mavocado

MADE FOR PEOPLE. / A PRACTICAL GUIDE

From Screens to Systems.

A practical UI design guide for startup product teams.

For founders, junior designers and small teams building a growing product.

Leave with a small foundation set, a component checklist and a practical review routine.

8 PAGES / 5 CHAPTERS / A WORKSHEET TO USE

mavocado.in | hello@mavocado.in | +91 77956 70032

01 / THE PRACTICE

Audit the interface you already have

Look for repeated decisions before creating more components.

Collect a small set of important screens and states. Include the journey that matters most to the product, not only the cleanest presentation screens. Compare buttons, fields, text styles, spacing, navigation and feedback. Note where similar elements behave differently without a clear reason.

Group the findings by the decision they represent. Two buttons may look different because they communicate different priorities, or because the team made the same decision twice. Understand that difference before merging them. Keep a short inventory of patterns, intended use and unresolved questions.

AN EXAMPLE

A dashboard contains three versions of a Save button. One commits a draft, one sends a message and one confirms a destructive change. The audit reveals that the labels and consequences need attention as much as the border radius.

PUT IT INTO PRACTICE

- [ ] Include default, loading, empty and error states.
- [ ] Record the purpose of each repeated pattern.
- [ ] Separate intentional differences from accidental variation.

Try this: Choose five repeated elements. Write what each one helps the user do and where its behaviour changes.

02 / THE PRACTICE

Create a small foundation set

Use named decisions instead of scattered values.

Define a manageable set of type styles, spacing steps, colours and corner treatments. Give them names that describe their role in the interface. Semantic colour names such as text-primary or action-primary are easier to discuss than a long list of isolated colour values.

Choose the set around real screens and content. Review readability, hierarchy and contrast in context. Keep body text comfortable and distinguish interactive elements without depending only on colour. Document where a foundation applies and where an exception is intentional. The goal is a useful shared language, not an impressive token count.

AN EXAMPLE

A small product might begin with page and section headings, body and supporting text, a short spacing scale and a few semantic surface, border and action colours. Test those choices on a form and a dashboard before expanding the system.

PUT IT INTO PRACTICE

- [ ] Name values by purpose where that helps decisions.
- [ ] Check type and colour in real content and states.
- [ ] Keep the first set small enough to explain and maintain.

Try this: Choose one screen and replace repeated ad-hoc values with a short, named foundation list.

03 / THE PRACTICE

Design components as behaviours

A component is more than its default appearance.

For each component, define its purpose, content, states and interaction. A field needs labels, help, input, validation and disabled or read-only behaviour when those states are relevant. A button needs a clear action, loading feedback and a sensible way to prevent accidental repeated submission.

Review the component inside the complete journey. Explain what happens before and after it is used. Include keyboard interaction and visible focus with the development team. Avoid creating variants for every small visual difference if a simpler rule can explain the pattern.

AN EXAMPLE

A download action may validate a name and email, wait for a server response and then offer a PDF. Its loading, success and delivery-unavailable states need different language. A single attractive button mockup does not describe that behaviour.

PUT IT INTO PRACTICE

- [ ] Define default, focus, active and unavailable behaviour.
- [ ] Include loading, success and error feedback where relevant.
- [ ] Explain the component’s action and consequence in plain language.

Try this: Choose one component and describe every state a user could encounter during its main task.

04 / THE PRACTICE

Connect the library to implementation

Make the handoff answer the developer's next question.

Agree how design names relate to code names. Document component anatomy, variants, spacing rules and content limits with working examples. Explain responsive behaviour rather than supplying only a wide and narrow screenshot. A useful specification makes decisions and dependencies visible.

Review implementation together. Some differences come from platform behaviour, content, data or technical constraints that the design file did not model. Decide whether to update the implementation, the design or both. Keep a record of the agreed rule so the same question does not return with every screen.

AN EXAMPLE

A service card has a title, two lines of introduction and four capability points. The implementation needs to know whether the title can wrap, how cards align when content differs and what happens when only two capability points are available.

PUT IT INTO PRACTICE

- [ ] Map important design names to implementation names.
- [ ] Show representative content, limits and responsive rules.
- [ ] Review real browser behaviour with the development team.

Try this: Write a one-page handoff note for a form, including layout rules, validation and confirmation behaviour.

05 / THE PRACTICE

Keep the system useful as the product changes

Give consistency an owner and a feedback loop.

Choose a simple way to propose and review changes. Ask whether a new need can use an existing pattern, needs a variation or deserves a new component. Consider the effect on other screens before accepting a local improvement. Keep ownership clear enough that questions reach the right person.

Use the library as part of ordinary product work. Review a small sample of implemented screens regularly. Update guidance when it no longer matches reality and retire unused variants. A system becomes trustworthy when the team can understand its current rules and see how those rules evolve.

AN EXAMPLE

A new reporting screen needs a compact filter. The team checks whether the existing filter pattern can adapt without harming labels or keyboard use. If a new compact variation is needed, it documents when to use it and where to avoid it.

PUT IT INTO PRACTICE

- [ ] Define who reviews additions and changes.
- [ ] Keep a short decision log with real examples.
- [ ] Audit implemented screens and remove obsolete patterns.

Try this: Agree a lightweight review routine: owner, cadence, proposed change and evidence needed.

MAKE IT YOURS

Turn the ideas into a working plan.

Use this page in a team conversation. Keep answers concrete, note the evidence and leave uncertain points open for review.

01 / The repeated decisions

Which five patterns appear most often in the product?

02 / The foundations

Which type, colour and spacing decisions should be shared?

03 / The component states

Which states and interactions are missing from the library?

04 / The implementation link

What names, rules and responsive examples will developers need?

05 / The maintenance plan

Who owns changes? How will the team review them?

A fresh perspective.
A practical next step.

Begin with a real journey and its repeated patterns. Build only enough system to make the next design and development decision clearer.

Mavocado designs websites, mobile apps and SaaS products around the people who use them and the businesses behind them. If you need help applying the ideas in this guide, start with the problem you want to make clearer.

TALK TO MAVOCADO

hello@mavocado.in
+91 77956 70032
mavocado.in

About this guide

Original educational material by Mavocado Design. The examples are illustrative. Use the prompts as a starting point and check decisions with your users, product context and implementation team.

Further reading: W3C Web Accessibility Initiative - Forms Tutorial. Practical guidance on labels, instructions, validation and user feedback.
www.w3.org/WAI/tutorials/forms/