips

Further reading

- Icon Classification: Resemblance, Reference, and Arbitrary Icons (Nielsen Norman Group)

References

1. KDE Human Interface Guidelines
2. GNOME Human Interface Guidelines
3. Nielsen Norman Group
4. Nielsen Norman Group
5. Microsoft UWP Guidelines
6. Todd Wolfson

3 Intro to UI Design Patterns

Intro to UI Design Patterns

A design pattern is a reusable solution to a commonly occurring problem. As a recipe in cooking provides the ingredients and structure that make up a recognizable dish, so too does a design pattern provide an identifiable and predictable solution to an interface design problem.

The idea of design patterns initially came from architecture and programming, where the idea was to optimize solutions that are known to work well within given contexts.[1] Solutions that emerged frequently enough became recognized as **a formula that can be reused**.

Structural and behavioral features of a pattern are familiar to users. Your team can leverage this knowledge, rather than reinventing the wheel, to provide greater ease and use of their product. It's good to point out, however, that while design patterns are useful for informing design decisions around your particular problem, you may likely need to modify them around your users' and business' needs.

We're going to start by taking a look at a design pattern, look at a few examples of the pattern in use, and deconstruct their implementation. Then we'll list some common patterns for interface design and you may explore them in depth.

The Cooking Analogy:
In cooking we combine ingredients to prepare a dish. Let's say for instance, we're planning to make a fish taco meal. If you're familiar with this dish, you know that you'll usually prepare it with a flaky fish like Cod perhaps, tortillas, different seasonings, oil, salsa, and maybe sliced lime for garnish. There are different ways to add to this dish to make it yours, but the basic combination of ingredients make a fish taco pretty unmistakable.

This is very much like using design patterns. We have a general model for how to create this dish to make it recognizable. We can add or subtract from those ingredients and how they're put together to make it unique.

Elements of a design pattern

Design patterns are typically written with a common set of attributes that looks something like this:

Design pattern model

- **Pattern Name (and description)**
  - A label that provides a clear way to communicate and reference this pattern, particularly when discussing it with colleagues.

- **Problem**
  - Describe what problem this solves and why this pattern exists.
- **Context**
  - Describe when to use this solution.
- **Solution**
  - How it works.
  - Describe the solution in detail.
- **Recommendations**
  - Provide further recommendations.
- **Examples**
  - Provide further recommendations.

Design pattern model

Let's explore writing a design pattern for a website Shopping Cart component into this model. This seems like an obvious description of a very familiar component.

While you're reading, think about how this compares to other purchasing experiences like a one-click purchase, or how it compares to a similar purchase of a service like a reservation or booking experience. Think about how this might be different on a mobile phone for example.

Pattern Name: Shopping Cart for E-commerce Site

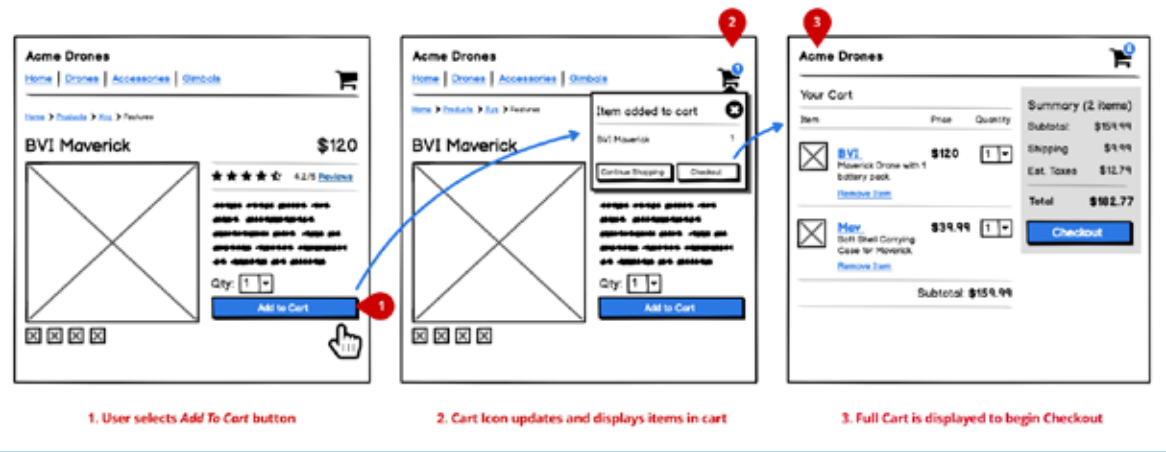
Description: The Shopping Cart component consists of 1) An "Add to Cart" button to purchase an item and 2) an accompanying cart icon used to indicate that the item is held for purchase and provides a link to view the items and begin checkout.

Problem: Users want to purchase an item in an e-commerce site.

Context of Use: Use this pattern when an online store allows browsing items, has more than a single item to purchase, or requires review of order before completing purchase.

While shopping on an online store the user may select items to purchase, but want to continue browsing, and may want to review and edit what they've selected before beginning checkout. This is similar to holding items in a shopping cart in the physical world.

Solution:



1. Viewing Product and Adding an Item to Cart

1. Present a button with the product to add the item to a shopping cart.

2. Updating and Previewing Cart

1. When the user has selected the Add to Cart action, provide feedback that the item has been added. Show the item count increase in a numeric indicator next to the cart icon.
2. Optionally show a preview of the cart with the item and the selected options displayed.
3. Provide feedback about the next steps, for example editing the cart, viewing the full cart, continuing shopping, or beginning checkout to complete the purchase.

3. Displaying the Shopping Cart

1. Provide a separate Shopping Cart View of the items added to cart so that the user may modify or complete their order.
2. Provide actions for editing quantities, removing items.
3. Provide an action for “checking out” or completing their purchase.

Recommendations:

- Add to Cart may be presented with options, for example a selector for quantity, selectors for style, etc. Provide conditional logic if the item requires options to be selected, for disable the Add to Cart button if style or size has not been selected.
- Consider a one-click option for signed in users, or navigate directly to the checkout process if the store offers a single item for purchase.

Examples:

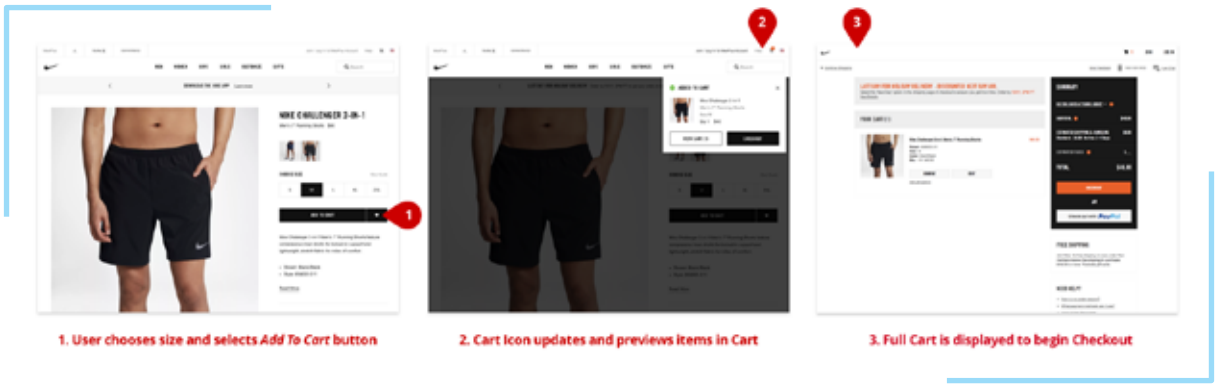
- Nike
- Shopify
- Crate and Barrel

Examples

Let's look at 2 of the Shopping Cart examples listed above and go a little further to deconstruct how they solved this specific need and how this reflects the pattern.

We'll look at the point where a customer decides to purchase a product using the Shopping Cart metaphor in 2 sites: the Nike Store and Tatt.ly, a site that uses Shopify.

Nike.com cart

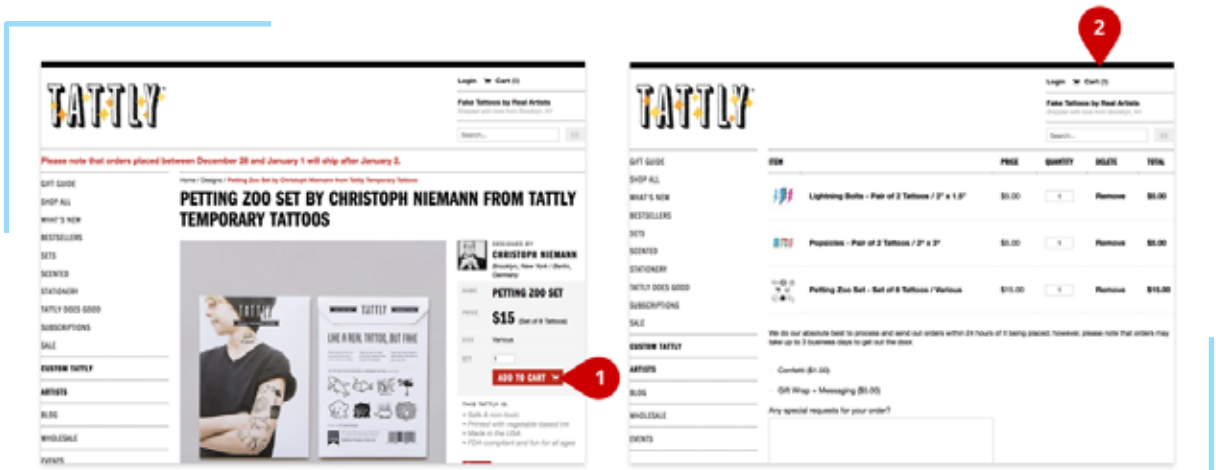


1. User chooses size and selects *Add To Cart* button

2. Cart Icon updates and previews items in Cart

3. Full Cart is displayed to begin Checkout

Tattly cart, which uses Shopify for e-commerce



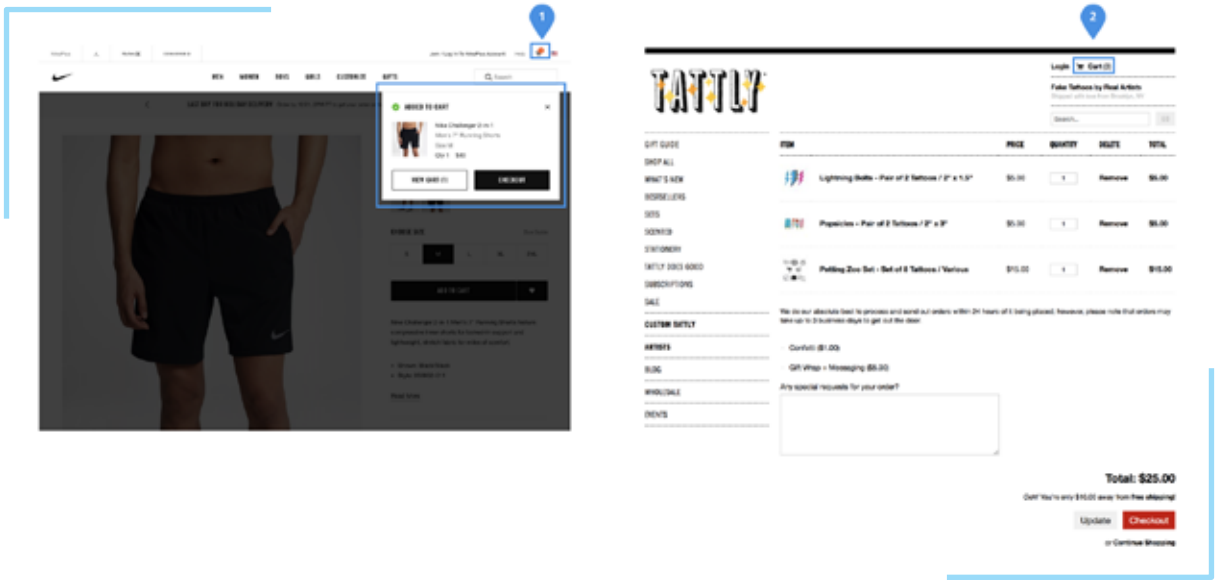
1. User selects *Add To Cart* button

2. Cart Icon updates and displays Cart to begin Checkout

Comparison

In both of these examples, the design pattern is based on using the shopping cart as a metaphor for temporarily holding things you are going to purchase.

There's a common iconography that's been established, there's usually an indicator of the number of items being held, and an assumption that the next step is to "check out" as one would do in the physical world at a store.



There are only subtle variations between these experiences.

1. Nike provides a nice behavior to preview the cart without leaving the product page, perhaps making it easier for the user to return to shopping. In another site with a similar dialog to Nike's Crate and Barrel uses this same idea of previewing the Cart, but additionally promotes other products the customer might want to purchase based on what's in their cart. Nike promotes other products on the cart view itself.
2. Shopify directs the user to the Cart immediately after adding a product to cart.

In both Nike and Tatt.ly (Shopify), there's the same general behavior and structure of the experience. Instead of re-creating the experience, for the most part, both of these sites rely on predictable conventions, which make up the common shopping cart pattern.

Using and creating your own patterns

It's likely that many of the interfaces you see were probably designed with a common design pattern in mind. The Cart example is one that is copied often because it's based on an understanding leveraging the recognizability of this pattern. It puts users at ease, because they have expectations for how online shopping works, and it satisfies business needs to make the experience as frictionless as possible, while also looking for valuable add on sale opportunities.

As you begin to use design patterns, remember that while they are great for helping you inform your design decisions when solving a common UI problem, they're not meant to be copied without thinking about your users' and product's particular needs.

Jennifer Tidwell, who wrote an excellent interface design book titled *Designing Interfaces*, gave this word of advice about using patterns in the section called "About Patterns."

"**They aren't off-the-shelf components;** each implementation of a pattern is a little different from every other. They aren't simple rules or heuristics either. And they won't walk you through an entire set of design decisions..."

You'll find many potential solutions for your own design problem in our Design Pattern and User Interface Galleries. You can also find pre-built interfaces for many of these patterns in Wireframes to Go.

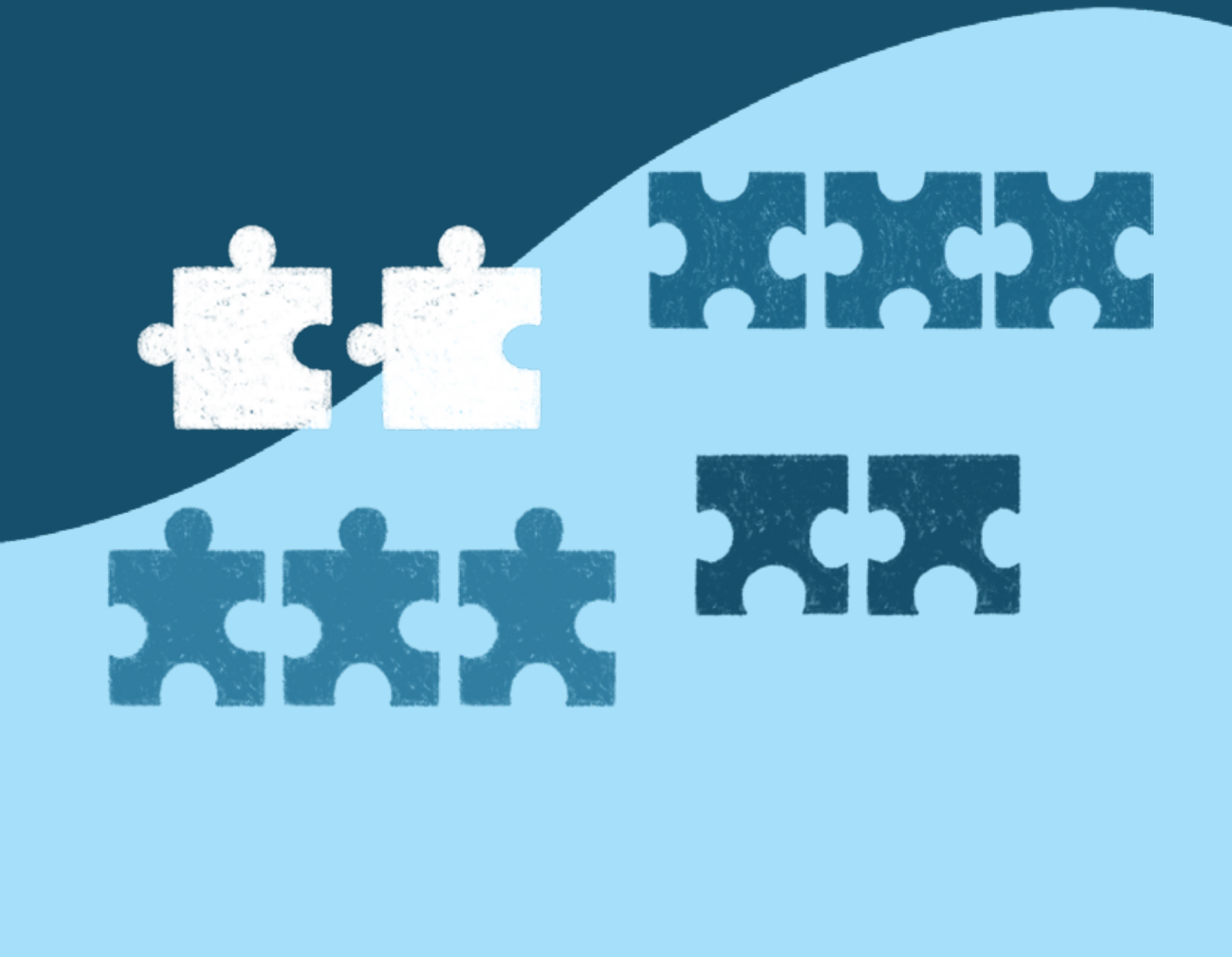
Further reading

- "Elements of a Design Pattern" by Jared Spool

References

1: More about the origins of design patterns: A design pattern in architecture and computer science is a formal way of documenting a solution to a design problem in a particular field of expertise. The idea was introduced by the architect Christopher Alexander in the field of architecture, and has been adapted for various other disciplines, including computer science. An organized collection of design patterns that relate to a particular field is called a pattern language.

4 Visual Design Principles



Visual Design Principles

Design principles are foundational rules that work well for communicating information so that the user can do something with it, whether it's making a decision, taking action, or just making sense of what's being conveyed. Like patterns, they emerge because they have proven to be optimal attributes for the purpose they serve, in this case for communicating effectively.

The design principles we'll be covering are:

1. Contrast
2. Hierarchy
3. Proximity
4. Alignment

Until you familiarize yourself with them, these principles may seem new to you. By labeling and understanding what they are, they become under your control so you can use them. With practice using them will be second nature, and they'll be part of the language you use to communicate information in your interfaces. You won't be able to look around without thinking about them.

The Cooking Analogy:
Imagine that you ordered a dish of fish tacos. But when it arrives at your table, you get a tortilla soaked in oil, your fish is dry, overcooked and served on a separate plate, and spices come in a separate bowl. Sounds like something you'd send back.

Separate ingredients don't usually make an interesting dish by themselves. The selection, preparation and arrangement of parts is what gives those parts form as the dish. The techniques and style of presentation can also make the dish more appetizing — this part is what makes the dish really become an experience.

In design we have something like that too, and it's based on somewhat universal principles that make communication with design effective.[1]

We're going to consider some **basic visual design principles that are helpful for designing interfaces**.

We've selected 4 principles that you'll get the most benefit from as you begin your practice in interface design. There are many more beyond this list, but these will give you a good foundation for making your wireframes more effective.

Why use design principles?

The use of design principles may be unnoticeable at first, when we're looking at screens or pages in the world. You may, however, start to notice when some of these design principles have not been applied. The next time you find it hard to make it through a big form or can't find a button among a sea of controls in a cluttered application, think about how applying these design principles might have helped.

Well-considered design tends to disappear when it serves its purpose in optimizing communication or use. It **gets out of the way and gets users to goals faster**.

While you may not be doing the ultimate visual design for your product, some familiarity with design principles goes a long way toward communicating your ideas. We may deal less with the rendering of the final interface surface, but much of the understanding of what we want to communicate comes from what we explore in interaction and information in our wireframes.

Now let's look at each principle and learn when and how to use them to communicate in our interface wireframes.

Contrast

Contrast refers to **the differences between things**.

There are various ways to achieve contrast. A common way of adding contrast is to make things very different in size. Another way is to use contrasting colors.

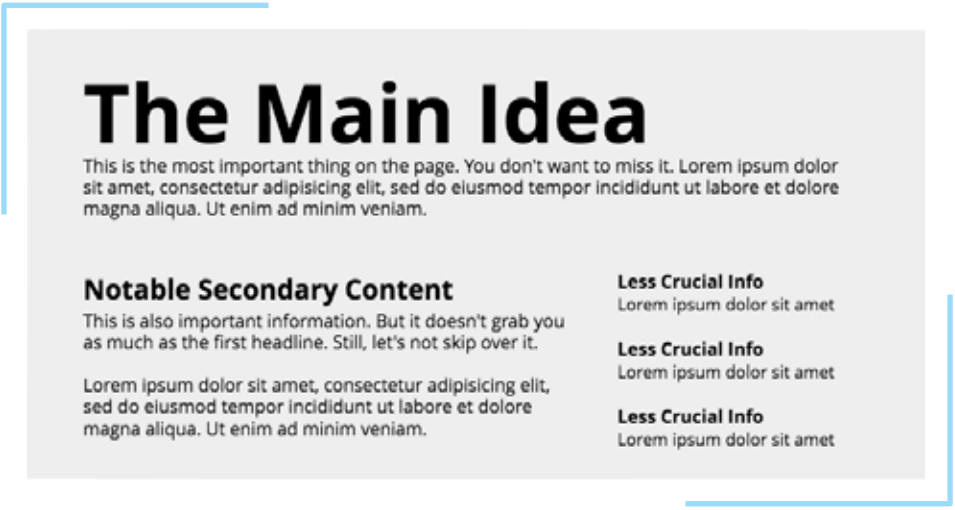
When to use contrast

In interface design we use contrast **to organize information and direct the eye**.

A wall of text with no contrasting elements to break up the flow can be tiresome to read.

Contrasting font sizes and styles can indicate a heading that precedes a section of subordinate text. It breaks up the flow and gives the reader visual cues to pause and shift their focus and allows them to process the information in chunks.

The example below might be a layout for a website homepage. Notice how typography helps add contrast in the headings.

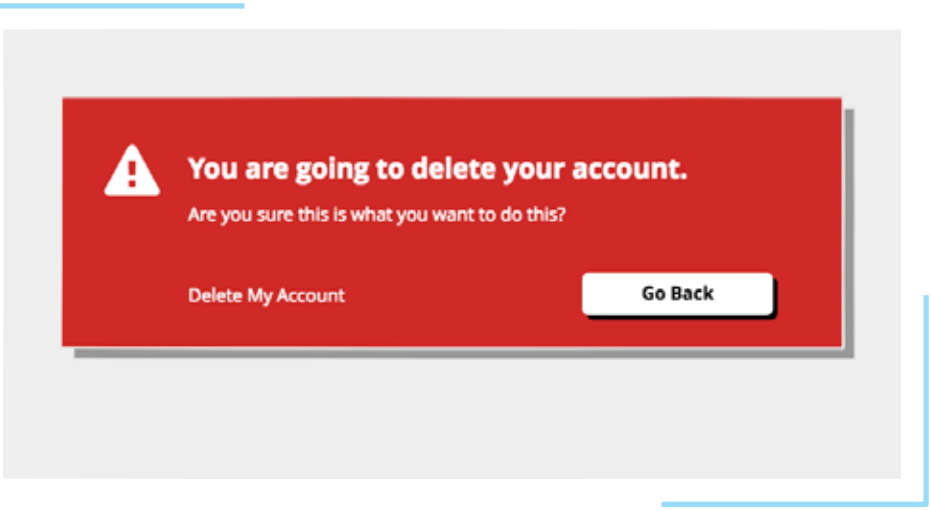


You'll find examples like this in many traditional Newspapers. This is also an example of visual hierarchy, which we'll talk more about below.

We also use contrast **to indicate important information**.

Using different typography or colors can alert users when a warning or crucial message demands attention.

A red text block, for example, might make a user pause before a destructive action. This red dialog contrasts highly against the pale gray background.



You'll find examples like this in application design, as with the warning that the Google Chrome browser shows if you are about to enter a site that contains malware.

Note how well the white text works against the red background. That's because the white also contrasts highly against the red background.

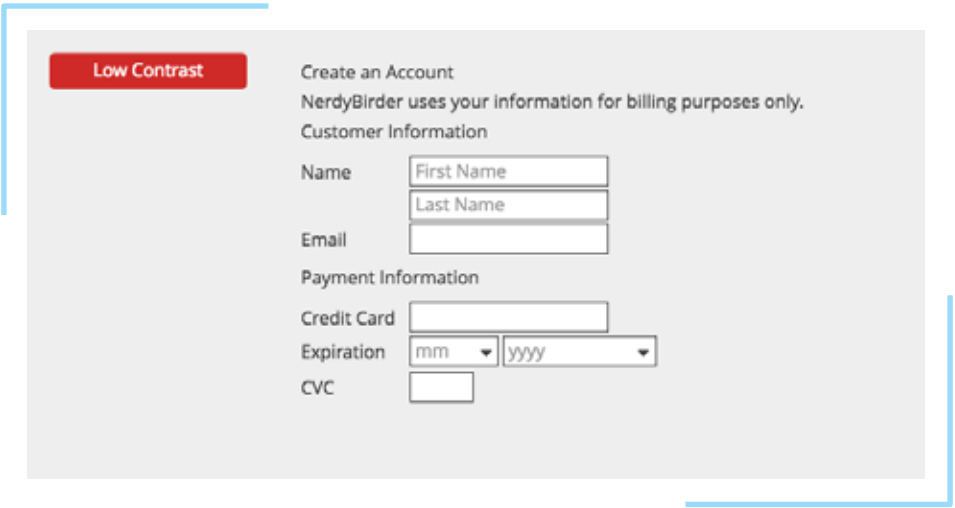
There are some caveats when using color. These are less of a concern when it comes to wireframes, but be aware of some of the issues below as you move on to visual design.

- Be careful not to use colors that don't contrast enough, because they may be difficult to read.
- Be sure to consider cultural norms when it comes to color.
- Colors have different meanings in different cultures.
- Also be aware that users who have experience color deficiency in their eyesight may also have difficulty recognizing certain colors.

How to use contrast

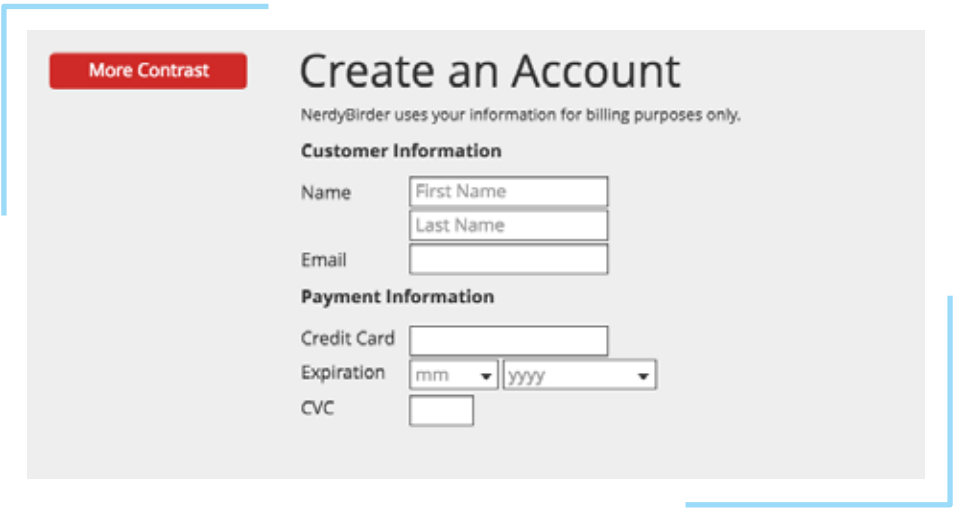
Contrast works best when things are very different from each other.

In the form below, all of the text is too similar. The headings and labels show little contrast and it's subsequently more work for the user to parse the text to determine where one piece of information ends and another begins.



We can add more contrast so that the pieces of information contrasted with each other.

By making the title and headings stand out visually using scale and weight, they can be read at a glance, before continuing to the inputs below. This helps the user get through the information more easily.



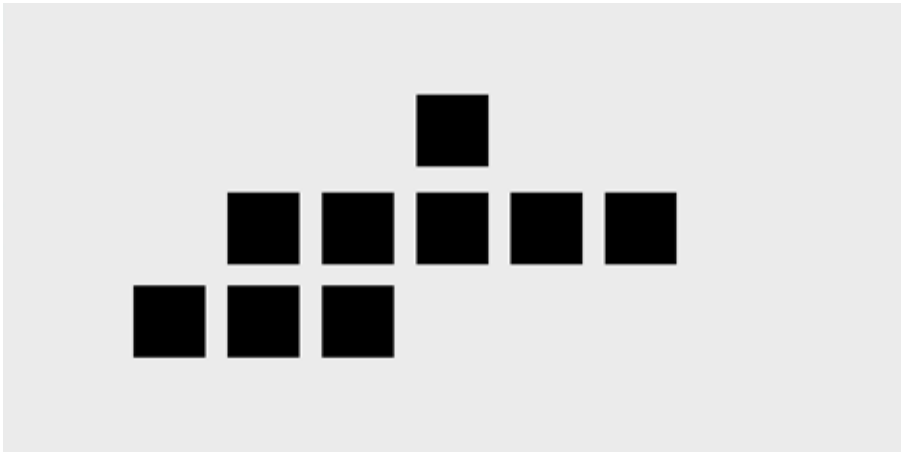
Certain controls may help you to easily use contrasting text. In Balsamiq, a Title control is 40px by default and a text block, useful for body text, takes your default project setting (around 13 - 16px).

Remembering to use controls that are intentionally made for specific contexts like these will get you in the practice of using contrast effectively.

Hierarchy

Hierarchy refers to a group of things where **items are rank ordered by importance.**

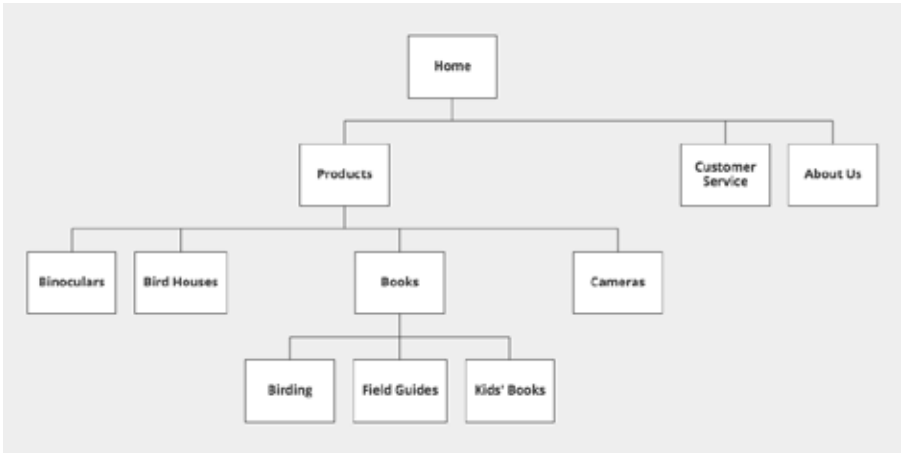
There are various ways to display hierarchy in a system. Typical examples may orient information from most important to least important visually, as in this top-down display of a hierarchical tree. You see this often in Organizational Charts or Site Maps.



When to use hierarchy

We use hierarchy **to organize information in a system.**

Site maps like the tree representation below are typical of how we use hierarchy when organizing screens in website or application design.



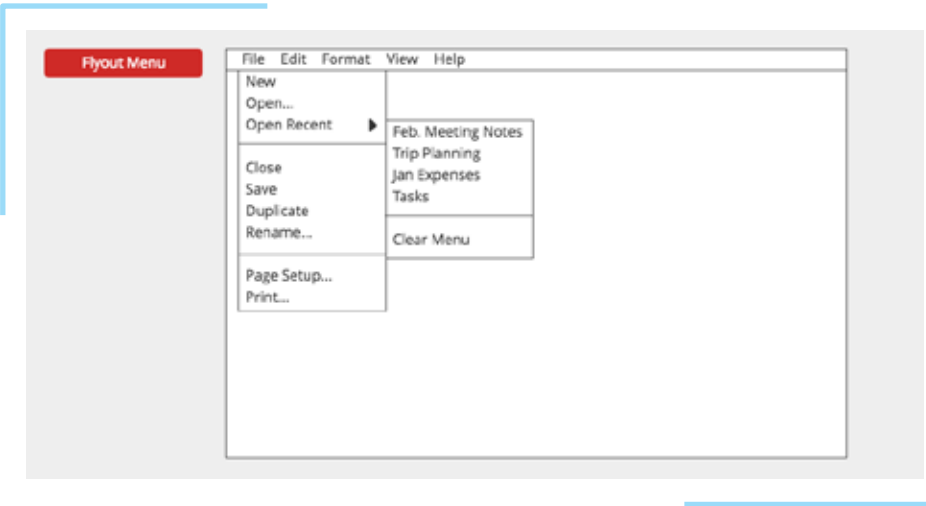
There can also be a hierarchy within individual screens.

One relevant use of hierarchy is also shown above in our contrasting text examples. We used size and scale to draw attention to the most important information on the screen, and then used smaller scale typography for body text.

We also use hierarchy **to help users understand the relationship of content in a system.**

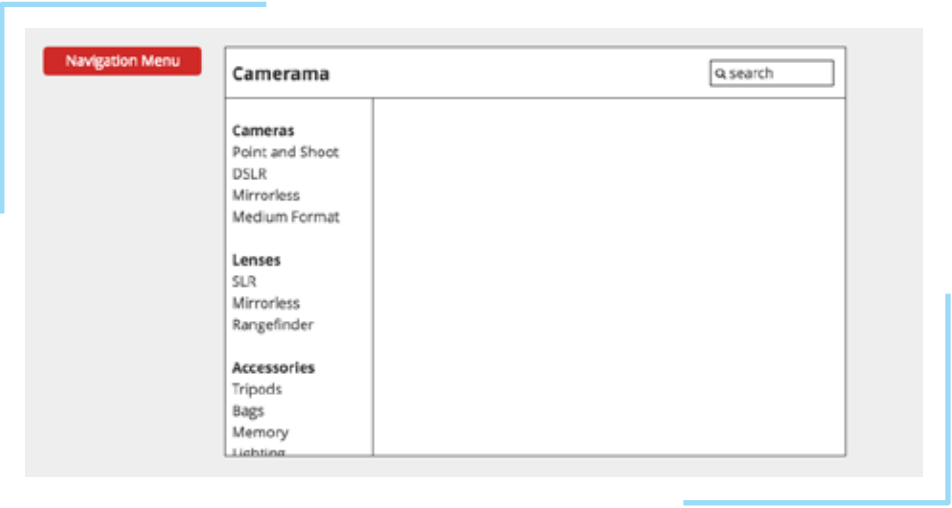
There are various ways of indicating hierarchy.

When we create menus with nested flyout sub-menus as in the example below, we're showing a hierarchical listing of functions in an application.



We're also using hierarchy when we're showing expanded lists of categories and their children as in the next example.

This might be the type of interface you would show in a shopping site, for instance and there are various ways of showing hierarchy in menus.

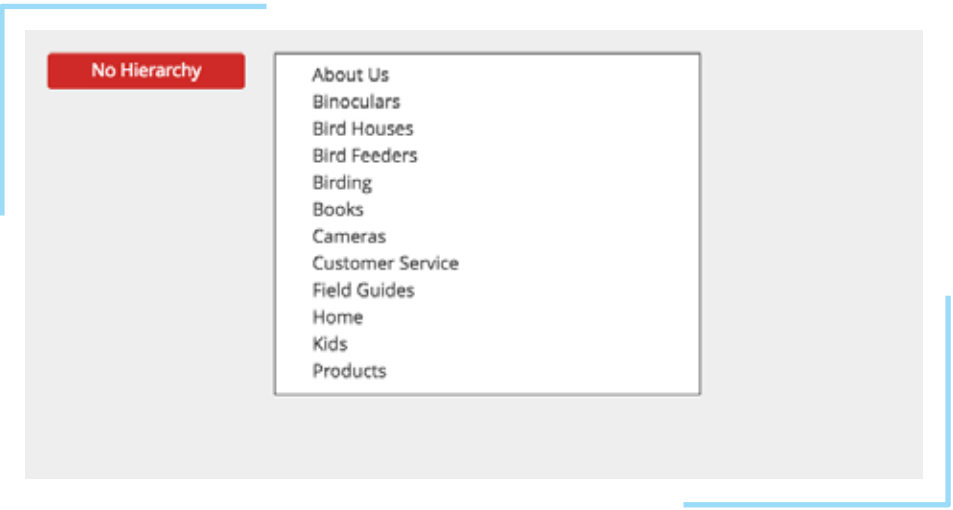


You'll find examples of hierarchy in software controls like the flyout Menus in your Windows or Mac operating system, in the collapsible lists in Windows Explorer or the Mac Finder that look like stairs when expanded and in Breadcrumb menus where parent-child relationships are indicated. Hierarchy is everywhere!

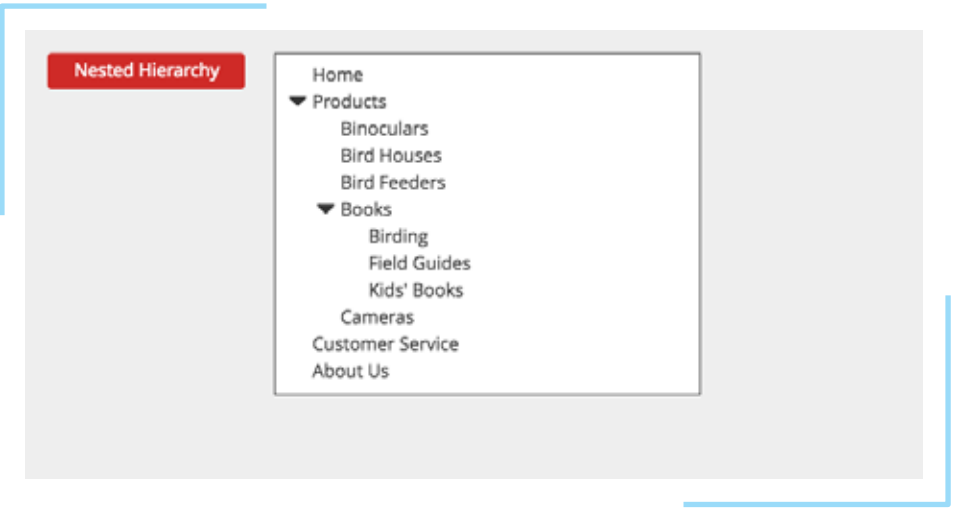
How to use hierarchy

The common way to achieve hierarchy is by **visually containing or nesting groups of child information within a parent container.**

Here's an example of a set of pages in a website, arranged alphabetically. There's no sense of relationship between the pages. They all appear to be equal in this flat hierarchy.



By grouping the information into parent-child relationships, we can create the menu for the web site below.



This is the kind of navigator you'll find in applications for nesting child branches in a program like PowerPoint, Keynote or even Balsamiq.

Hierarchy helps users make sense of complex systems.

There are great examples where hierarchy helps in extensive information systems. On large online stores like Amazon, for instance, hierarchy is used in navigation to help users drill down to content from broad to smaller categories.

And as we also saw in our contrast examples above, smaller interfaces such as forms can also benefit from visually displaying hierarchy within the screen.

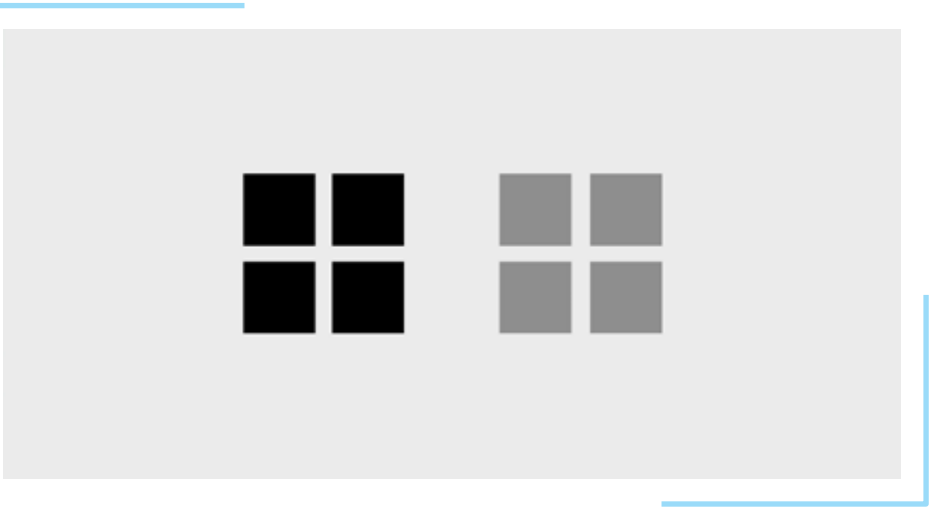
By establishing a highly contrasting text style for the headings, we indicate the most important parts of the screen content. This is also an example of how to use hierarchy in the screen.

Proximity

The Proximity principle is about **creating relationship between items by putting them near each other.**

The reverse is also true. Objects that are seen as distant from each other are understood to be unrelated.

We know how we perceive relatedness from human psychology. This is called the Gestalt principle of grouping. We can use this to our advantage in interface design.



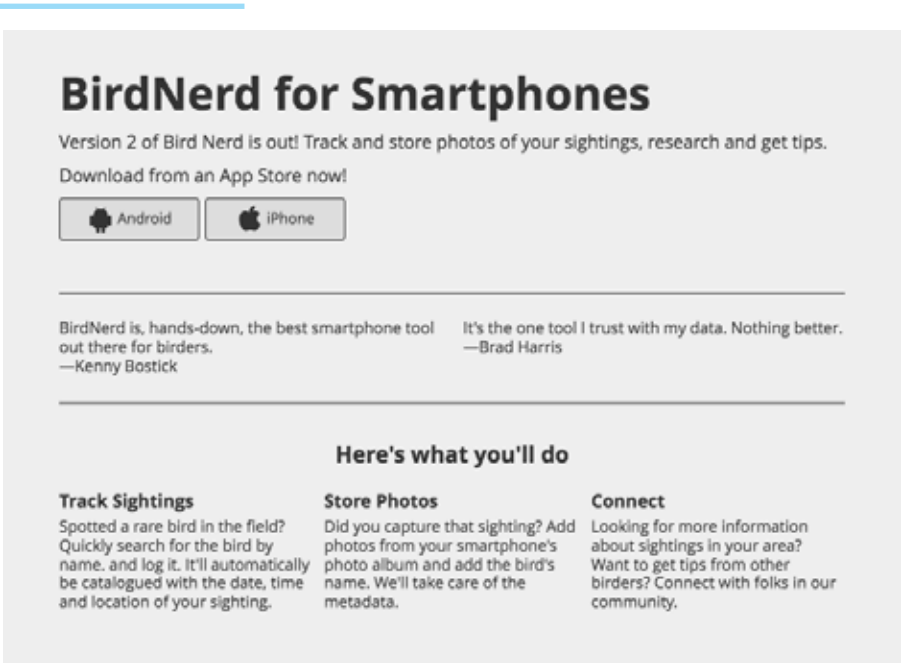
When to use proximity

When you want to use similarity to help the user, proximity can be helpful. Grouping items that are similar in function, for instance can help them find related actions.

This example from an application creates groups of buttons in a toolbar. This is a wireframe of the Keynote toolbar. The first grouping is for document-level actions like view, zoom, and adding new slides, the second grouping is for presentation functions, the third is for inserting content, etc. Grouping them this way helps the user to recall buttons that are similar in function with repeated use.

When we want sections of content to be clear we can move them apart to show that they're unrelated.

In the website wireframe below, notice how there is some important information at the top, including some buttons for important actions this site wants users to take. Then there is some more important, but denser information at the bottom of the screen. But then notice the smaller text in the middle separated using border lines. This is clearly serving a less crucial function, so the site decided to separate it from the other bits.



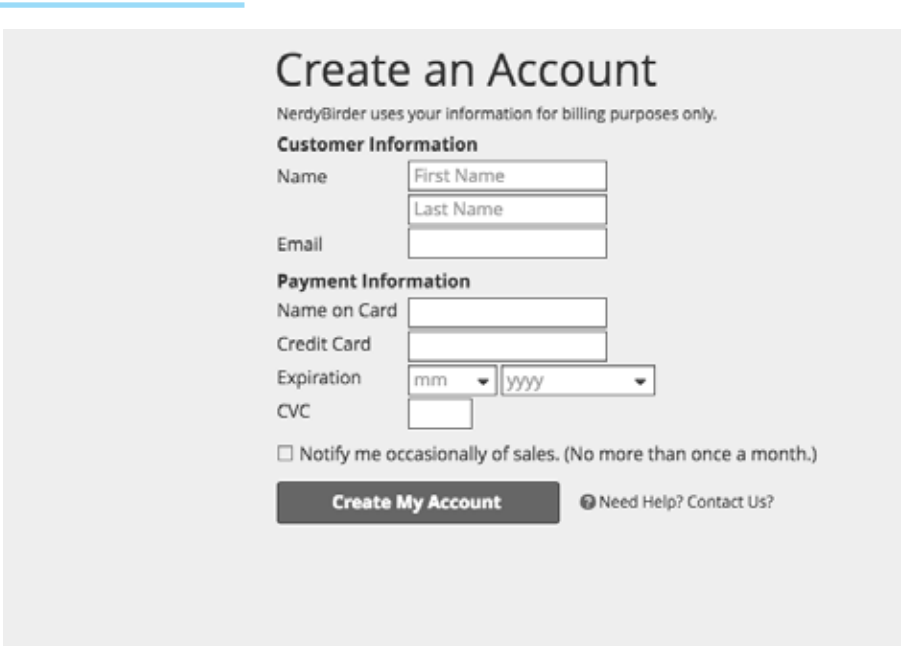
Proximity is useful in this case because it gives the user some help. In some cases, as in the testimonial block of the wireframe above, it might help to make something visible at a glance, but then guide them towards the more substantial chunks of info.

How to use proximity

We've seen in the examples above how the use of proximity can help **organize the information in your interface**.

To use proximity effectively, make sure you're using enough white space between groups. Let's bring back a more fleshed out version of our Customer Information form.

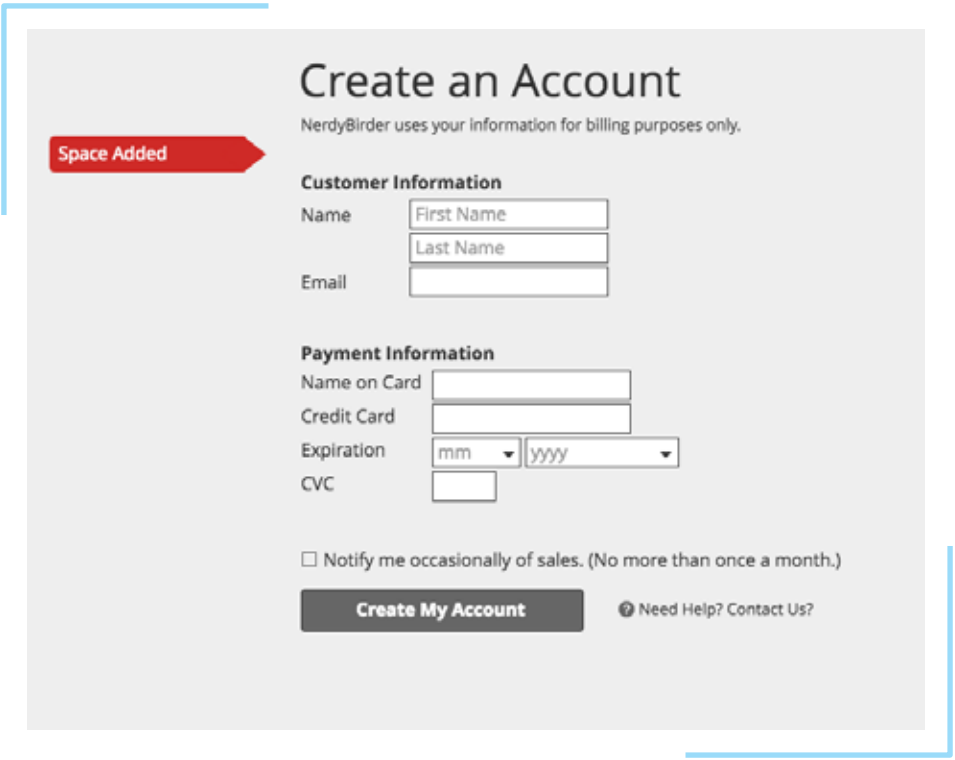
With the Opt-in checkbox, the submit button and the help tip, the form is more densely loaded with information.



The contrast we added to headings by making them larger or bold worked to help separate the parts of the form. But it's still a lot to get through.

Forms can be pretty tedious to fill out, so we can do a lot better to make it easier on our users.

Here's where proximity can really help. We can move the groups of related form fields close together and add white space to separate the unrelated groups.



This is much easier to take in, visually and the user has a clear indication of when they're done filling out a group of fields, so they can pause and work on the next.

With time you'll be seeing how the proximity principle is used everywhere.

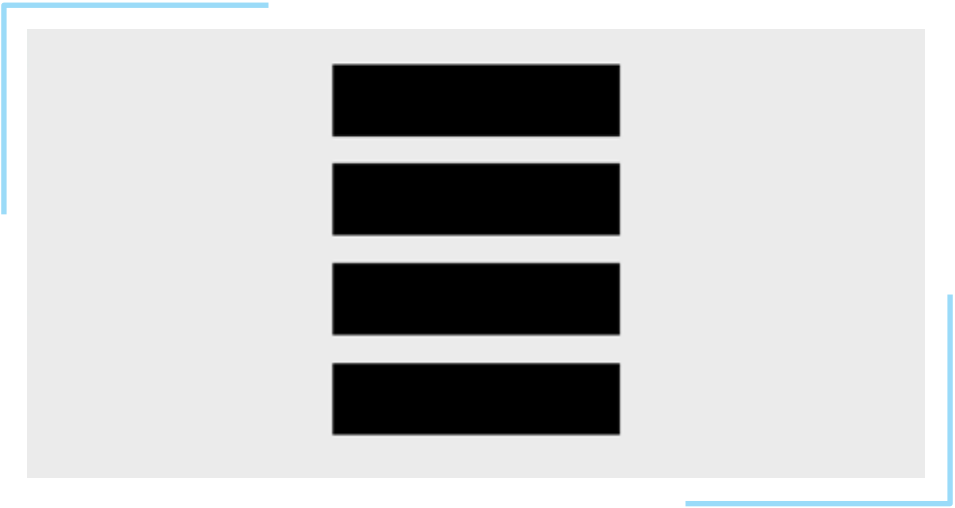
When you're wondering when the proximity of objects is enough or too little, try the squint test. Move back from the screen and blur your vision by squinting.

If you can detect that the clusters of information are separate while doing this, you'll know that proximity is working. If the space between seems too little to make the groups recognizable, try adding more space between groups.

Alignment

The alignment principle is about **making it easier for users to process information by guiding their eyes through aligned objects.**

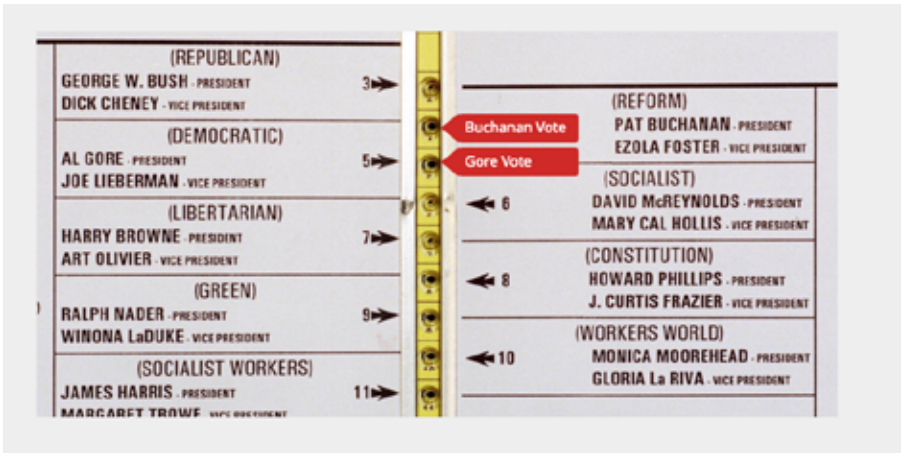
When users can scan through information that has a clear line or path for their eyes to travel, they intuitively know where to look to take the next action. Think of how hard it would be to read a book where all of the text is centered. Blocks of texts in books are left aligned so that your eyes automatically return to the beginning of the next line effortlessly. We can use alignment this way with more than just text in our interfaces.



When to use alignment

Alignment can and should be used **whenever you lay out a screen with information to process**. Using alignment, you can ensure that nothing is placed arbitrarily on the screen.

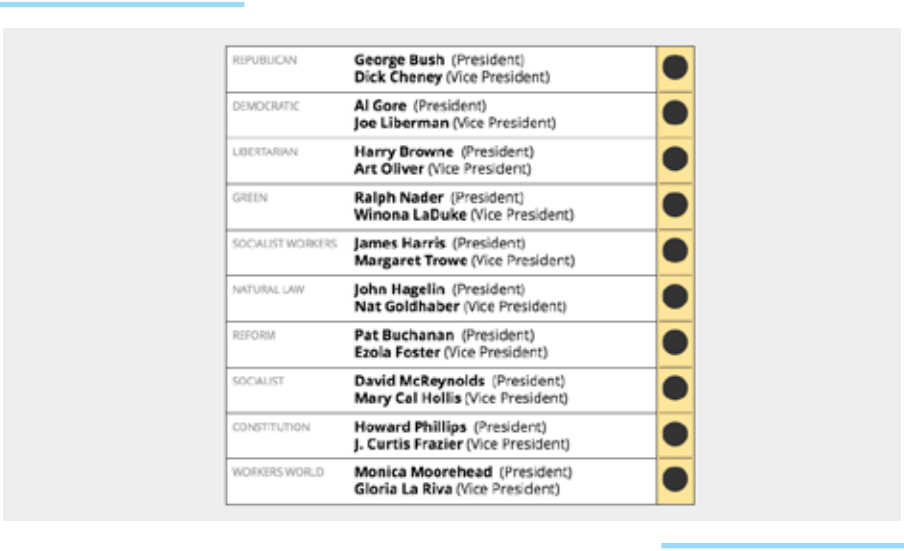
Seemingly arbitrary placement and lack of alignment can lead to unintended consequences. In the United States Presidential Election of 2000, a voting ballot in the state of Florida featured a poorly aligned interface that led to many voters voting for the wrong candidate.



This ballot confused a lot of voters. Note how the Democratic candidate, Al Gore, is the second on the left side. It became apparent that the Reform candidate Pat Buchanan on the right side had gotten many more votes than expected, and it is believed that the Democratic voters punched the second hole, registering a vote for Buchanan.

There are quite a few elements that lead to this confusion. The lack of alignment and the multiple border lines and arrows are hard to make sense of.

When you need to make the path clear to information, you can use alignment. This redesigned ballot might have made it more obvious to voters which hole to punch for their candidate.

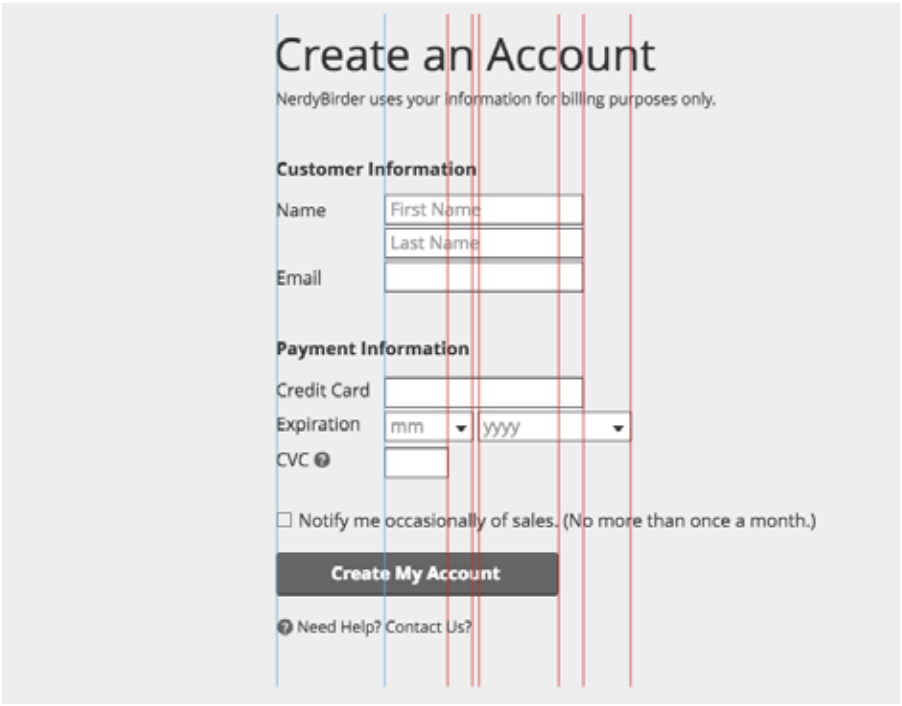


When you start to think that everything should be aligned to something on your screen, information becomes more clear and easier to use and your interfaces will begin to feel more cohesive.

How to use alignment

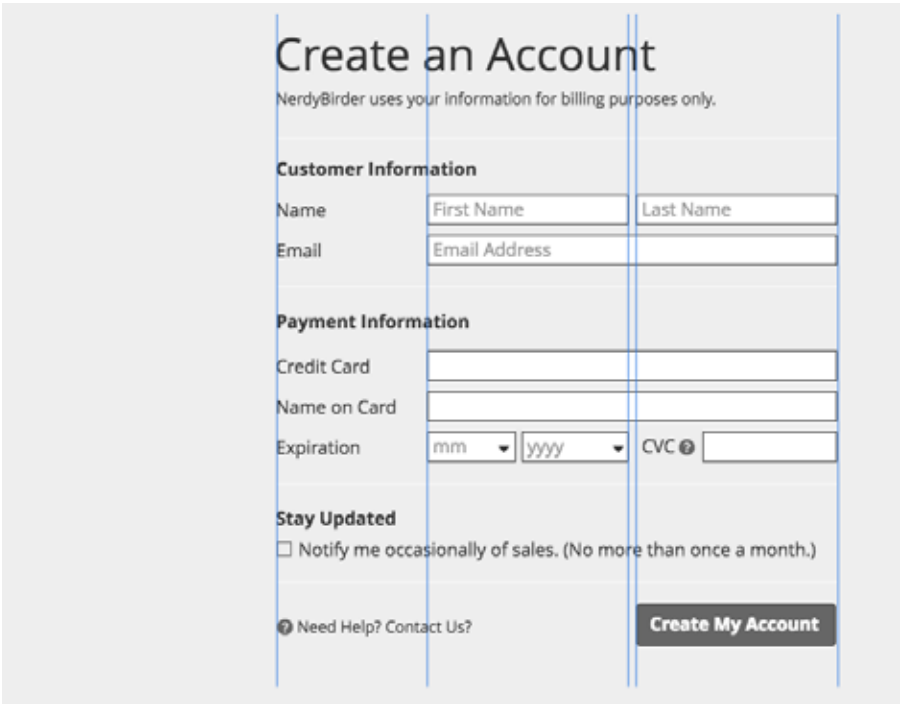
To use alignment, we look at everything on our interface and **think about how each item relates to the related items around it**. One exercise you can do is to try to see if each object can align with something else.

Let's take one last look at our Customer Information Screen. We did a good job at adding contrast and using white space to put related items next to each other. But take a look at the screen overlaid with blue and red lines below.



The blue lines show objects that have a strong left alignment to other objects. But the Red lines show how some of the objects don't align to other objects, particularly on the right of the form. The form inputs are sized so that they create a ragged right edge that makes the eye jump around on the right, compared to the strong left alignment.

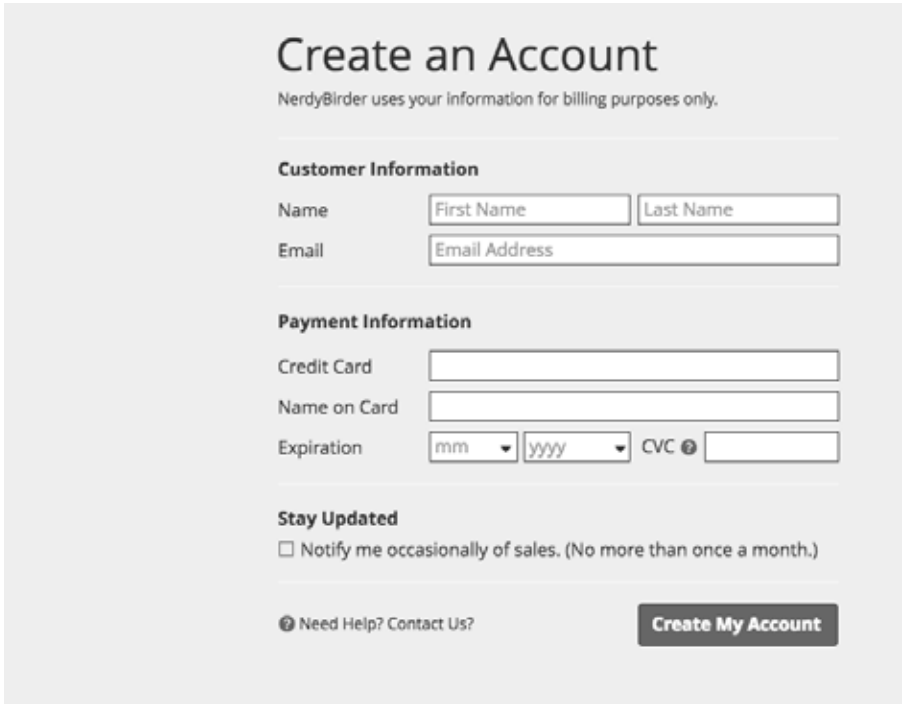
We can create a more solid feeling layout that directs the eye down through the form by resizing and laying out objects so that each one aligns with something else.



Notice the blue lines now. All of the objects have a strong left alignment, and most of the objects also have an object they align with on the right side. Adding a light hairline in our white space between chunks of the form also helps to create this sense of solid alignment.

We even reinforced the sense of movement by using the natural direction our eyes take when reading (down and to the right with Left To Right languages) by moving the submit button to the bottom right of the layout.

Here's what our form looks like without the alignment lines.



There are a lot of different ways you could have aligned objects out on this form that would work just as well. The goal here is to just think about how to help users move forward through the interface to completing whatever the intended task or goal is.

In this example, we showed how to use all of our base principles to improve one interface. When you're working on your own products or websites, use this step-by-step approach to see if you can apply the principles to each of your screens. You might find it useful to take a screenshot of your work before to compare how much easier it is to use your interface after you've applied the principles.

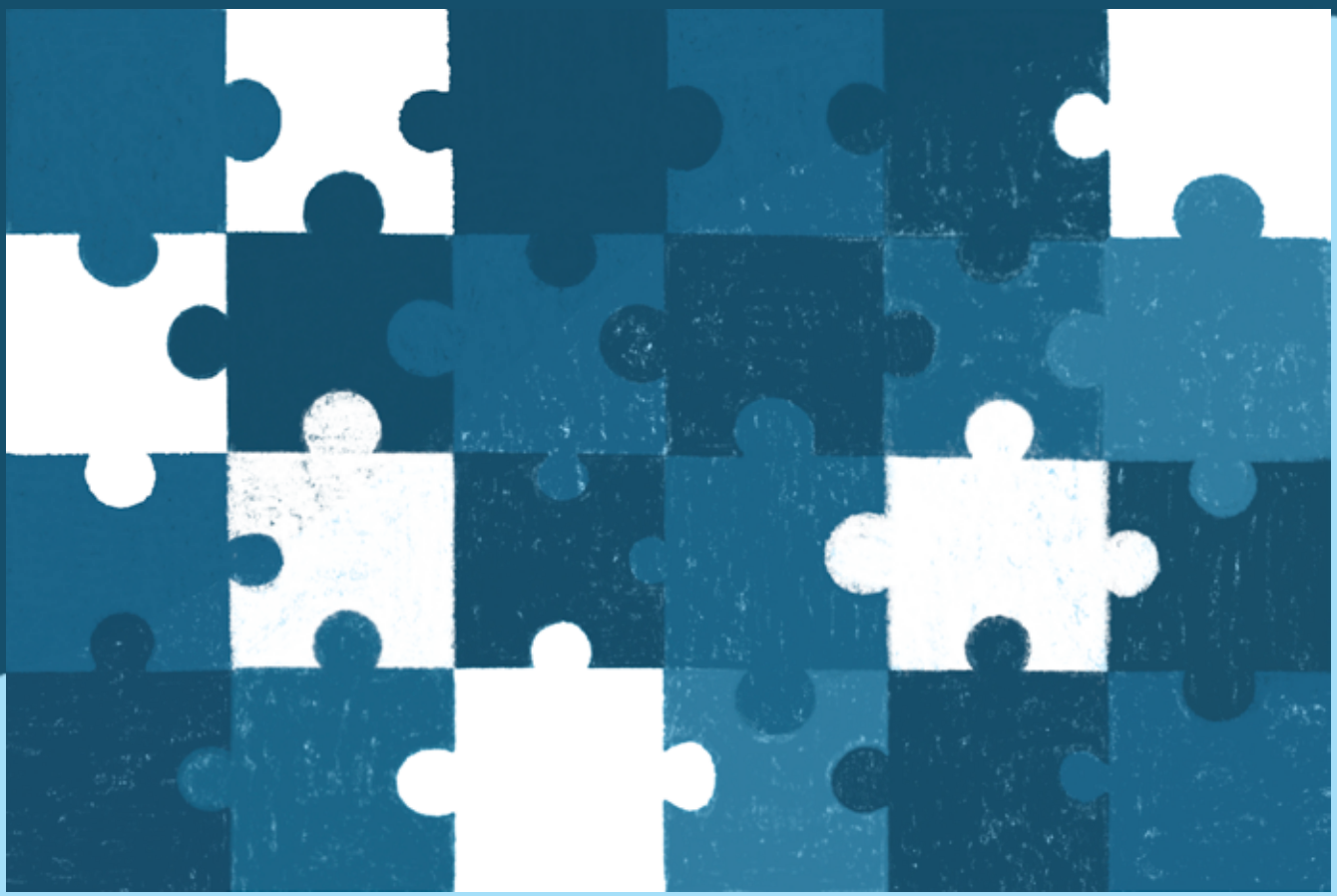
Further reading

- Robin Williams' "Non-Designer's Design Book", written for those new to graphic design, describes how to use a set of 4 core principles when communicating with design. The idea of beginning to internalize principles in the beginning of this section comes from her.
- "Universal Principles of Design" by William Lidwell

References

1: Note that this differs from individual and organizational design principles. Dieter Rams' 10 Principles for Good Design is one example. You can find more about those at the principles.design website.

5 UI Design Templates



UI Design Templates

You've probably used file templates before. A common type is an office document template, like a cover letter template in a text editing application.

Templates can be great because they provide a starting point for a document with some preset attributes. They help you go faster by starting your project from ready-made screens that you can edit.

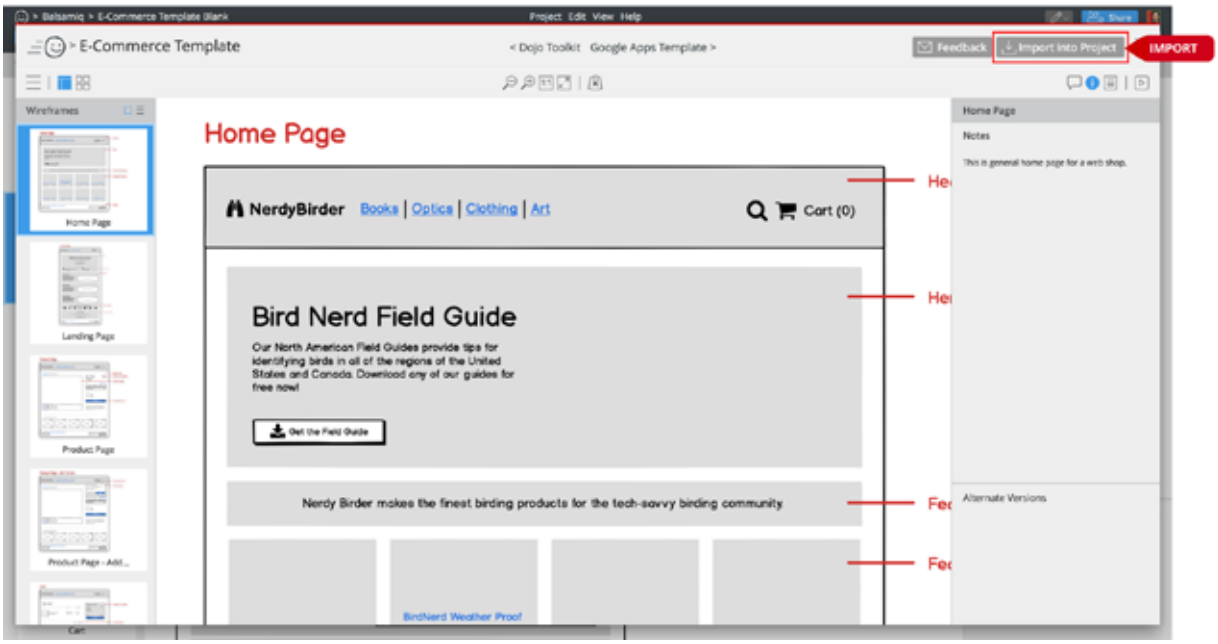
There are a few types of templates for interface design that you might use:

- **Solution-Specific** — Templates may be specific to a type of problem you're solving, and provide pages (screens) laid out with a library of controls for typical solutions. One example might be a template for creating a Search feature, or our previously mentioned E-Commerce Shopping Cart Solution.
- **System/Framework Specific** — A template may provide standardized screen layout and Symbols for a particular design framework. Some example templates may be for the Bootstrap framework or a design system like Material Design.
- **Corporate/Product Specific** — You may find it useful to create templates that are very specific to your use, like a template for your product with Symbols for your style guide.

Let's start by seeing how to modify a template. By the end of this section, you should be able to create your own.

Importing templates

In Balsamiq, a template is simply a starter project file that you import. If you're using Balsamiq Cloud, add a template via the menu *Project > Import Controls From Wireframes to Go...* Browse for templates and use the import button to add it into your own projects.



Selecting the screens you need

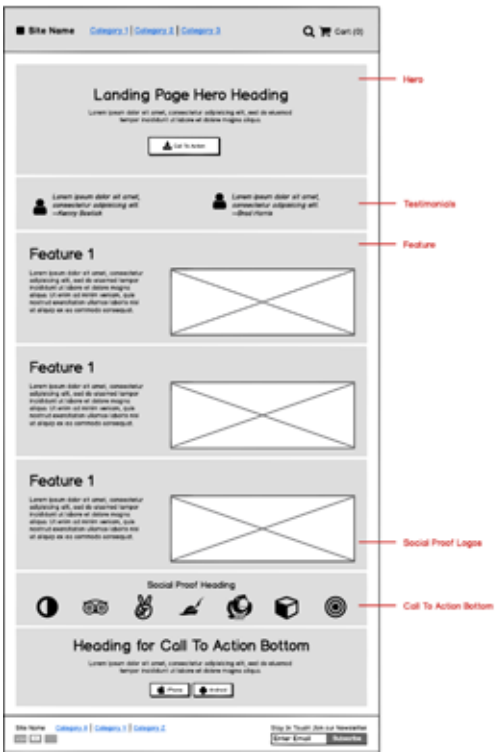
Continuing our example of an online shop, let's use an E-Commerce template to design more of the shopping experience. Obviously, we'll want to flesh out our shopping cart flow all the way through confirming the transaction.

Below is an overview of the screens that come in this template. Your solution may not require all of them. You'll likely have unique user expectations to consider and business rules to meet, so you'll modify the experience as necessary, pick out the ones you need and discard the rest.

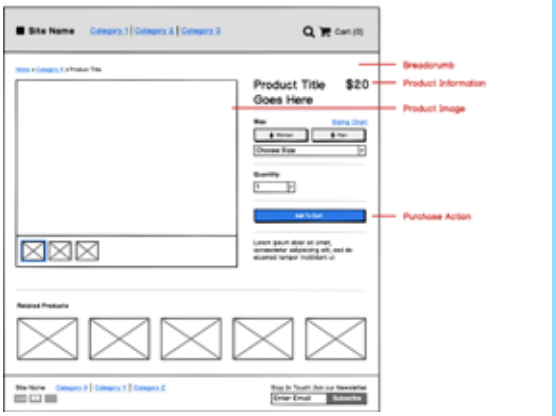
For our example, let's say we want to flesh out the pre-purchase experience on screens like the landing pages. Let's use the template to create a landing page for a fictional smartphone app.

Home Page

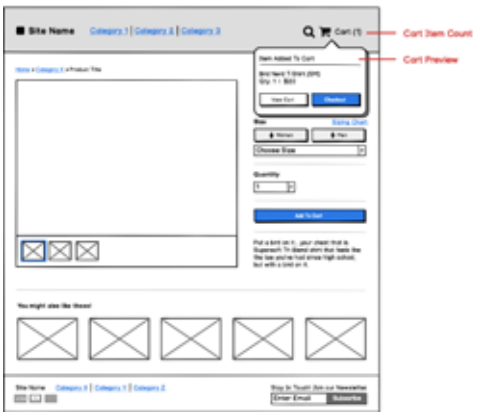
Landing Page



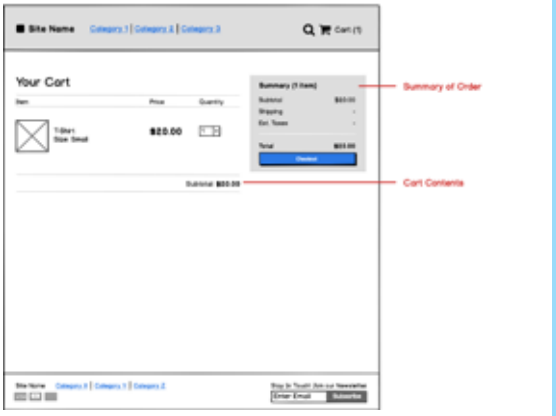
Product Page



Product Page - Add To Cart



Cart

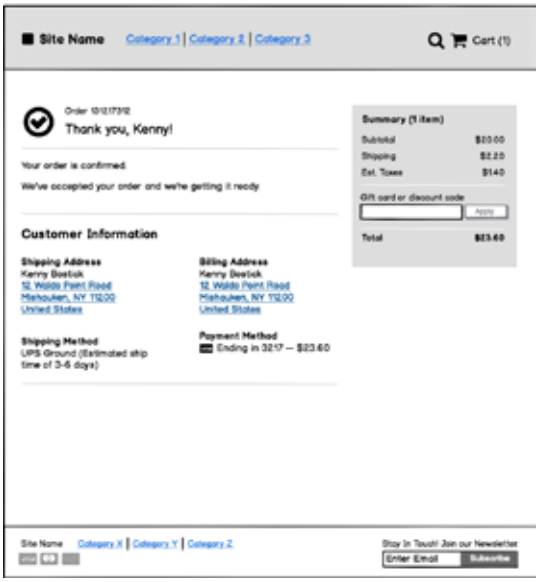


Checkout - Customer Info

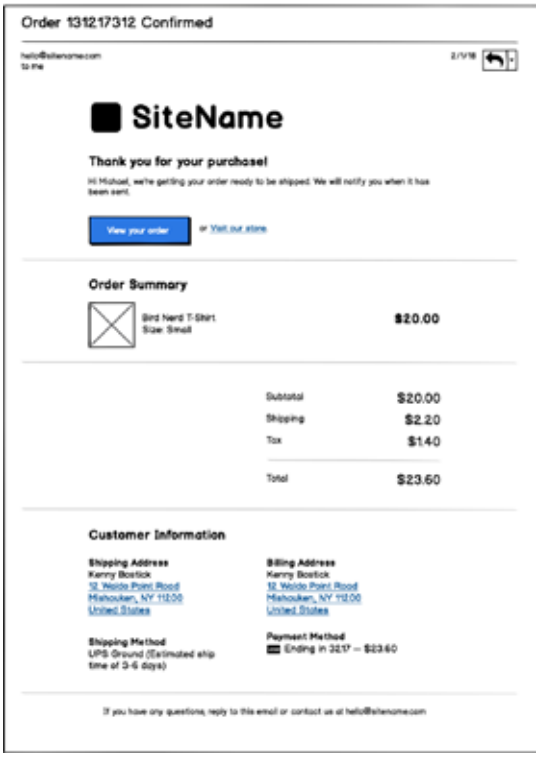
Checkout - Shipping

Checkout - Payment

Checkout - Success



Email Confirmation



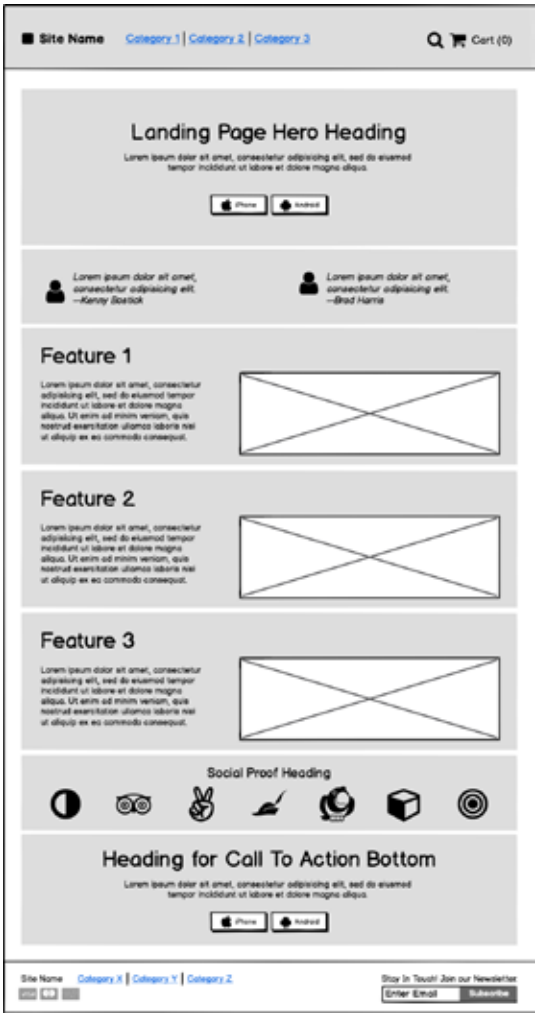
Modifying pre-made screens

For this example, we'll modify the generic Landing Page template using our own copy and add some elements for our fictional smart-phone app.

When you open a template you'll probably see some placeholder copy as in the example on the left below. Simply select and edit each of the place holders.

In the wireframe on the right, we've modified the text, we added some wireframed elements to illustrate our product, and using our new knowledge of design principles we laid out some of the parts to fit better with our new illustrations and copy.

Template with starting suggestions to design a landing page.



Edit placeholders to customize the screen.

Pretty simple! Hopefully, editing a template should seem much faster than laying out all of the components from scratch. You'll want to tweak it to suit your needs, but it's a good way to start the design discussion with your team.

A note about symbols libraries

Some templates may also contain Symbols Libraries. Our E-Commerce template has Symbols for the components used on the screens above.

The idea behind using Symbols is that you can easily drop in these reusable components that are made specifically for this type of project. You can find out more about Symbols in the Balsamiq Docs.

Closing thoughts

Like design patterns, templates serve as a starting point and are usually built on an understanding of what typically works well for a common problem. Templates differ from **design patterns** in that they **usually propose a plan for solving a broader scope of problem**.

Interface designers are known for building systems to make their work efficient. Templates are one example of that. Once you've become adept at working with templates you can start to set up your own systems to make your work go faster. If you work in-house on a product team, create templates that reflect your organizational style. If you work in an agency or design/development shop, create templates for the repeatable types of client projects you work on.

The Cooking Analogy:
Let's step back for a moment and put templates under the lens of our cooking analogy. In cooking, the orchestration of dishes that work well together create a meal — a complete culinary experience. Templates are like that. Pieces of user interface often go well together to create an entire experience.

As you explore your users and their problem more deeply, you'll begin to see what parts of the template work for your needs and swap out what doesn't, just as you would deviate from a recipe to make it your own.

Additional Resources

- Download the template used in this section: E-commerce Template from Wireframes to Go
- Landing Page Template and Designing a Landing Page tutorial

6 An Opinionated Guide to Creating Software

An Opinionated Guide to Creating Software

The final lesson in our Introduction to User Interface Design Through Wireframes course is about what process to follow in order to make sure that your ideas end up seeing the light of day in production, and delighting your end users day after day.

The previous lessons discussed how to design user interfaces. This one, focusing on the design *process*, is the result of many interviews that Balsamiq has done with entrepreneurs, product managers, UX designers, marketers and developers. Every organization is a little different, but we think that what we describe can serve as a useful starting point for you and your team.

The process is a little different if you're not creating something completely new but are instead adding a small feature to an existing software or website. It's also a little different whether you're a product company or a consulting company. We'll highlight those differences along the way.

The steps in the process are:

1. Gather requirements
2. Sketch out ideas
3. Submit your proposals to the core group
4. Incorporate feedback into wireframes
5. Pitch to the larger group
6. Make a high fidelity prototype (optional, discouraged)
7. Work closely with developers
8. Help with testing, deployment and marketing
9. Monitor how it's received

Step 1: Gather requirements

So it happened: an idea for a new product or feature landed on your lap and you have to design a solution for it.

Ideas come from all over the place: if you work for a product company, they often come via the support or sales teams because of customer requests. Other times they are deemed strategically important by *the powers that be* (PMs, executives, marketing...). Some other times, the development and design team come up with them! If you're a solo founder, you're doing all of this idea gathering and prioritizing yourself!

If you work for a consulting company, these dynamics often happen within your clients' organizations. They just call you when they're ready to execute on the idea.

Regardless of how it happened, it's time to start executing.

The first step is to understand what people are asking you to do. This phase is normally called gathering requirements.

Gathering requirements is its own discipline, some people make it their whole careers. Here's a very high-level overview.

First of all, make sure you **shift the conversation from solutions** and ideas for solutions **back to "what problem are we trying to solve"**.

Humans have a very strong tendency to automatically jump to solutions any time they encounter a problem, so it's likely that by the time the request comes to you, it's already framed with a solution in mind.

For instance, you might be asked to "add search to the site or app". Seems clear enough, right? Well, because the request is framed as a solution, you might do all the work to design it, but the solution might miss the point entirely. Maybe the problem your customers wanted to solve would have been solved much better by improving the onboarding flow, or by adding phone support as a support channel! You just don't know!

So step one is to create a wiki page or Google Doc — or whatever tool you use to communicate internally on projects — and put the "What Problem(s) Are We Trying to Solve" title right at the top.

Now, **make sure you assemble the correct group of stakeholders**. You should include those who asked for this feature, leadership from development, marketing, and ideally someone who will be impacted by this change. If it's too hard to find an external customer (it shouldn't be too hard), find an internal customer to act as their proxy. Someone from tech support, for instance.

Next, go around and ask all the stakeholders *what problem are we trying to solve with this change?*, and take notes on the page. In some cases, you might have to use the 5 Whys technique to get to the root of the problem. Look for patterns in the responses and categorize the answers so that you have a clean bullet list of requirements on your page.

Once you get to the bottom of what the problems are and have digested it fully (sleep on it!), you might want to circulate the page with the stakeholders again, to make sure everyone agrees that yes, those are the problems we want to solve.

At this point, you should also have a better understanding on how important or urgent this change is for everyone.

Once that's done (ideally in a day or 2), you can start shifting towards thinking about possible solutions.

First up, try to avoid the *"if all you have is a hammer, everything looks like a nail"* problem. In other words, don't jump to what's easy for you to do. Try to think more holistically, and **really try to put yourself in the end-user's shoes**.

Don't limit your thinking to a software solution, think about the Whole product concept instead. Maybe the best solution is better documentation, or training, or a process change, or starting a podcast!

Spend a day or 2 testing these possible super-high-level ideas with the stakeholders again.

If you all agree that a software solution is worth pursuing, it's time to move on to the next step.

Note that this doesn't necessarily mean that you're committing to implementing whatever solution you're about to design. It might well be that what you design cannot be implemented in time and on budget, or cannot be supported by your organization... at this point you're just committing to exploring a possible software solution to the problem.

Further reading

- 10 techniques for gathering requirements
- Software Requirements (3rd Edition)

- Stop gathering requirements - How To Be A Good Product Manager
- Product requirements documents, downsized
- Gathering Product Requirements
- Using Wireframes with Agile User Stories

Step 2: Sketch out ideas

This is the most creative part of the whole process. It's a lot of fun.

Some software teams like to do this part together in a meeting room with a whiteboard, or by using real-time-collaboration in a wireframing tool like Balsamiq Cloud. Other teams like to do an exercise where multiple people come up with solutions on their own, and then get together to review them and pick the best elements of each.

The vast majority of the time though, this phase is done by one person, alone.

So, make sure you get a good night sleep, block out at least 2 hours in your calendar, make sure you're properly caffeinated, quit Slack, turn off email notifications, put your phone to charge in the other room, close your door, put your headphones on, play some relaxing music (Balsamiq Wireframes has it built-in in the View menu), and get yourself ready to get in the flow.

If you're artistically inclined and are not afraid of a white page, you might want to start by sketching ideas on paper. Just keep it high level, boxes and arrows, so to speak.

If you can't draw a straight line to save your life, or find a white page intimidating because it allows you too much freedom (which might make your design too hard to implement), a tool like Balsamiq is perfect for you. You can just pick standard elements from the UI Library and assemble them. Make sure you use the Sketch skin for this step! Switching to the wireframe skin or — God forbid — using a tool that allows you to **design at high fidelity, is a terrible mistake at this point**! Unless you have superhuman self-control, you'll find yourself thinking WAY too far ahead, fiddling with details like colors, fonts, and pixel-perfect alignments. That's a huge waste of time right now.

If you're just adding a piece of functionality to an existing user interface, you might find it faster to take a screenshot of the existing UI and tweak it by using the crop image tool.

Now it's the time to use all the tips and techniques you learned about in the previous chapters.

Most importantly, **don't be afraid to start over**!

If you want to see what this process looks like in real life and in real time, check out our *Wireframing with Balsamiq* YouTube playlist.

Once you're at a good stopping point, go for a walk, take a hot shower, or a quick nap! — the warm and cozy environment away from the computer will help your brain digest what you just did and suggest changes and improvements.

Your goal right now is not to come up with "the definite solution", but rather to **come up with some user interfaces to submit to the team as proposals**, to see how they react to them.

Be careful not to fall in **the "version 3" trap**! Because of how our brains work, we usually tend to put way too many details in our wireframes.

So, once the initial wireframes are done, try to scale them back once (version 2), and then do it again, to get to what should be released in version 1.

If you want to show the whole vision to the team, add some annotations that say "v.2" or "this can come later". You could use yellow Pointy Buttons controls for those, or Vertical Curly Braces.

Further reading

- Creating Your First Wireframe
- How to Start a Wireframe Project
- Sketching User Experiences: Getting the Design Right and the Right Design

- UX Sketching
- Why It's Important to Sketch Before You Wireframe
- Quick, Useful UI Sketches
- UI Sketching Conventions

Step 3: Submit your proposals to the core group

Once you have a few sketches, it's time to go back to the stakeholders for a quick check-in, to see if the direction you've gone in makes sense to everyone.

Make sure you call these sketches "early, quick drafts" and you tell everyone that "you're not married to any of these ideas". This way they won't hold back honest feedback for fear of hurting your feelings.

DO NOT email your sketches or otherwise "send them over the wall". They're too rough to be understood. Schedule quick 1-1 chats with each stakeholder, and walk them through each sketch.

Your goal is to **gather as much feedback as possible** on where each role would like the design to go. This step is important because your sketches — even in their roughness — are very powerful: they allow everyone to visualize the solution in their minds.

This will in turn trigger thoughts and concerns that they couldn't have thought about during requirement gathering.

Note that **each person might care about different aspects of the solution**. Executives or sales people might care more about branding and strategic importance rather than usability. Developers care mostly about implementability, graphic designers generally care about aesthetics. Learning what each stakeholder cares about is an important part of the process! The more you do it, the better your wireframes will be next time: just like in those old cartoons, you will feel like you have each little person on your shoulders giving you feedback as you design.

Make sure to take good notes as you do this round of feedback — put them on your wiki page!

If some of the feedback is great, "obvious" and non-controversial, go ahead and incorporate it in your design right away, you don't have to wait until you've spoken to everyone.

A word of caution: if you get very little feedback or just simple "looks good to me", you should view it as a red flag. Either people aren't engaged, don't understand what you're proposing, are the wrong audience, or something else. Design at this level of fidelity should **always** elicit feedback.

Step 4: Incorporate feedback into wireframes

Now it's time to turn your sketches into proper wireframes.

Look at your different designs, study your feedback list, and try to put it all together into a single solution.

If you had been using paper sketches before, this is a good time to transform those into Balsamiq wireframes, as they're generally easier to understand and share.

Once again, use everything you've learned in the previous chapters to create wonderfully usable, lovable designs.

Note that you should still very much **keep things at low-fidelity** here, it's still too early to risk getting people get distracted by pretty colors and icons. This is especially true when building a new feature, web page, or product.

We recommend only designing the key screens at this point (probably less than a dozen). Just design the Happy Paths for now. There will be time to think about confirmation dialogs and error conditions later.

You might want to use the Symbols feature for headers and footers, and you might want to link screens together to help you walk through them — though it's generally not required as long as you sort your wireframes in a way that flows naturally.

We believe that it's important at this point to **allow others to participate in the wireframing process**. Just because their job title says "developer" or "business person" it doesn't mean that their user interface ideas should be discounted. Balsamiq, for instance, was built specifically to allow non-designers to have a seat at the table. Don't be too protective of your designs. Instead, invite others who are interested in sharing their user interface ideas to your project! In Balsamiq Cloud for instance, you can do this very easily. They can work alongside you, creating alternate versions for you to review, and you can easily comment on their designs so you can iterate together.

Make sure you **work closely with development** at this point! They will likely have constraints you don't know about, and their greenlight — even a reluctant one — is essential to move forward. By the time you're done, you should know that what you designed is buildable by X people in roughly Y amount of time.

Further reading

- What is Wireframing and How It Can Improve Communication
- What Are Wireframes?
- How Product Managers Build Healthy Relationships with Designers

Step 5: Pitch to the larger group

Once you've worked on your design for a couple of days, and maybe circulated it electronically with the core team one more time for final review, it's time to present it to everyone.

This will be more of a sales meeting than a brainstorming meeting. You will be pitching your design to all the stakeholders, confident that it will satisfy the requirements. You will still accept feedback, but don't expect too much of it.

Aside from all of the stakeholders from step 1, make sure you invite the person that's *high up* enough to be able to **give your design a green light, so that development can start** — usually an executive, or *the client*. That's your main goal with this meeting.

One trick that some of our customers use at this point is to switch to the Wireframe skin. It's still low-fidelity, but it's a bit more polished, just enough not to elicit the same amount of feedback as the Sketch one.

Walk people through each of the screens you designed, one by one. Try to keep the presentation engaging: avoid getting bogged down in too many details. If objections occur, make a note of them and promise to deal with them 1-on-1 after the meeting (if possible).

If you did your job well involving the core team over and over ahead of the meeting, this presentation should be a breeze.

See Tips for Presenting Your Wireframes for more about this important milestone.

Step 6 (optional, discouraged): Make a high fidelity prototype

Back in the days of waterfall, when 18-month release cycles were the norm, building prototypes was a great way to reduce the risk that your solution — even if well thought-out and approved by your team — would miss the mark with customers.

Today's world is much more *agile*. Developers can turn your wireframes into something for you to play with in days, not weeks or months.

High fidelity prototypes take a while to build (you have to design every single screen in detail, link every single element to its destination, provide fake data, etc.) and are done with expensive tools with a high learning curve. If you have a UX designer on staff — lucky you! — this would be a job that only they can do.

Iterating on prototypes is also painful: changing high-fidelity designs in Sketch or Photoshop takes time, and re-wiring all the pieces of the prototype is a painful process.

Last but not least, once you're done with a prototype, you throw it away.

For these reasons and more, we usually advocate skipping the high fidelity prototyping phase entirely, and **going straight to code**. Even if your wireframes missed the mark enough for you to have to change 15% of the UI code, you'll still be able to keep the remaining 75%.

That said, creating high fidelity prototypes might make sense sometimes — say, you're building a car, or a satellite... something not easy to iterate on after it's released.

Some, usually large, companies use high fidelity prototypes to get buy in from executives. The idea is that in the enterprise world, *the suits* don't want to see low fidelity, sketchy-looking stuff. They want to see the real thing, in all of its branded glory.

Another use of high fidelity prototypes is for self-service user-testing. Some people think that end-users will not give you meaningful feedback unless they see the whole thing, as if it was real. It might be, but we've involved our users at the wireframing stage several times, with great results.

You should try and see what's best for you and your company.

If you do this step, expect to add at least a few weeks to the schedule.

Step 7: Work closely with developers

Alright so you have the green light, now it's time to work closely with development to make sure that your designs get built faithfully.

You can help in several ways.

First off, **add a lot more details and annotations** to your set of wireframes.

Be ready and quick to answer any questions the developers will have.

Make time in your schedule to review early implementations, on staging or via screen-sharing sessions from the developers' machines.

Think of all the possible edge cases and write them down. This will help the developers write unit tests, it will help manual testers to verify each case, and it will allow your support and documentation teams to make sure everything is well documented, with the right screenshots.

Schedule weekly (or even daily!) check-ins with the developers, to see how work is progressing.

Don't be afraid to go back to wireframing! If you notice parts of your design not feeling quite right once they've been implemented in real code, go back to your wireframes and tweak the design to make it work! Life is full of compromises after all.

Step 8: Help with testing, deployment and marketing

As the feature gets close to being ready, you'll be in a great position to help answer questions about the feature from different stakeholders.

Marketing will want to know how to highlight the feature, the documentation team will have you review their work on it, the execs will want a real demo before launch.

You'll be very popular, and like a proud parent, you'll see your new feature get released to the world. Yay!

Step 9: Monitor how it's received

It's amazing how many people don't do this, or don't even consider doing this.

In a way it's understandable, it was such a struggle to get from idea to production, most designers are probably exhausted at this point.

But if you don't know if your feature is well received by the end users, how will you get better at designing?

There are many ways to see if a feature was successful: you can track usage via metrics, you can look for mentions of it on Twitter, or — our favorite way — you can interview your users about it!

Closing thoughts on process

That's it. We hope this brief high-level outline helped you in preparing for what the life of a UI designer — even a non-professional one — is like when taking something from idea to production.

We hope it gave you a sense of who the stakeholders are and what each phase entails.

As we said before, every organization is different, the process above is a weird average of what we heard during dozens of user research interviews with Balsamiq customers.

If your process is different, tell us about it! We're always happy to improve these guides.

balsamiq.com/learn