SAMPLE CHAPTER

Usability Matters

Mobile-first UX for developers and other accidental designers

Matt Lacey

Usability Matters

by Matt Lacey

Chapter 6

brief contents

6 User-entered data

This chapter covers

- The principles behind good form design
- Alternatives to typed entry
- Optimizing form entry
- Handling validation and reporting errors

In this chapter, we look deeper at some of the specifics of capturing data from the people using the app. Few apps are simple enough that the only input they need is from the intent to launch the app and basic touch events. The usefulness of such apps is dwindling. Your apps will need to do more than this, as the level of sophistication demanded by consumers continues to grow.

Capturing complex information of differing types takes more than just putting controls onscreen. A great experience requires you to execute the basics perfectly and then add sophisticated refinement. Just doing the basics isn't enough to be remarkable. This means going beyond the triviality of what's required and adding a level of smart logic to the app. Then, you'll create an app that people will find effortless and love to use. To help you create such an app, we'll dive into

- Understanding the goals of the people using the app
- Tailoring forms within the app to aid ease of use

In this chapter, you'll also hear about the experiences of Sam, who developed an app for his company that required lots of data entry. His app went through many changes to improve the experience of providing data, making it easier for people to provide accurate information.

6.1 The goals of the people using the app

In this section, I'll focus on ways to improve how you provide information as part of meeting the goals of the people using the app. If you remember figure 1.2 in chapter 1, you'll recognize figure 6.1 as a part of it. Here you see the interaction between the person and the app. The actions start with the person, their expression of intent, and subsequent input.

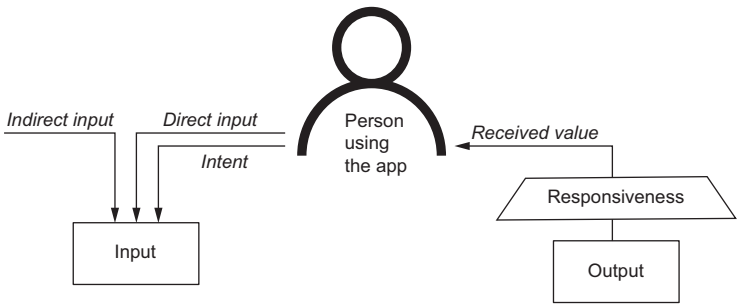


Figure 6.1 Apps get input in many ways, but the aim of each is to give value to the people using the app.

The primary goal of the people using your app is to obtain value. Whether this is finding out the latest news, listening to music, making a purchase, or being entertained, your app needs to meet their desires. Having people provide input and any work performed by the app stand between the person and getting that value. You have the power to reduce both.

Beyond a primary goal, your app and the people using it will have many secondary goals. The primary goal in using an app for a social network could be to keep up with what friends and family are doing. Secondary goals may include sharing their own updates, responding to what they're sharing, and gaining an awareness of relevant wider news. When using a music and podcasting app, the primary goal is to receive entertainment or information. Secondary goals might include the ability to record and track what's listened to or to share and comment on what's heard.

Secondary goals vary based on who uses the app, where and when it's used, and the device it's used on. But the primary goals of everyone using the app are for it to be as easy to use as possible and quick to respond. This may seem obvious to you, but it's something that often goes unstated. Because people rarely verbalize their desire for these goals, many apps overlook this. This provides an opportunity for you to set your app apart from others.

Jef Raskin worked at Apple and was a key player on the team that developed the Macintosh. He had strong views about software. In his book *The Humane Interface* (Addison Wesley, 2000), he defined two rules of interface design. His first rule, inspired by Asimov's first law of robotics, is an idea I'll cover later in this chapter; it's the second I'll focus on now:

1. A computer shall not harm your work or, through inaction, allow your work to come to harm.
2. A computer shall not waste your time or require you to do more work than is strictly necessary.

Consider the computer he speaks of as your app running on a device. The *time and work* referred to are what's required to input data by the people using it. To reduce the input required by an app

- Avoid requiring the input altogether
- Minimize the input requirement as much as possible
- Use an alternative input method

I'll explain how to minimize input in the next section. Then we'll look at defaults and suggestions as a way to improve the experience of data input for the people using your app.

6.1.1 Improve tasks by minimizing input

A general (and universal) rule of form completion is that the longer a form, the less likely it is that a person will complete it. Correspondingly, by making something easier for a person, they're more likely to do it. If you're asking for information in your app, there should be a good reason for doing so. If it makes the form longer, you risk not getting anything. This section describes three approaches to minimize the amount of time and effort needed to enter required input:

- Make it easier to provide the input.
- Reduce the amount of input by not asking for things you don't need.
- Remove duplication by not asking for the same thing more than once.

The simplest way to minimize required input is to only ask for something when it's needed or when you're able to do something with it. If you don't need to send a person something via email just yet, wait until you do before asking for their address. If a person gets value from an app without registering, don't force them to do so before using the app.

Sam's app had a long registration form that needed to be completed when someone first used the app. That's when the company assumed they had a captive audience. It saw this as an opportunity to gather as much information as possible. To do that, the marketing department added lots of fields to the registration process. The length of the registration form was highlighted when Sam began investigating why the

number of people who registered and used the app was so much lower than the number who downloaded it.

The volume of information requested in the registration form was seen by many as unnecessary, and raised uncertainty about what the app would do with all that information or how it related to the app. After receiving this feedback, Sam reduced the amount of information requested when registering, and the registration rate soared.

Like some of the people in Sam's company, you might think it's better to ask for more information earlier, so you have it when it's needed. This ignores two important points:

- You may never need it. In this case, you've asked someone to do something for no reason and wasted their time.
- You degrade the experience for the people using the app by asking for more than is necessary at the current time. This can decrease the likelihood they'll still be using the app when you do need the data.

Don't worry; people will provide the information you need at a time that's relevant and provides value to them. And, using the approaches mentioned in this book, you can improve the experience further.

Minimize the effort involved in providing input

It's useful to have a measure of the effort involved in providing input. One way to do that is to count the number of taps or touches needed to provide the desired information. This means how many times a person has to tap or swipe the screen, press a button, or click a mouse. It's not a perfect measure, but it's quick to calculate, easy to understand, and simple to compare results (lower is better). The following shows some approaches to reducing effort when providing required input:

- *Select instead of type*—If you need a piece of information that's one of a small number of options, provide buttons for each selection. A person only needs to make a single touch for any input, rather than several touches, depending on the length of the word.
- *Shortcuts*—Scrolling through long lists to find the desired value requires some touches impacted by the number of swipes that must be made to scroll to the value, based on its position in the list. Using a sorted list combined with fast scrolling, indexes, or jump lists gets a person much closer to the desired value with a single swipe or a couple of taps.
- *Auto-suggest/auto-complete*—Instead of typing a whole value (one or more words) or scrolling through long lists, you can reduce the total number of required touches if the app displays a shortened list of values. These can be filtered to those starting with (or containing) what's been entered. This allows a person to type the first few letters and then tap the desired value. It's usually shorter than typing the complete value.
- *Combine inputs*—Where two related values are needed, it's simpler to capture them both with a single control. For example, consider a need to provide starting

and ending dates. It's easy to have one calendar control for the start date and another for the end date. A simpler alternative is to have a single calendar that displays both dates: the earlier date would be the start and the later, the end. Ignore the order of selection, and there's no potential for the start date to come before the end date.

In another example, consider the need to specify a range between two values. A slider can represent the entire range. Using a slider control with two handles to select the start and end values within the range is simpler than selecting from two lists.

- *Auto-focus*—If an app presents a series of input controls for a person to provide data, and nothing else is on the page, it's unnecessary to first tap the field. Instead, set focus appropriately so a person can just start typing.
- *Auto advance*—If the app requires multiple values to be entered, once any values of a specific length are entered, automatically advance the focus to the next field. A word of caution: use this approach with care. Test its usefulness with the people using the app. If you're moving the focus when they don't expect it, or not moving it quickly enough, the confusion you introduce will outweigh the benefit of saving a keystroke.
- *Advance until submit*—When focus can't be advanced automatically, a person should still be able to move it from the keyboard. Consider a form containing three mandatory fields and a Submit button. In a worst-case scenario, a person
  1. Moves their finger to the first field to set focus
  2. Moves to the keyboard to enter the value
  3. Moves their hand to set focus on the next field
  4. Goes back to the keyboard for the second value
  5. Moves again to set a final focus
  6. Uses the keyboard to enter the last value
  7. Moves to tap the Submit button

  On a desktop, such a scenario is exacerbated by transitioning between mouse and keyboard. But even on small, handheld touch screens, changing hand positions for holding and typing on a soft keyboard at the bottom of the screen and tapping fields at the top of the screen is unnecessary. As a better alternative, use the Enter key to move from the first to second fields and then again to the third. When pressed while the focus is in the third field, it should perform the same action as tapping the Submit button directly. The worst thing for an app doing this is when a person can't set focus by tapping on a field directly. For the increasing number of people who expect such functionality, your app will meet their expectations and not frustrate them.

MINIMIZE INPUT BY REDUCING WHAT'S ASKED FOR

If less input is better, it follows that no input is best. Although it's rare for apps to require no direct input, this doesn't mean individual components for input can be

avoided. If you were to avoid input entirely, you could prevent people from achieving their goals and severely limit the functionality of your app.

The value of data to business has grown in recent years. Advances in data science and machine learning have increased what's possible and have made those prospects more widely available. Because of this, it's become desirable for businesses like Sam's to capture, store, and use as much information as they can gather. This can lead to apps like the one in figure 6.2, where unreasonably large amounts of data are requested and extend far beyond what's relevant to the app.

Figure 6.2 is an example of the conflict between what's best for the business creating an app and what's best for the person using it. Like Sam's app, the business wants as much information as possible, and the person using it wants to provide only what's necessary. Focusing on what's best for the person and not the business can be the

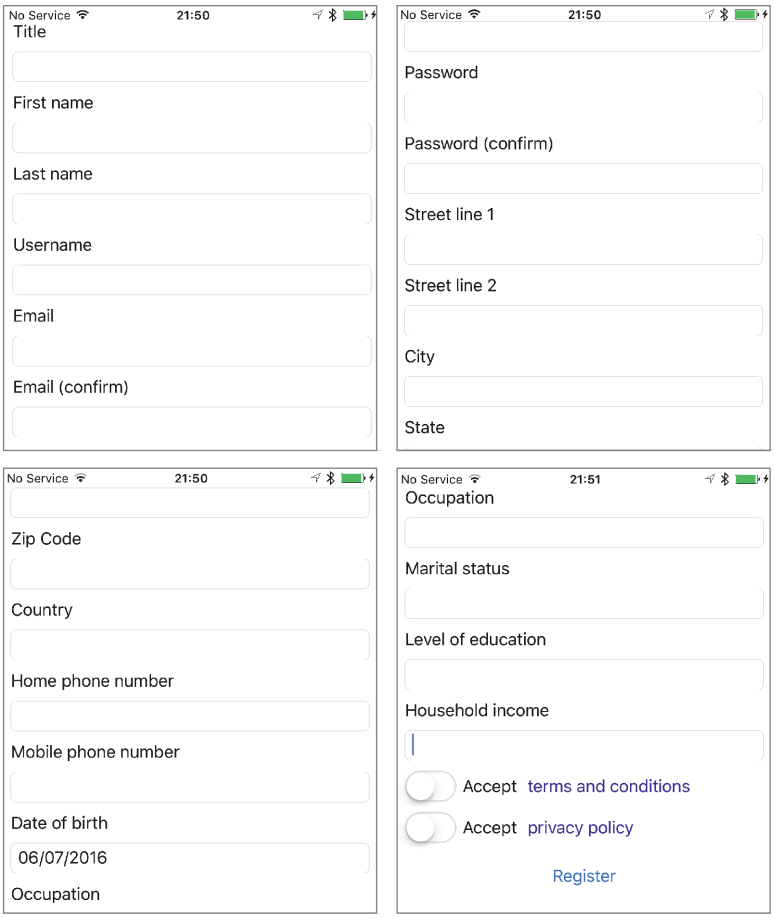


Figure 6.2 A large form spread over many screens is hard to use. Including fields not directly relevant to the app's immediate needs creates more work for the person using it and stands as a barrier to their goals.

difference between lots of contented customers or a smaller, less happy number whom you know more about.

There'll be times when the requirements of a business or the opinions of a stakeholder demand gathering specific input. In such scenarios, ensure that the business or stakeholder is aware of the potential negative consequences of increasing the amount of information that's asked for. The best way to make this clear is by providing data for your app. If you're able, test this by having versions with and without extra required fields (an A/B test). See how the inclusion of extra required fields affects how many people complete the form and the number of validation errors they experience. Once you're clear you want to avoid as much input as possible, you'll have to decide which input you do ask for.

> **NOTE** When all data is potentially useful, decide what to ask for based on what will add immediate value. I summarize the guidance this way: only ask for input that is necessary to perform a task that provides value to the person using it.

Notice the guidance is to only ask for what's needed. You may be tempted to use optionality as a compromise when trying to avoid imposing large input requirements on the people using the app. There are, however, some unwanted consequences to this approach:

- It isn't always obvious which parts are optional, so people will still experience the negative feelings associated with being shown a large form.
- Incomplete forms may be less valuable to the business. Having lots of partial information provides significantly less insight and is harder to work with.

Regardless of how much is optional, larger forms are significantly harder to create and accurately test. On almost every app I've encountered that includes a long registration process, it's this area of the app that has the most bugs. It's understandable to argue for providing input for all the information a business desires, but only make what's necessary required.

MINIMIZE INPUT BY AVOIDING DUPLICATION

As you strive to remove the unnecessary input in an app, duplicated input is an obvious place to start. Consider these three ideas:

- Not asking for something you can calculate or infer
- Not asking for something twice as a way of detecting accidental typos
- Not asking for something to be retyped unnecessarily

Remove the need to enter something you can calculate or infer from another response. For example, don't ask for a person's age and date of birth when you can calculate one from the other. Or, don't ask if a person is employed and for the name of their employer when "none" for an employer's name shows they're not employed.

Asking for the same data twice is also an obvious candidate for unnecessary input to remove, but doing so requires addressing the reason the duplication exists. The most common reasons are to avoid accidental mistakes or typos when entering email addresses and passwords. It's important these values are entered correctly, but asking for them twice has issues:

- Many people copy the first value they enter, including any mistakes, and paste it into the second field.
- It can be an excuse to not provide more sophisticated analysis of the value entered or to address common mistakes. Telling someone they haven't typed the same thing twice isn't as helpful as pointing out or automatically correcting:
  - Common domain misspellings
  - Use of a comma instead of a dot
  - Commonly confused symbols (quotes, apostrophes, or the number 2 instead of the at [@] symbol)

  You can also identify possible errors based on other information, such as highlighting if I entered `mtt@example.com` (note the missing *a* from the local part), when I've already said my name is Matt.
- If used with passwords that are obscured when entered, duplication doesn't address the issue of not being able to see what was incorrectly entered. (More on this later in the chapter in section 6.2.5.) Avoiding the entry of incorrect credentials can be an excuse for not optimizing the recovery process. Even if previously entered correctly, data can still be lost or input forgotten, so recovery handling is essential.
- Having two fields takes up more space. Having a single, larger display of the entered value makes mistakes easier to spot and to correct without prompting.
- Duplicate entry tends not to work well with platforms that support the automatic completion of forms. The app ends up working against the platform's attempts to make input easier.

If you want to know if having duplicate entries makes things easier for the people who use your app, track

- How often errors are encountered during validation
- How often people edit what they're entering
- The number of invalid or unusable email addresses you receive

The most impactful issue with duplicate content is with apps that lose or forget what's entered. It's normally easier for people to correct a minor error than to retype the whole thing. Avoid the following:

- Clearing previously entered data when people set focus on a text input control
- Automatically removing data that fails validation

Save all data entry, including partially completed forms, so the same data doesn't need entering again. For example, if a person is part way through typing a message to someone but leaves the app to answer a phone call, on completing the call and returning to the app, they'd expect to be able to complete the message. Have the input there when they return.

> **NOTE** Passwords can be an exception to saving all data, depending on the security requirements of the app.

It's possible the person may not want the saved data. If so, give them a way to clear it (figure 6.3). This is better than forcing them to enter it again.

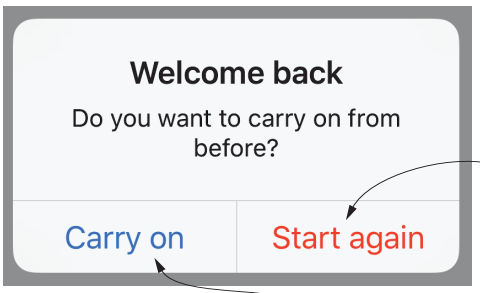


Figure 6.3 Prompt people on returning to an app in which they'd previously entered data rather than assuming they want to keep or delete it. This keeps them in control and avoids unnecessary duplication.

In our example, if the person using the app chooses to save the partially completed message, you must make it obvious how to access and retrieve the draft. If the app is used by the same person on multiple devices, having drafts roam so they can be started on one device and then finished on another is an added way to help the people using your apps.

6.1.2 *Improve tasks with defaults and suggestions*

You'll want to avoid unnecessary input, as that's desirable and preferable for the people using the app. By requesting unnecessary information, you increase the chances the people you're asking will be put off by the volume of data, decide not to provide any, and go elsewhere.

Even if a person has no choice but to use the app and must provide the information that's asked for, the request to provide what they see as unnecessary or irrelevant creates frustration and disappointment. The key word here is *unnecessary*. You don't

want to, and probably can't, remove everything. If it doesn't contribute directly to the functionality of the app or service, it's not needed. Be careful, however, that you don't remove the ability for people to provide the information they want to give you or the business needs.

> **NOTE** The more you ask of a person, the less likely they are to do it successfully. Any such failures reflect poorly on the app because it makes the person using it feel bad about themselves for not being able to use it successfully.

You can avoid input by not asking for things you can make an informed guess about. Here are two common scenarios:

- Configuring default settings when first using an app
- Making suggestions for what to search for

Sam's app has many configurable options affecting how it behaves and how it structures what's displayed. Some of the people at Sam's company had strong but differing opinions about which options should be the defaults. The compromise was to ask the people using the app which options they wanted when they used the app for the first time. Many people who used the app didn't have the knowledge to know which choice was best for their needs, and so ended up choosing sub-optimal options.

This issue became apparent when Sam looked at the support requests and negative store reviews the app was getting. Many issues were due to the configuration options people selected. The app was changed to remove the requirement to select configuration options on startup. Sam selected default settings based on the most common usage scenarios as indicated in the analytics data. This not only made it easier for people to get the desired behavior from the app; it also removed the delay in getting value from the app on first use.

Making an app customizable or allowing personalization of the way an app looks or performs is often a desirable feature. Some people like to change color schemes, control the frequency with which new content is loaded, or arrange the order in which items are listed. Forcing a person to make decisions about every setting when they use an app for the first time is unlikely to be valuable or useful.

Rather than having the person using the app specify settings they don't have sufficient knowledge or experience to answer, provide sensible defaults and inform the person that settings can be changed if desired. Many people choose not to customize or personalize the apps they use. Presenting options they'll skip is an avoidable obstacle to obtaining value from the app.

Search is another common feature easily improved with a zero-input approach. Where identical searches are often performed, either by other people or the same person, you can save them from having to type or retype a term by automatically displaying recent searches, as well as popular or trending terms.

6.1.3 Improve tasks with alternative inputs

Faster experiences lead to happier users because they get the value they want sooner. Measuring how long it takes to get a desired result and the number of clicks/taps/touches taken to get there and minimizing both is great. You must also consider how conditions vary and the specific contexts of use for your app. For this, look at

- Alternatives to typing
- Manually and automatically correcting mistakes

Here's what Sam did. Sam introduced support for alternative input in his app in two ways: autocompletion and speech-to-text input. Sam used text fields for entering one of a large, but fixed, number of options, such as country. These were changed to provide suggestions after a few characters were entered. Whereas typing was seen as preferable to selecting from a large list, only having to type some of the text was better still.

In Sam's app, fields for capturing potentially large amounts of text, such as the app's feedback-gathering system, were also changed to support spoken entry on devices that had microphones. This was captured if speech to text was available. Otherwise, an audio file was submitted instead of the text, and it was converted to text on the backend.

Even when touching the screen is the quickest way to enter data into an app, it isn't always possible or practical. Cold and wet conditions or a small screen make it hard to type accurately. In these scenarios (and more) and where the device permits, you improve the experience with the app by supporting text entry in these ways:

- *Typed with keyboard*—Supported by all devices, but it's not always the most practical. This is normally the best choice for fine-grained alterations and editing.
- *Dictation (voice input)*—Can be faster and more accurate than typing. But it produces mixed or no results with some languages and accents, or in noisy environments. Potentially limited by the network connectivity needed for translation and is hard to edit.
- *Handwriting (with stylus)*—Can work with a finger but normally works better with a stylus. Accurate interpretation of input can vary greatly, and editing input is often easier with a keyboard. Works well if already using a stylus with the app or when entering text that's hard to input with a standard keyboard, such as mathematical formulas.
- *Optical character recognition (from image or camera)*—Requires a network connection and good lighting if using a camera. It's most effective for large or complex text entry.

Whatever method people use, it's necessary to be able to correct any mistakes. Even better than making it easier for people to correct invalid entries is for your app to cope with errors. Also, people expect support for partial matches and intelligent

suggestions for alternatives from web-based searches. There's no reason your app, like the one in figure 6.4, can't do this.

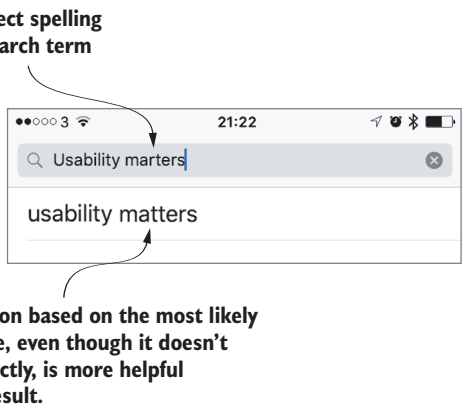


Figure 6.4 Make searching for something easier by being smart about incorrect entries and providing results based on likely or common misspellings.

Sometimes the best alternative input is an entirely different approach. The most common example of this is in capturing a location. The text-based approach is to type an address or location name, but alternatives also exist. You can

- Use a geographic lookup based on the public IP address of the device.
- Use the location settings on the device to provide the location.
- Allow a person to specify a location on a map.
- Select an address from a contact in the device's address book.

Put it into practice: align your form with user goals

Do the following to ensure you make your forms as intuitive and easy to use as possible:

- Identify everywhere data is entered or changed in your app.
- Understand the business implications of removing any optional or unneeded fields. Remove all that the business doesn't need, and then remove anything that's duplicated or can be combined.
- Don't ask for what you can get automatically or from the device.
- Auto-advance the focus on any fields that have a fixed length when that length is reached.
- Support use of the Enter key to advance focus through a multi-text-field form.
- For any pages, screens, or views in which the only task is to enter text, set the focus appropriately on accessing the page.

Exercise: minimizing input

Test your understanding of the last section by stating whether the following are true or false:

1. Prefer shorter forms to improve the likelihood they'll be completed correctly.
2. Ask for important information to be entered twice to avoid and identify any typos during entry.
3. Only include default selections from a list if there's a strong likelihood that they'll be correct.
4. Avoid asking for information you can easily obtain from other sources.
5. Allow text entry for all information required.

6.2 How to ask for data to be entered in forms

This section shows how to ensure you'll create a good, intuitive experience with the forms in your app. These include forms for login, registration, search, data capture, sharing content, and settings. I use this broad definition for a *form*—anywhere data or values are entered or adjusted in the app.

It's common for people to blame themselves when filling out a registration form or completing a purchase that is hard or goes wrong, but human error is often the result of design failure. Whether accidental errors or mistakes are due to something being misread or misunderstood, your actions can prevent this. To make the entry of data into your app significantly more intuitive

- Optimize the layout of the form.
- Capture text.
- Capture and process passwords.
- Specify a value in a range or set.
- Indicate and validate optional and mandatory data.

I've yet to meet a single person who's never complained about a website's registration process, or struggled to check out when buying something online. It's not only websites that have such issues; traditional desktop applications can be just as bad. Even if you ignore the differences between platforms, I'm sure you appreciate the idea that building mobile apps like desktop apps and websites isn't a smart move.

6.2.1 Optimizing how the form is arranged

In the first version of Sam's app, all form fields were the same size onscreen. The simplicity of this was considered visually pleasing, but it caused some issues that affected usability. It made it hard to view or edit all the entered text if it was longer than would fit in the space provided. This led to the frequent entry of incorrect and incomplete information. To address this, Sam provided appropriately sized fields so people could see the information they entered.

If all things were equal, being able to fit all parts of a form onscreen at once is best, but this is rarely practical. Consider three areas key to an optimal arrangement of the elements in a form:

- Support for scrolling and different orientations
- The ordering of fields
- The position and inclusion of labels and hints

Without the need to scroll, all parts of a form must be visible. Trying to squeeze a lot into a screen in a way that scrolling is never necessary is tempting; doing so often compromises both aesthetics and usability. On a portrait screen, it's impractical or unrealistic to do so without the need to scroll. With the device in a landscape orientation, space is further reduced (figure 6.5).

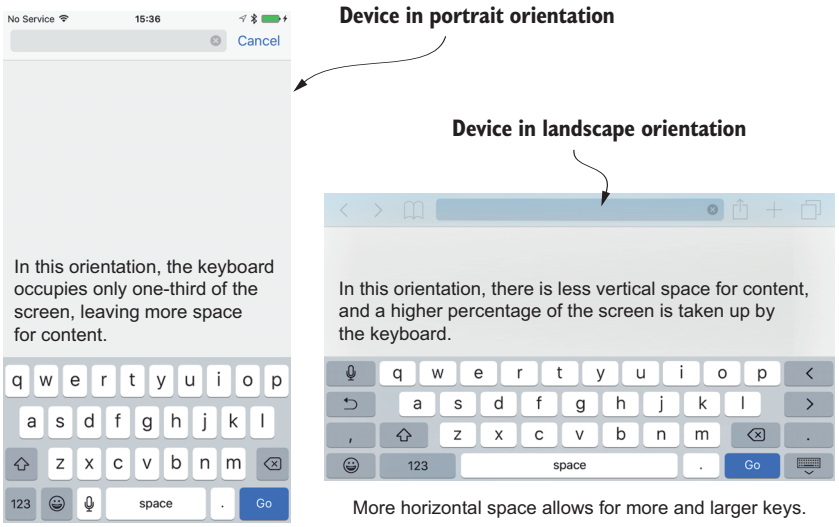


Figure 6.5 Screens displayed in portrait and landscape orientations show the differences in keyboards and available space for content.

Don't use the limited space on a landscape screen as an excuse to not support it. Landscape orientation offers the following benefits:

- Space for larger keys means fewer typing mistakes.
- More space allows for extra keys, making it easier to edit or change the text.
- Holding the device in this way can be more comfortable if using two thumbs to type.

Whichever orientation you use, ensure that it's possible to navigate and understand the position within the form. Keep the label of the focused control in view, and show the next field too. You don't want the person using the app to be unaware there's

another field they haven't gotten to yet, or to be unable to get to a button that's behind a keyboard. To require a person to tap another part of the screen to dismiss the keyboard so they can get to the next field or a Submit button is frustrating and unnecessary.

Arrange the contents of a form in a single vertical column. Position clear, simple labels above each field. As shown in figure 6.6, having the label beside a control limits the space for the control, making it harder to see or select all that's been entered.

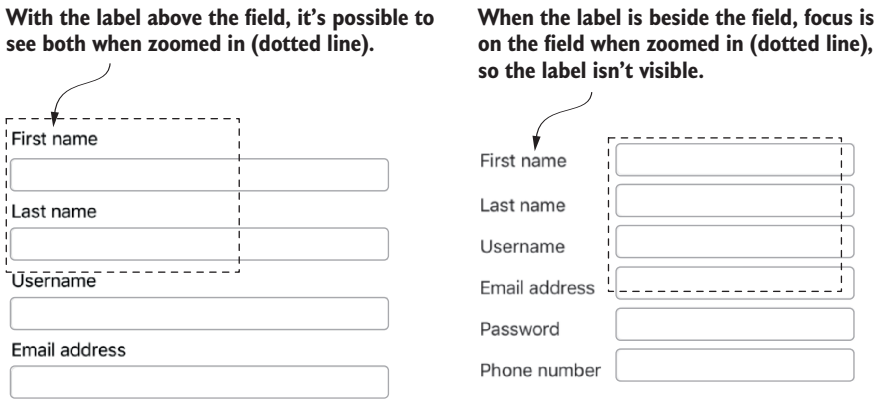


Figure 6.6 Placing labels above instead of beside the input controls to which they relate leaves more room for the control and improves usability with assistive tools or with increased text display size.

Even when displayed on a much wider screen, the vertical stacking of labels and fields is preferable. This makes it possible to see both labels and fields at once when the screen is zoomed in or the text size is enlarged. Using a single column also avoids any issues or confusion about the order in which fields are expected to be completed, and how the focus is advanced. Also, ensure that the field with focus is indicated clearly so people know where the data they'll enter will appear and what the app is expecting them to enter. Use a flashing cursor and a change in the border of the focused control to make this clear.

Placeholder or hint text can also be used as an addition to a label for a specific field. It needs to be styled so that it's distinguishable from actual values. People need to be able to easily differentiate between the fields they must enter data in and those already populated. This is important if the app or OS can automatically fill in part of the form for the person using it.

Using a placeholder instead of a label reduces the amount of space needed and is appealing on a device with limited space. But this not only loses the benefits of having labels, it also requires more concentration when entering data. It negatively affects accessibility by reducing the size of the area that can be touched to set focus (as touching a label should set focus on the associated field), and requires more effort to work

with screen readers. The top example in figure 6.7 shows how useful information is hidden when placeholders are used in place of labels.

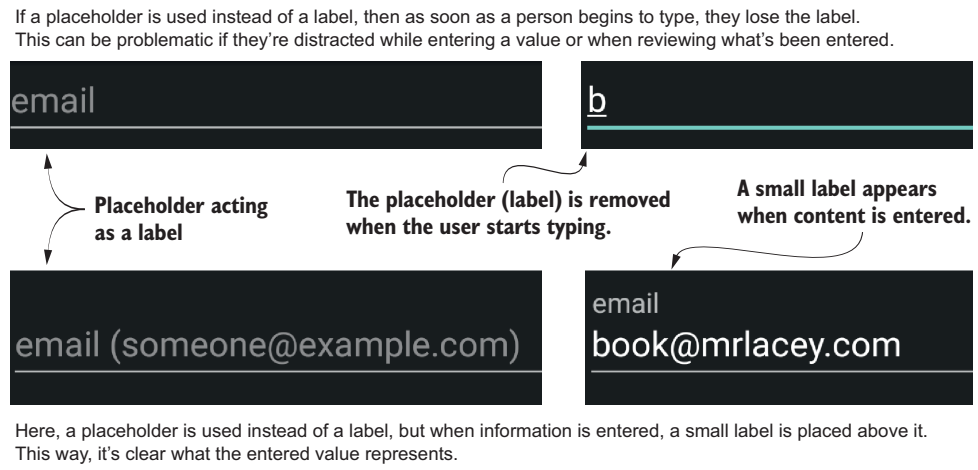


Figure 6.7 Using placeholders instead of labels makes it unclear what a value represents once entered. Adding a small label above is one way to address this.

The lower example in figure 6.7 uses the *float label pattern* to provide samples or additional information. Then the placeholder becomes a small label once the field has focus or content. This is increasingly popular as a compromise between requiring labels and the space saved by not displaying a label. When using this technique, consider these questions:

- Are you providing useful information in the hint?
- Is a hint the best way to provide information?

Showing an example of an email address may be useful to some people, but it can seem patronizing to others. What information or example to give depends on what's appropriate for the people the app is intended for. Remember, not everyone is like you. Some people require that everything is spelled out and made as simple as possible. In general, doing something that benefits some but not others is still worth doing.

The second consideration is whether a hint is the best way to provide information. Hints are often used to indicate that formatted data must be entered. If you need information in a specific format, such as formatting characters around parts of a phone number, the use of input masking or automatic formatting ensures that you get these values as you want.

Removing a hint while data is being entered makes it harder for the person entering the data to know they're doing so correctly. This means you must redisplay the information from the original hint when showing something has been entered incorrectly. Keeping the hint constantly displayed as part of, or next to, the label avoids this issue.

> **NOTE** There's a simple rule for defining the order in which information should be asked for: ask for information in the order it's traditionally given, unless a different order will make it easier.

Asking for a person's last name before their first name can cause them to pause and wonder about the order. You should avoid even such small amounts of friction, as it's unnecessary. In another example, a zip code is normally entered at the end of an address. If an app can look up addresses from a zip code, then you could ask for the zip code and present a list of addresses for the person to select from. It's quicker than typing a full address. This variation from the normal order of an address is common and so is unlikely to confuse the person entering the information.

In addition to the judicious use of hints and the order of fields in a form, the wording used is important too. Use clear terminology and language appropriate to the people using the app. Use terminology consistently within the app and any associated website. Don't, for example, ask someone to define a password during registration and then ask them for a PIN code when logging in. And, don't require different credentials for the same account on different platforms. Avoid jargon or complicated and inconsistent language that won't be understood by the people using the app. Figure 6.8 highlights some of these issues.

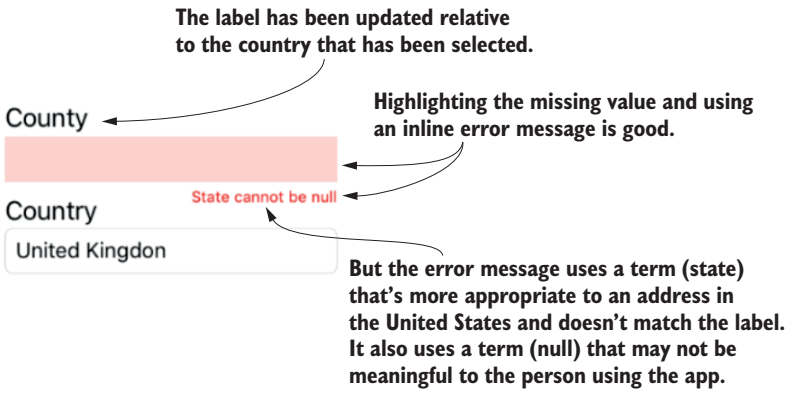


Figure 6.8 Part of a form showing inappropriate jargon and inconsistency in an error message, both of which are to be avoided.

Consideration for the people using your app shouldn't stop with the language used. To not allow for variation in the accessibility requirements of the people using your app is to do them a disservice. Make your app fully usable with common accessibility tools, including screen readers, color-blindness settings, and enhanced zoom.

The transit test

One way to evaluate how practical it is to use your app and particularly any forms is with the *transit test*. This simple and unscientific test asks one question: "Can the app be used on a crowded bus or train?" Although this isn't the only environment where your app is used, it's a challenging one. If the app is usable there, it'll be usable almost anywhere.

This test serves as a loose proxy for general usability. If a more advanced or capable user can successfully use the app in difficult circumstances, anyone with additional requirements should be able to use it in a less challenging environment.

6.2.2 Simplify how text is entered

This section looks at three ways to improve the accuracy, speed, and simplicity of entering text. I'll cover how to

- Make it easy to enter correct data.
- Make it hard to enter incorrect data.
- Make it simple to provide multiple pieces of data.

Typing on a mobile device isn't fun. It's much easier to type on a full size keyboard than a small one. This doesn't mean that people don't type on mobile devices, but they do so differently. I've written parts of this book on a phone and a tablet, but I wrote most of it on a laptop because doing so is easier. Correctly formatting the text on the phone is hard, and so I avoided doing this as much as possible.

I once had to make a series of complex annotations and modifications to a presentation while on holiday, with access to only a phone. Doing this took a lot longer than if I had access to my laptop, and it repeatedly felt like an exercise in frustration. What this means is that we cannot argue that people don't or shouldn't use a specific type of device for a task. People will use whatever devices they have available, even if it's not best suited to the task. So it's in your interest to provide apps that make it as easy as possible for people to complete the tasks they're trying to do.

Make it easy to enter correct data

Small devices don't have the space to show as many keys on a keyboard as a larger device, and so can show one of many onscreen keyboards comprised of different options. By choosing the keyboard most appropriate to the information that's being asked for, you reduce the number of touches needed to enter a value. Figure 6.9 shows this.

Exposing all the required keys on the soft keyboard prevents the person using the app from wondering how to access a specific key or enter a specific character. Such a pause is small, but you should avoid any barrier to a person getting the value they're after—no matter how trivial.

In addition to the keyboard displayed, use other options as necessary for text entry. Facilitate the capitalization of the first letter, no letters, or all letters as appropriate for the field. For example, proper names and sentences should start with capital letters,

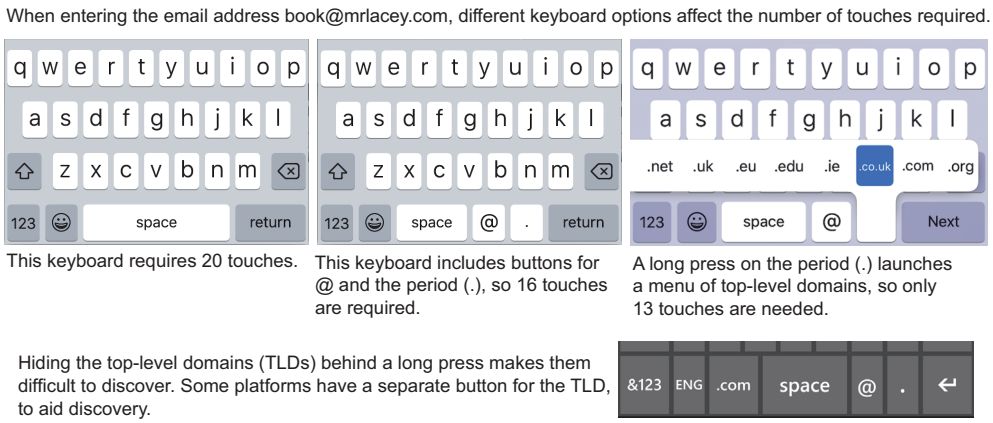


Figure 6.9 Different keyboards make it possible to enter the same value with fewer taps. Use the one that enables the fewest touches for each field in your app.

so have this enabled. Disable this if requiring case-sensitive input. Also, enabling auto-correction, auto-completion, and spell checking are useful additions if the entered text is likely to comprise common words or proper sentences. Conversely, if the entry doesn't contain common words, such as a username, then this isn't helpful.

MAKE IT HARD TO ENTER INCORRECT DATA

If only certain values or characters are acceptable, prevent the entry of anything invalid with an input mask or by handling events when the entered value changes. Avoid trying to do this with key-based events; these won't work as expected if pasting text into a field. For long or complicated values, such as phone numbers or credit card numbers, specific formatting makes those easier to read, and, therefore, spot errors. Controlling formatting and limiting the characters accepted makes entry easier and still benefits readability.

If you're unable to use controls with built-in capabilities for masking input, you can re-create something similar by doing the following:

- Automatically adjust capitalization as appropriate. If you want all letters uppercase, accept any case but change anything entered in lowercase.
- Automatically correct common errors or substitutes. This might be the letter *O* instead of the number zero (*0*) or a hyphen instead of an underscore.
- Ignore any characters that are definitely invalid by doing nothing when they're typed.
- Add formatting automatically. If you want a space between the third and fourth characters, add one even if it's not typed.

To the people entering data, aim to provide the freedom to do so as they wish, yet consistently display correctly entered information. This will provide yet another reason for people to continue to use your app.

Make it simple to provide multiple pieces of data

As mentioned earlier, using the Enter key to advance the focus through multiple fields on a form can make the process faster. Better still is when you combine multiple separate controls into one. For example, don't have separate fields for the different parts of a phone number or an address; use a single field, and decompose the respective parts as needed. Likewise, you shouldn't need the country or area code for a phone number to be entered in separate fields. Instead, allow the person to enter the phone number as needed, and then use some simple processing to break up the number and reformat it as necessary when saved. By ignoring any spaces or formatting characters, it's easy to tell if the entry is valid based on the length of the entry and optional inclusion of a country code. Figure 6.10 shows a comparison of these two entry possibilities.

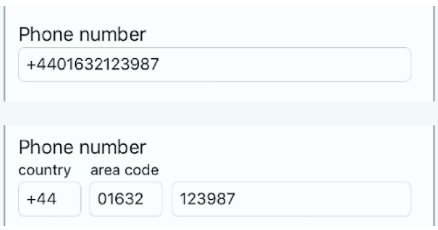


Figure 6.10 Phone number input as single and multiple fields. A single field makes entry simpler, without losing any ability to validate what's entered.

Addresses are another common example of multiple fields that you can combine to make data entry easier. For the parts of the world with structured address systems, an address can contain many optional components. This leads to many unused, and, therefore, unnecessary, fields, or to people having to think about how to enter an address in the fields presented. Showing a simple, multiline text field as in figure 6.11 is a simpler alternative. Any address validation or lookup tool should be equally capable of coping with addresses provided this way, too.

The ability to cope with semi-structured data given as freeform text is also beneficial for data entered through a speech-to-text interface. With that functionality, it's simpler and preferable to dictate an address all at once, which is hard when the UI requires that each line be entered separately. The ability to accept even more complicated, semi-structured data allows for further possibilities.

Imagine an app that allows people to create reminders. The app needs to know what to remind the user about, and the date and time to do so. Rather than display a text box for the reminder and separate controls to select date and time, allow a single piece of text entry such as, "Remind me about Bob's party at 7 P.M. next Tuesday." Then you can extract and convert the appropriate parts and act on them. Not only would this make speech-based entry easier and more powerful, but it'd also make it

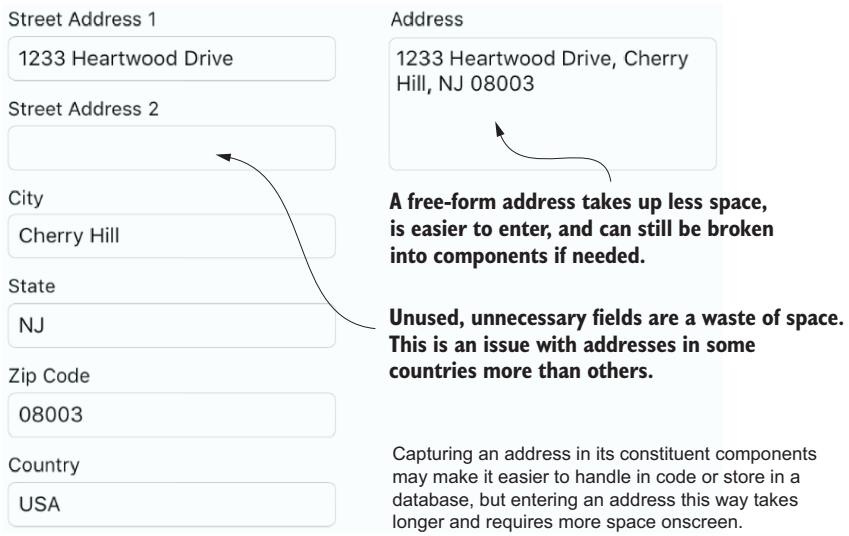


Figure 6.11 Comparing address entry as many fields and as one field. A single field uses less space onscreen and provides flexibility for how it's entered.

simpler to integrate with voice-based input via the OS or with text-based input from a chat interface—both of which are popular areas for future app growth.

6.2.3 Password entry requires special consideration

Every time someone wanted to use Sam's app, they were required to sign in with the same credentials as on the website, which had strict rules about password complexity. Correctly entering complex passwords was an issue for many users of the app, so the company allowed the creation of a numeric PIN value that could be used instead of a password. The PIN was tied to individual devices to prevent its use elsewhere.

When it comes to entering them, passwords are a special form of text entry. They suffer from all the same challenges of accurate typing as regular text, but with the added challenges of being unable to verify what's been typed and a need to meet arbitrary added restrictions. Mobile security is a massive subject and more than enough for a book on its own, so this section focuses only on the entering of passwords.

The entry of a password is as much about perception as it's about security. Past use of computers has taught people passwords shouldn't be displayed onscreen, where someone could be looking over your shoulder to see what you've entered. But if someone was looking over your shoulder, they could see which keys you pressed on the keyboard, so there's little to be gained by hiding the entered text. Think of it like an ATM, where you're reminded to "cover the keypad while entering your PIN."

This doesn't mean you shouldn't obscure the entered password. You'll make people using your app question its security by breaking their expectations if you don't

obscure it in some way. What you do need to do is address the issue of not being able to easily verify the entered values.

As it's so easy to make a mistake when typing on a mobile device, add a way to view a password while it's being entered. This allows a person to check what they've entered without the round trip of submitting the form. Without this, a person who suspects they've entered it incorrectly must delete it and start again, or guess how many characters to delete and then re-enter those. Figure 6.12 shows ways of doing this.

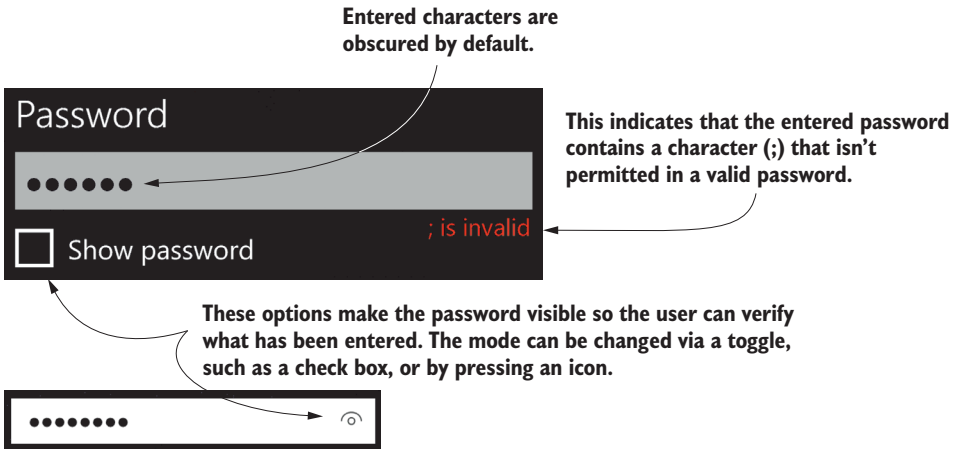


Figure 6.12 Password entry is made easier by providing a way to view what's entered and indicating if what's entered is invalid.

Making the password briefly visible does lessen security a bit, but the password is already visible if someone is watching you type. The benefit of improved usability far outweighs the small drop in security. But, as is often the case, you must find a compromise between security and usability that's right for your app.

You also make it easier for a person to accurately enter their password by verifying the entered value against any basic rules enforced when creating a password. People use different approaches to manage their passwords, but these don't always correspond with the constraints an app, site, or service imposes on what can be used for a password. If a person enters a password containing a character that's unacceptable, or because they thought it was part of the password, tell them so.

There's a misconception that this compromises the security of your system and makes it easier for someone trying to guess passwords. If there's a public registration process for the app/site/service, and this enforces rules about what's acceptable, then make public what those constraints are. Using this information to help people avoid accidentally entering an invalid value doesn't affect the security of the system in any way.

Similarly, when asking for a password as part of a registration process, show any rules that exist for the format of the password as or before it's entered. Don't wait for a person to submit all their registration data and then tell them their password must contain (or not) certain characters, symbols, or combinations thereof.

App vs. device security

Should a person have to re-enter a password every time they launch your app? If the app has highly sensitive information or makes it possible to perform irreversible, highly consequential actions, such as a banking app, then repeated password entry might be needed. For most other apps, it's not. With the wealth of capabilities on and information available in a modern device, far worse things are possible with a stolen, unlocked device than the use of an app. If you feel it's necessary to have password protection for each use of the app, consider only doing this if the device doesn't have a PIN lock or passcode enabled. This gives your app the extra protection you want, but without forcing a person to sign in twice.

The security of a system and the ease of use of an app may conflict. If you lower security for the sake of increasing ease of use, have the person opt-in to this behavior. Here are some ways to increase the ease of use:

- Allowing gestures instead of passwords
- Using biometric alternatives to passwords, like facial recognition, iris scanning, and fingerprint readers
- Supporting integration with third-party password managers
- Allowing the pasting of password values from the clipboard
- Advising on what makes a strong password without forcing its use

Forcing someone to use a password more complicated than they want will cause them to write it down and compromise its security, or it'll mean it's forgotten more often, and so the person must use the forgotten password process many times.

Forgotten passwords

When someone must reset or recover a forgotten password, they're already far from the golden path that leads to the value they're after. Because the process has become more than just logging in, it's more complicated than desired, so it's important that this experience be as easy and simple as possible. Sadly, most developers pay little attention to this process. Spend time on this functionality to try to help the person who must use it. They're likely already frustrated by having forgotten their password, so don't make things worse with the experience you provide.

Security recommendations are frequently updated, making it inappropriate and impractical for me to try to give you specific advice on how to fully manage passwords and

security in your apps. That said, the following are universal points and should be observed:

- Passwords shouldn't be recoverable.
- Never store a password in plain text. If you must store a password, combine it with a unique salt value and encrypt it with a one-way hashing algorithm.
- Encourage the use of strong passwords or passphrases.
- Use multi-factor authentication where possible.
- Don't write your own encryption algorithms, and use the strongest algorithms available.

When it comes to entering passwords, allow for future changes to the encryption algorithm you use. You don't want to become tied to something that, in the future, can be compromised.

6.2.4 Simplifying entry from a fixed set of options

Forms will often need to capture more than text. As mentioned earlier, when looking at minimizing input, it's often easier to select from a list rather than type the desired value. It's not enough just to present a list of options. Make choosing from a list easier by

- Making intelligent use of defaults
- Simplifying the list
- Making the most likely options easiest to find

Although these ideas are simple, each can make a big difference. We'll look at these in more detail, beginning with defaults.

Simplify entry with default values

Where a selected value impacts how an app behaves and the experience a person will have, the opportunity to set a default value is a privilege. The privilege is to control the experience that people will have with the app and how they'll interact with it.

Keep track of the values people choose. If you don't have a default but a majority of people choose a particular value, make that the default. Or, if you do have a default but people are regularly changing it, you may need to consider changing the default.

Many companies assume a large section of their user population work as accountants, live in Afghanistan, and were born on January 1st, 1970. These are all the default values or first in the list for occupation, country of residence, and date of birth in those apps. That these values are so popular highlights factors to be aware of when considering default values:

- If they don't matter to the person completing the form, default values will be used regardless of their appropriateness.
- People can overlook automatically populated data and the need to change it.
- Because data is submitted doesn't mean it's correct. Validate the input, especially if multiple defaults are submitted.
- If no value is provided (such as for date of birth), don't assume a default.

Automatically selecting a default is one way to help people avoid needing to evaluate the options in a long list. Another way is to simplify the process.

SIMPLIFYING SELECTION BY REDUCING OPTIONS

With many options for a person to select from, you make it harder for them to choose one. People need to evaluate all the options and choose the most appropriate, which takes time. This is defined by Hick's law: with more choices, it takes longer to come to a decision. To simplify selection, keep these points in mind:

- For some selectable values, common options exist that are chosen far more often than others.
- Provide quick access to the most recently used options to aid faster and more convenient selection, especially if a list will be selected from multiple times.
- If displaying a long list of options, track which are selected and remove any that are never chosen.

To remove unnecessary options, only ask for the needed level of specificity. If you don't need a fine-grained level of precision, don't ask for it. If you want an approximation of the industries people work in, you could have them choose from a small number of industries rather than a much larger number of possible job titles. If you want to know roughly where in the world people are based, you could ask them to choose from a list of five (or six) continents rather than from 200+ country names.

When it comes to estimating time, people gravitate to the "time anchors" of 1, 2, 3, 5, 10, 15, 20, and 30. If your app requires the selection of a time duration, having these 8 options will satisfy most people and will be easier to use than a list of 30 (or more) durations. When precision isn't needed, having a small number of values in a range will be easier to use than a large list of specific values.

MAKE LIKELY OPTIONS EASY TO FIND

The final factor that impacts the ease with which people can find their desired option in a list is the order in which the items are arranged. The most appropriate ordering will be dependent on the items in the list, but is likely to be one of these:

- Alphabetical (from A to Z is normal)
- Numerical (from smallest to largest or vice versa, with most common nearer the top)
- Time sequence (from shortest to longest or vice versa, with most common at the top)
- Hierarchical (arranged with most appropriate nearer the top)

Figure 6.13 shows an example of multiple factors combined to make selecting from a list as fast and simple as possible.

When selecting from a list, it shouldn't be possible to select an invalid option. And it's possible that no item will be selected when one must be. In the next section, we'll look at how to verify what is entered and how to indicate if it's missing or invalid.

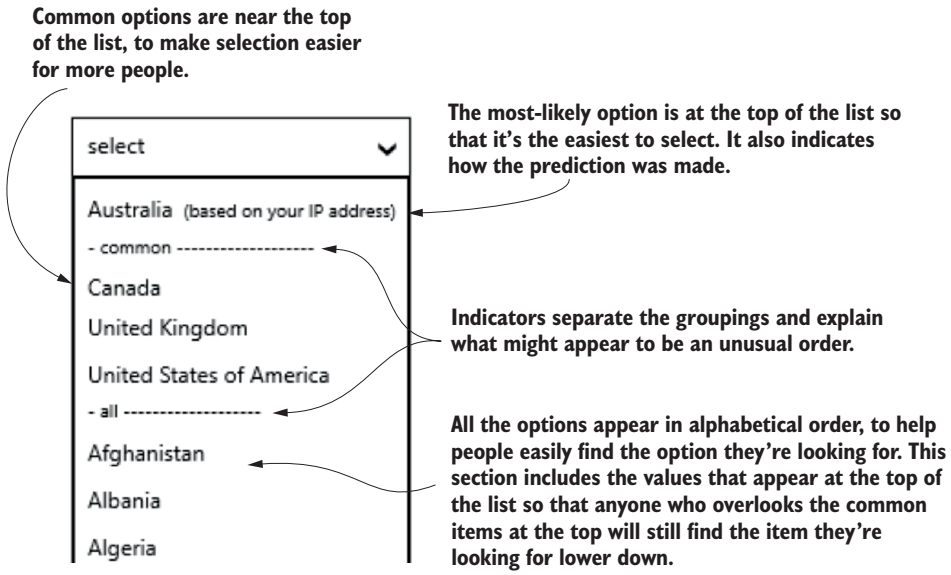


Figure 6.13 An optimized drop-down list for selecting country. It includes the use of external information to highlight the most likely option and common options. It also sorts all options to make the desired value easier to find, if not one of those called out at the top of the list.

6.2.5 Validation and required fields

The main data-capture form in Sam's app had many complex validation and requirement rules. Because it was only possible to do some of the validation on the server, it was decided to do all of it there. Missing or incorrectly formatted entries wouldn't be reported until after a round trip to the server had been made.

The consequence of this was that less than half of people were able to complete the form on the first attempt. Also, a large percentage of those who failed on the first attempt didn't try again and so never used the app, causing the company concern. The first-submit success rate was greatly improved when the indication of missing but required fields was added to the form by Sam, along with the addition of all the validation that could be duplicated in the app.

Postel's law

You may be aware of and follow the guidance of Postel's law:

> Be conservative in what you do; be liberal in what you accept from others.

But it often leads to more work for you and your app. To provide a robust experience for those using the app, this approach must be combined with Raskin's first rule of interface design (which you saw at the beginning of this chapter).

Because your app may receive a wide range of potential values, it's necessary that you verify that what's been provided is acceptable and advise if it isn't. You want to prevent any negative consequences that result from the data provided, both in the app itself and any backend system it passes the data to. This covers everything from ensuring an email address is in a correct format so it can be used to communicate with a person to preventing a potential path-traversal attack on a backend server through the submitted data.

Even if you've acted to minimize the cause and possibility of errors, they may still occur. I've said many times already that mistakes are easier to make on mobile devices, so you need to be forgiving:

- Ensure that any mistakes a person makes are easy to discover and correct. Highlight any issues while a form is being completed.
- Make it clear which information is required and which is optional.

Having an asterisk next to some fields without making it clear what the asterisk represents isn't sufficient. At the least, include the meaning of the asterisk and make it a different color so that it stands out. Also, because people will often want to do the minimum required to get the value they seek, not showing required fields creates a poor, unintuitive experience. You shouldn't leave them guessing which data they must provide and which is optional.

If a field has been skipped or populated with invalid data, indicate this when advancing to the next field. Don't wait for them to get to the end of the form before pointing out the issue. If the data entered into the form will be submitted to an external service for validation, do as much of that validation on the device as possible. This lets you inform the person using the app of any issue without the expense and delay of making the server call.

When a problem is identified, report it inline with a clear indication of the mishap and what's required to address it. Although a summary of issues may seem appealing, it makes the problem and its resolution less obvious. This is more noticeable where the form is larger than fits onscreen. A summary error message at the top of the form when the bottom of the form is visible is of no help. It may not be clear any issue exists at all. Equally unhelpful are error messages displayed in a dialog, like the one in figure 6.14.

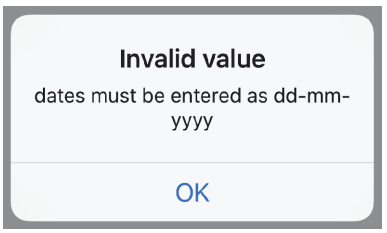


Figure 6.14 A dialog is a poor way of showing a message about invalid data that's been entered. It's shown out of context and requires the person viewing it to remember what needs changing once dismissed.

Once this message is dismissed, the person using the app must remember what's required. With multiple issues, or as in the figure, it's not specific about which entry is incorrect, this can be very confusing. A person may have to submit the form multiple times to be reminded of what needs correcting. Also, that many people ignore prompts makes this a poor choice for reporting actions they must take.

Put it into practice: optimize your forms

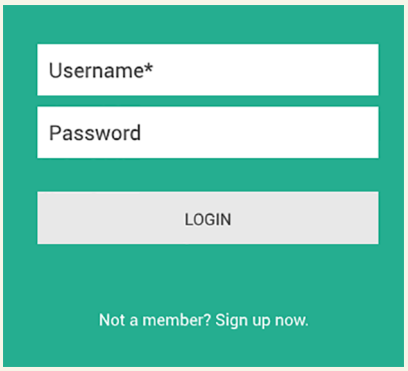
Do the following to ensure you make your forms as intuitive and easy to use as possible:

- Ensure that lists of options are ordered appropriately for the information you're asking for. For large lists, if a few options are selected significantly more than others, include a copy of those options at the top of the list.
- Allow a person to type to filter the list or jump to the options starting with the typed character(s).
- Position labels above the fields they relate to.
- Ensure that any hints about the required formatting of data are visible when the text box has the focus or is partially complete.
- Set all keyboards appropriate to the data being asked for.
- Add masking to input where appropriate.
- Ensure that default values are used appropriately. Add any you can; adjust any you must.
- Add smart validation to email addresses. But beware of using simple regular expressions, due to the large number of edge cases and international/Unicode domains to consider.
- Add tracking to analyze what's selected from large lists to keep common summaries up to date.
- Analyze validation failures to see what issues are being encountered, and address them.
- Indicate errors inline and not in a summary at the top or the bottom of the form, where they may be scrolled out of view.

Capturing information from the person using the app is necessary to provide the value they're using the app for, but can be complicated and covers many varied scenarios. This chapter presented some fundamental issues to be aware of and common approaches available to make the process simple and intuitive for the people providing the data. While this chapter was about data that's provided directly by the person using the app, in the next chapter, I'll discuss data that comes from elsewhere.

Exercise: evaluate this login form

Test your understanding of the last section by stating whether the following statements about this partial screenshot are true:



1. The username may be an email address.
2. The username is mandatory.
3. The password is optional.
4. No special restrictions on the contents of the password apply.
5. It's clear what to do if someone has forgotten their username or password.

Summary

- When it comes to entering data into forms, less is better; none is best.
- Typing accurately is difficult. Avoid the need for it where possible.
- Adjust keyboards for the type of information you're asking for, and don't ask for things you don't need. Ask for things when you need them.
- Selecting is easier than typing.
- Passwords are harder to type accurately and impossible to verify when you can't see them.
- Getting information for the business and doing what's easiest for the person using the app aren't always the same thing. You must find a balance.
- Make required fields and error messages clear.
- Give feedback as quickly as possible. Do this without calling a remote server when you can.
- Don't rely on just typed text input. Consider the many alternatives available, some of which are simpler and easier to use than typing.

WEB/MOBILE

Usability Matters

Mobile-first UX for developers and other accidental designers

Matt Lacey

Just because a mobile app works doesn't mean real people are going to like it. Usability matters! Most mobile developers wind up being part-time designers, and mastering a few core principles of mobile UI can make the difference between app and crap.

Usability Matters is a guide for developers wrestling with the subtle art of mobile design. With each expertly presented example, app developer and designer Matt Lacey provides easy-to-implement techniques that instantly boost your design IQ. Skipping highbrow design theory, he addresses topics like gracefully handling network dropouts and creating intuitive data inputs. Read this book and your apps will look better, your users will be happier, and you might even get some high-fives at the next design review.

What's Inside

- Understanding your users
- Optimizing input and output
- Creating fast, responsive experiences
- Coping with poor network conditions
- Managing power and resources

This book is for mobile developers working on native or web-based apps.

Matt Lacey has been creating mobile apps since 2001. He's an independent mobile development consultant, a community leader, and a Microsoft MVP.

"At last! A systematic treatment of the many different aspects involved in designing an app that people can use intuitively."
—Alan Lenton
www.IBGames.com

"An engaging and easy read. It is instantly apparent this book distills years of experience into easy-to-digest chapters."
—Desmond Horsley
NSW Health Pathology

"Highly recommended to any developer trying to wrap their head around app usability."
—Clifford Kamppari-Miller
The Tech Service

"There were many gems in this book that brought on an 'Oh yeah, that makes sense!' kind of response."
—Alvin Raj, Oracle

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/books/usability-matters

ISBN-13: 978-1-61729-393-1
ISBN-10: 1-61729-393-8
54499
9 781617 293931

MANNING US $ 44.99 | Can $ 59.99