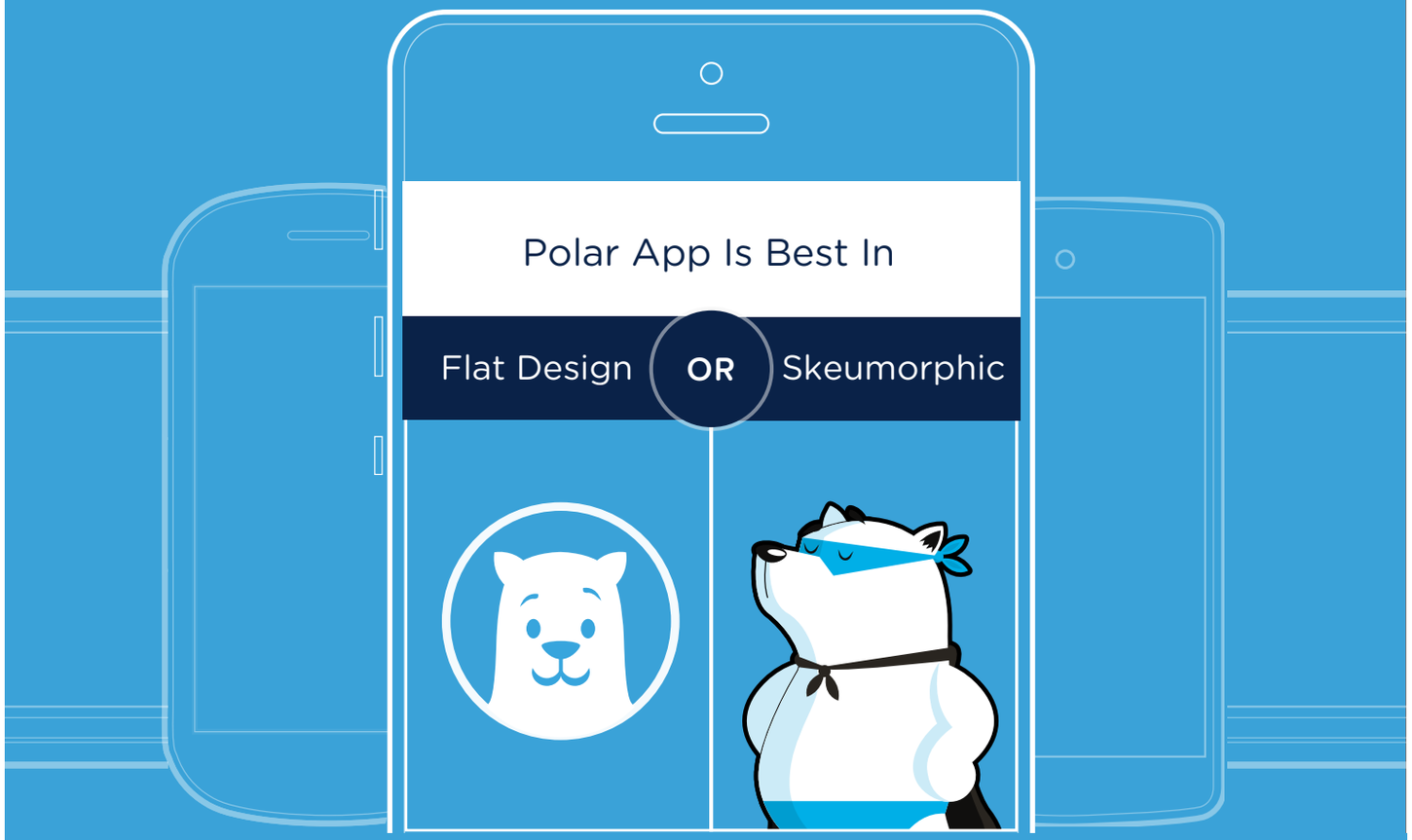
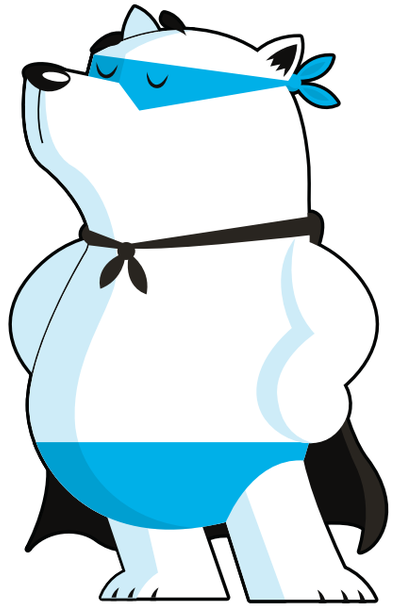
Mobile & Multi-Device Design

Lessons Learned Building Polar



Luke Wroblewski

Introduction



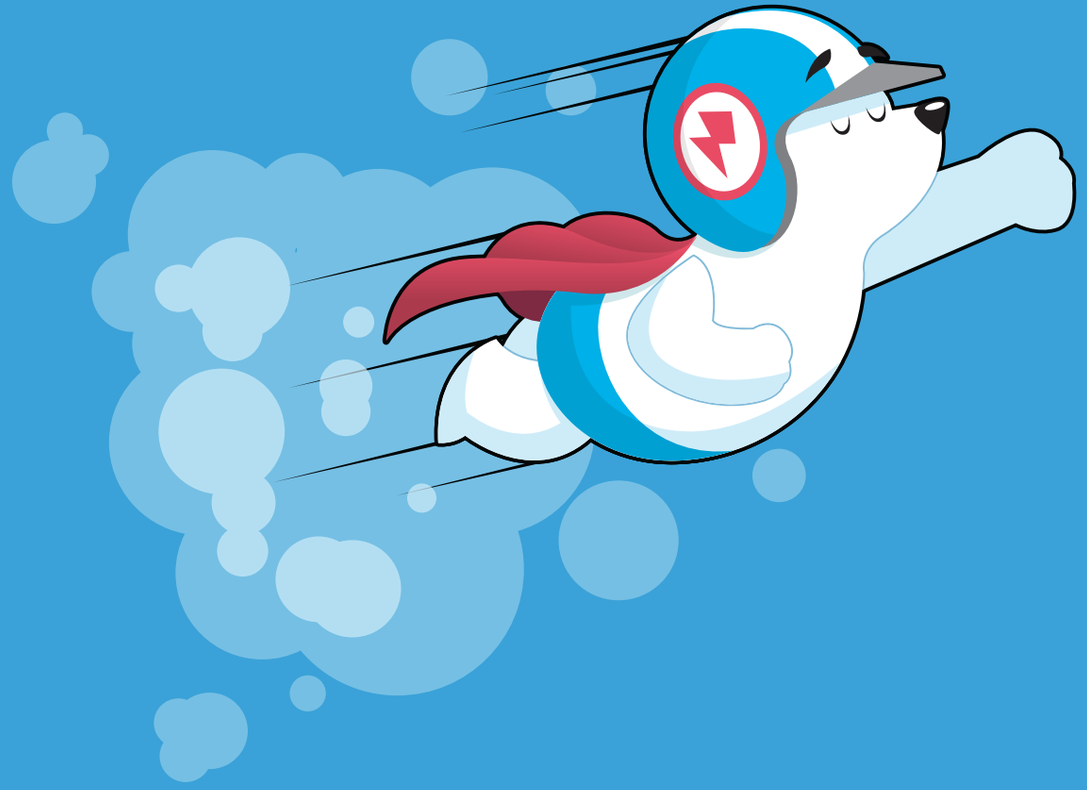
Creating products is a journey. And like any journey, it's filled with new experiences, missteps, and perhaps most importantly, opportunities to learn. My most recent product journey started nearly two years ago when we began working on Polar.

Polar started with the simple idea that everyone has an opinion worth hearing. But the tools that existed online to meet this need weren't up to the task: think Web forms, radio buttons, and worse. Ugh. We felt we could do much better by making opinions easy and fun for everyone.

Along the way we learned a lot and, as time permitted, shared some of our thinking, our failures, and our successes. This book compiles the articles I published over the past two years about Polar's mobile and multi-device design. It's our hope that some of our experiences help you with the product journey you're on.

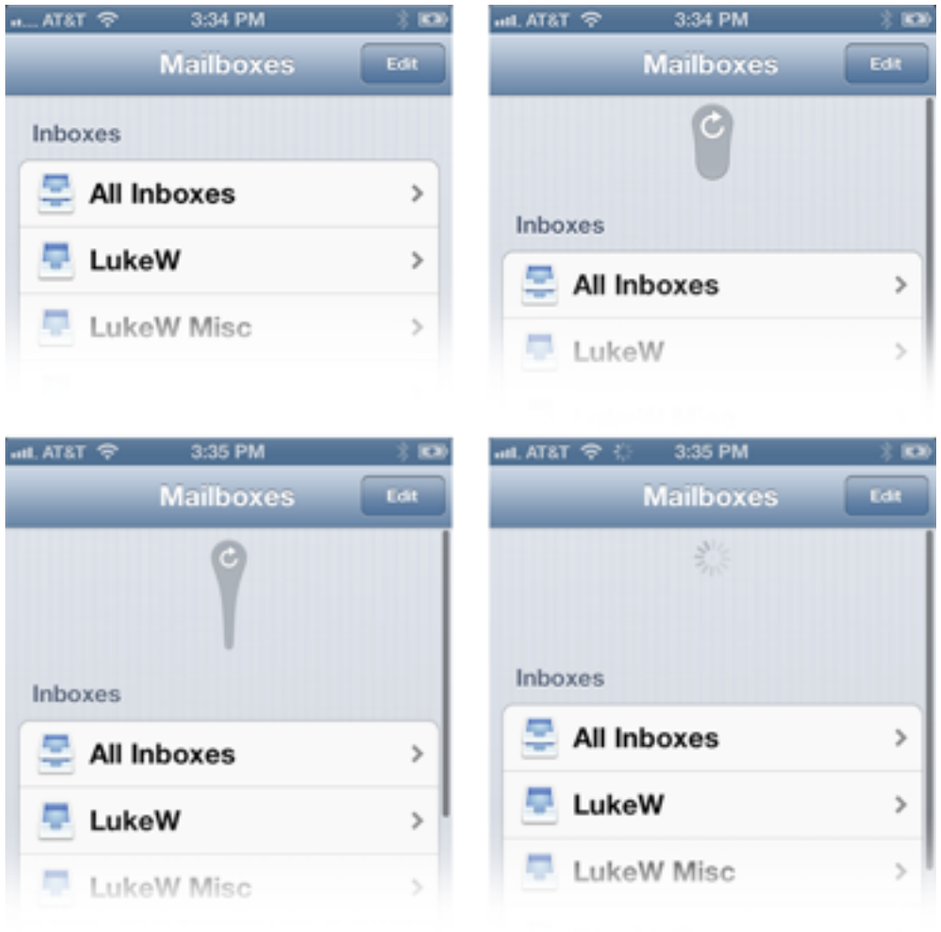
Safe Travels,
Luke Wroblewski and The Polar Team

Mobile Design Details



Mobile Design Details

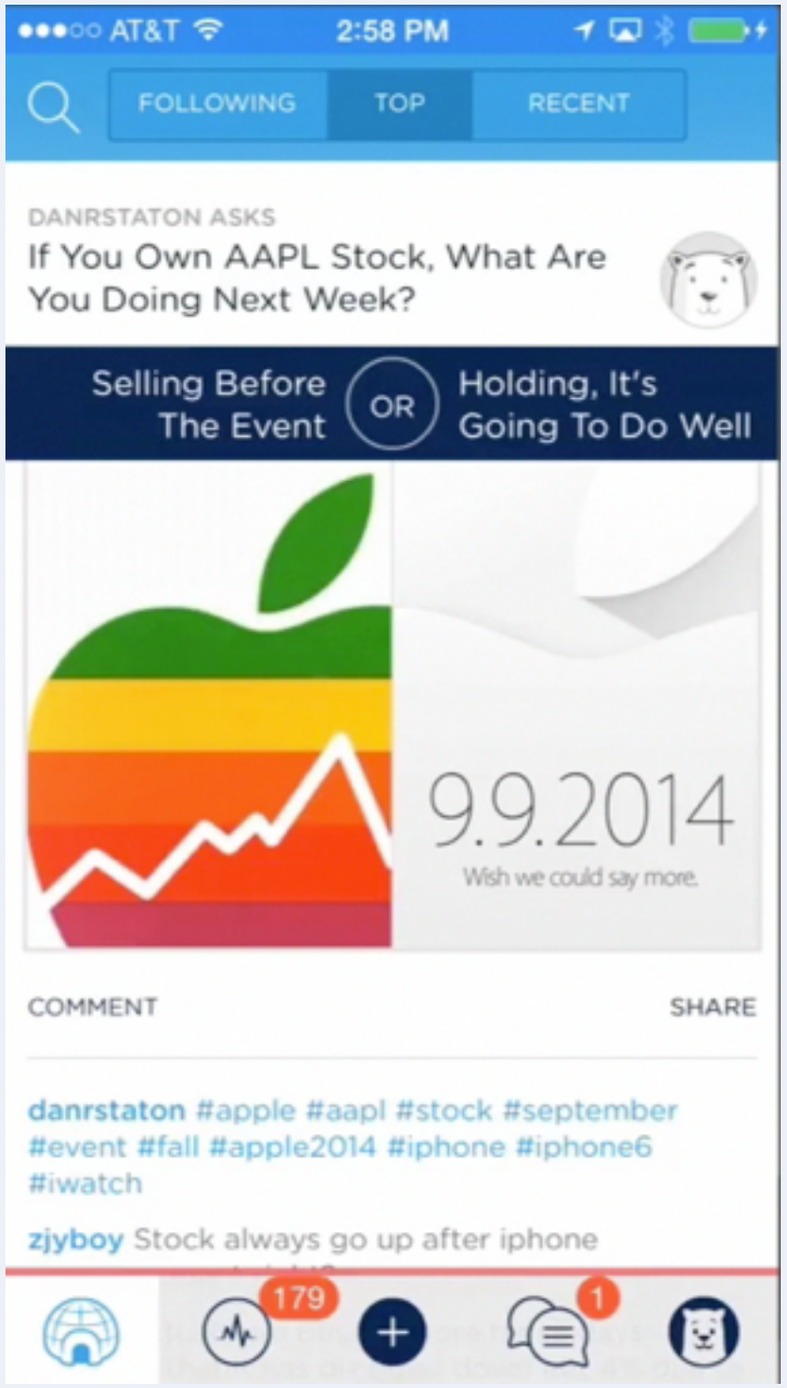
Pull To Refresh



The small screen real estate of mobile devices, often has designers and marketers pining for the branding opportunities that large screens enable. But there's actually lots of interesting ways to reinforce a brand on smaller screens as well. As this touch gesture example illustrates.

It's becoming quite common to find "**pull to refresh**" features on touch-based mobile devices. That is, to load new content, you simply need to drag a list down with your finger to trigger a refresh action.

As you do, not only does the content in the list update but a lot of new screen space gets revealed that was previously off screen. Screen space that often goes unused. But it could be used to reinforce a visual identity or brand, which is what we did with the pull to refresh feature on the Polar app.



As you pull down to refresh a list in the app, a bit of artwork is revealed. If you keep pulling, more and more of the art is made visible until you see the complete picture.

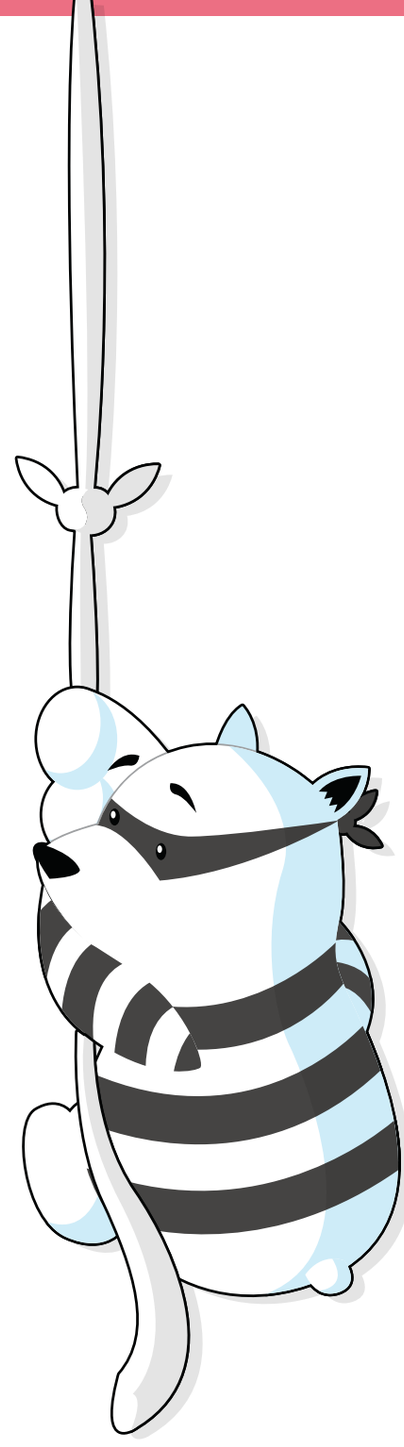
This little feature does a few interesting things for us:

1. It's another touchpoint for our visual identity, which is chock full of bears. Polar. Polls. Polar Bears. Polar Opposites. Get it?

2. It provides a reason for people to discuss what Jared Spool calls "socially transmitted functionality". That is features other people tell you about. There is no visual affordance for the pull to refresh action in the app. You either try it and see it works or someone tells you about. The little visual treat gets more people talking.

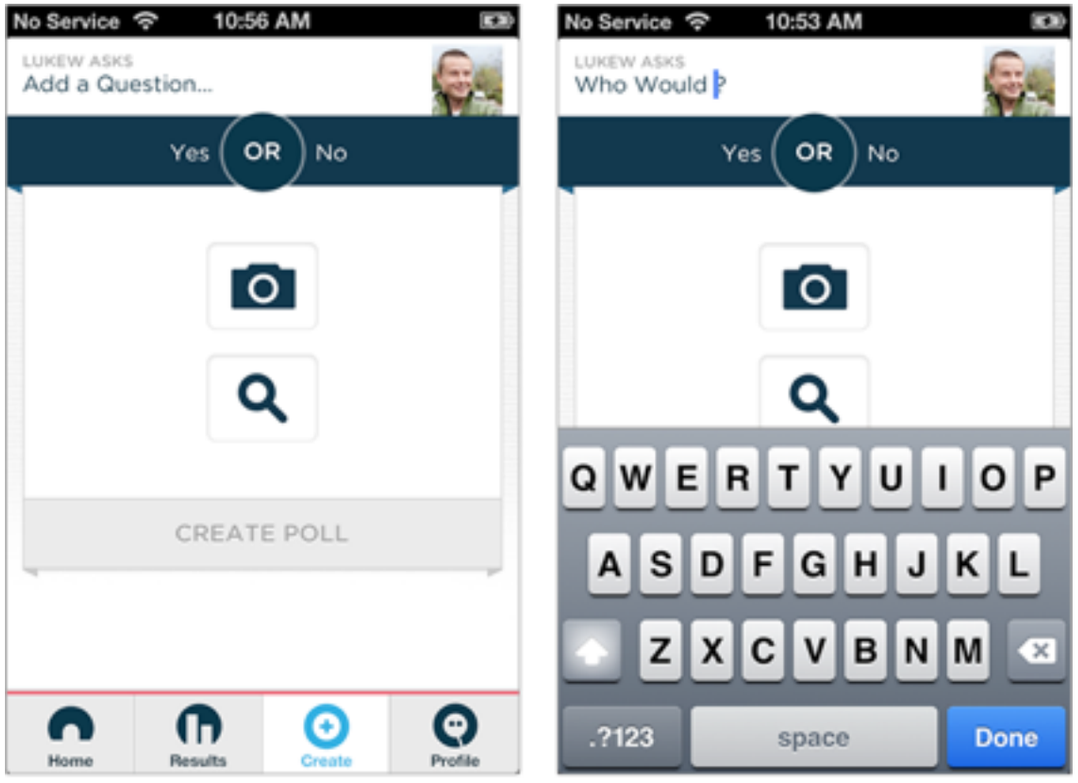
3. To keep the focus on lists of content, we keep our filtering actions & tabs (in this case the popular and toggle) off screen. Creating some fun in the pull to refresh action gets people using it more and, as a consequence, discovering these off screen features.

So far this little detail has been helping us realize the benefits listed. But there's more we could do like adding animation or switching between multiple images (or hiding bears in the landscape version of this book... hint, hint.)



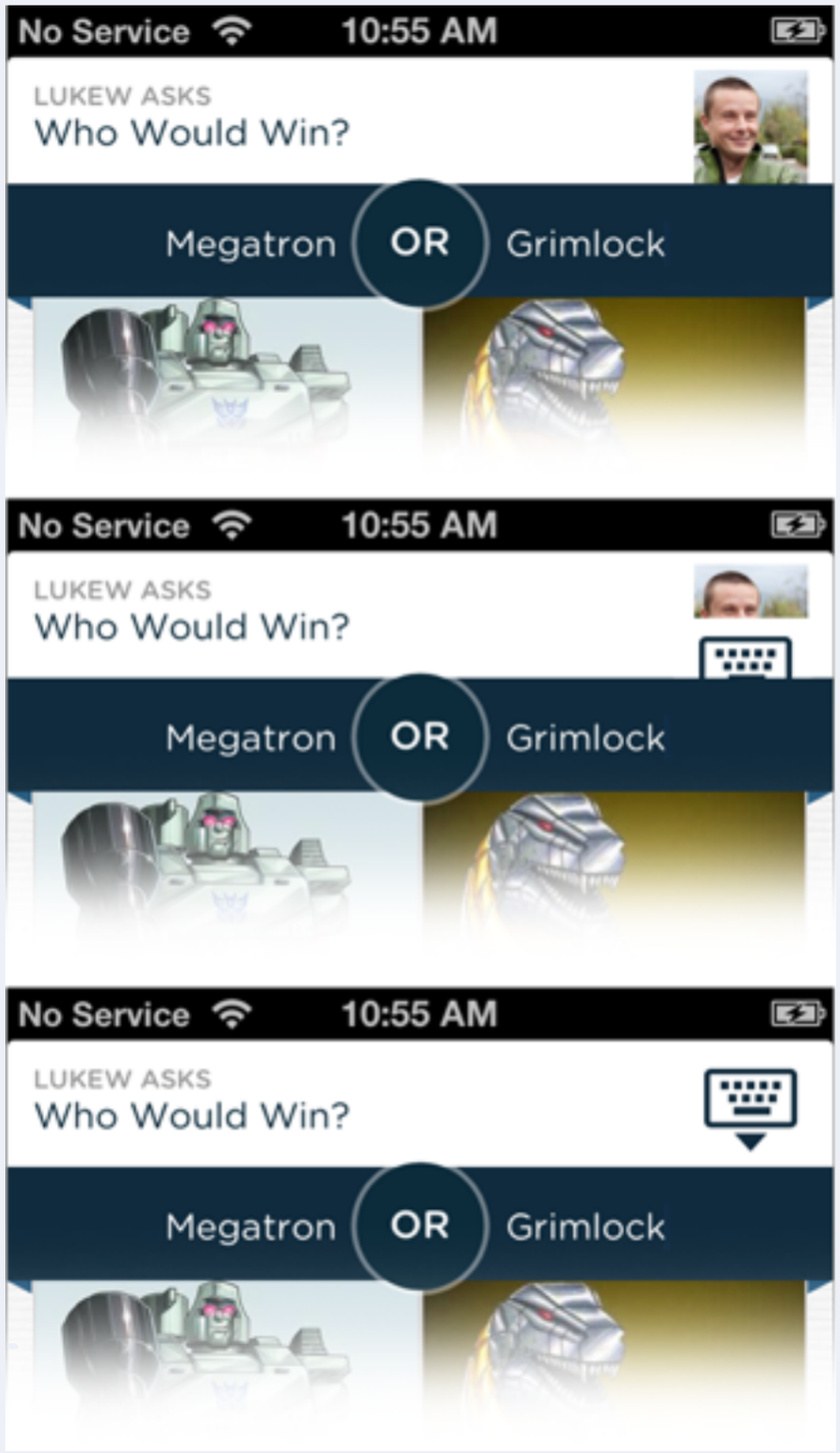
Mobile Design Details

Just In Time Actions



On mobile devices, there simply isn't room for a lot of user interface elements – even when they're useful. But while there may not be enough space to include everything up front, we can reveal relevant features only when they are needed. In other words, we can surface them just in time.

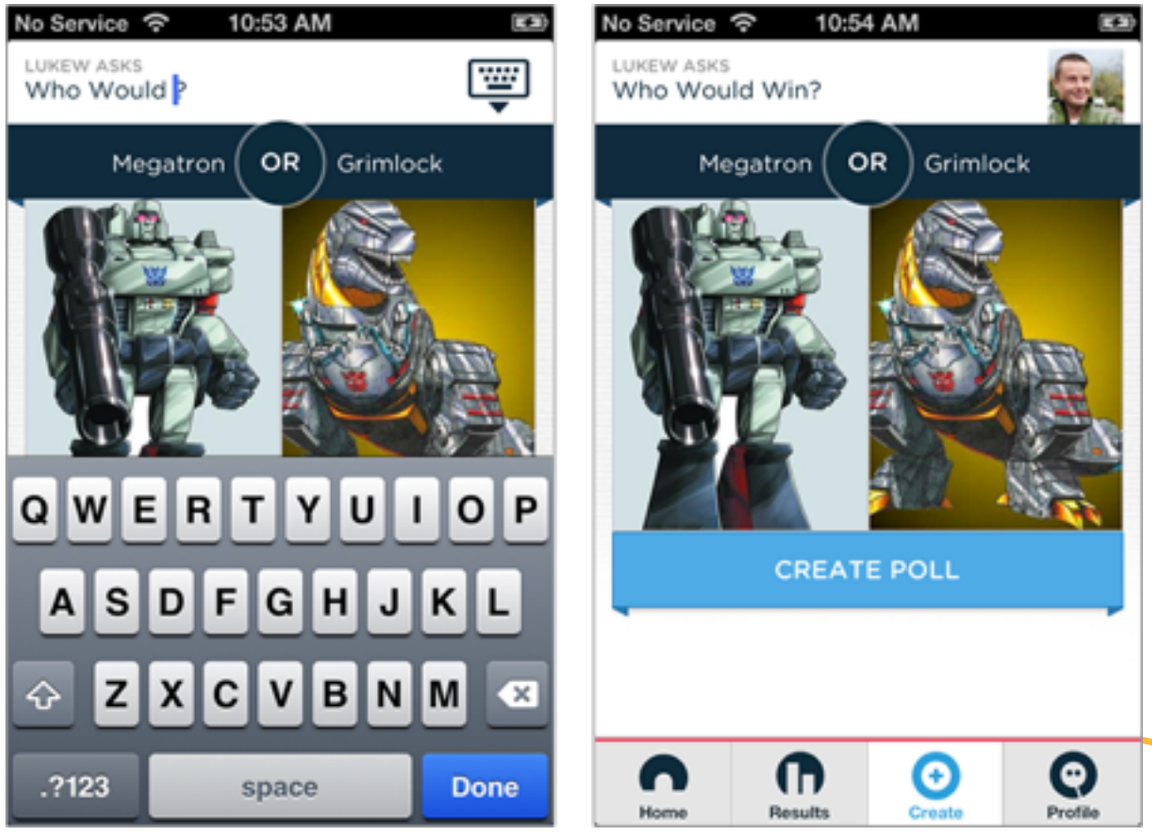
You can find an example of a just in time feature in Polar, our latest iOS app. Polar features a create screen that allows people to edit a number of text areas. When they do the virtual keyboard comes up and, unfortunately, partially covers people's work.



We heard frequently from people that they wanted to hide the keyboard in order to fully see the results of their work. This, of course, could be accomplished by tapping "Done" on the keyboard but that wasn't what many people did. Our first response was to educate people on the "swipe down" gesture that closed the keyboard by pushing it down. Our second was to make a number of the elements visible on screen (but not editable) dismiss the keyboard when tapped.

The problem with both of these solutions was the same: they were effectively invisible. People didn't know they were possible until they actually tried them out. While this worked for some people, we still didn't have a solution for everyone until we made use of a just in time action.

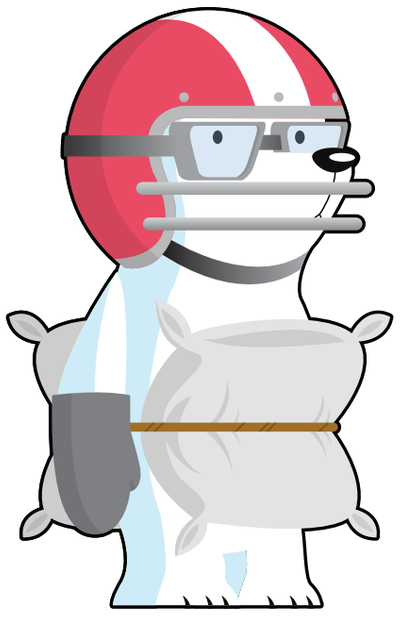
In the current app design, when someone decides to edit a text field and the keyboard comes up, we replace their profile photo with a "down keyboard" action that slides up with the keyboard (the subtle animation brings just a bit of attention to the feature). When the keyboard is dismissed by this action (or any other means), the "down keyboard" action slides down and the profile picture is revealed again.



This dismiss keyboard function is a just in time interaction. There when you need it, gone (and saving you screen space) when you don't. It also helps communicate otherwise invisible (often gesture-only) functionality.

There's a number of other places in the app where we've applied this technique to reveal "hidden" functions when they are needed. And since it's been quite effective, we're likely to find more...

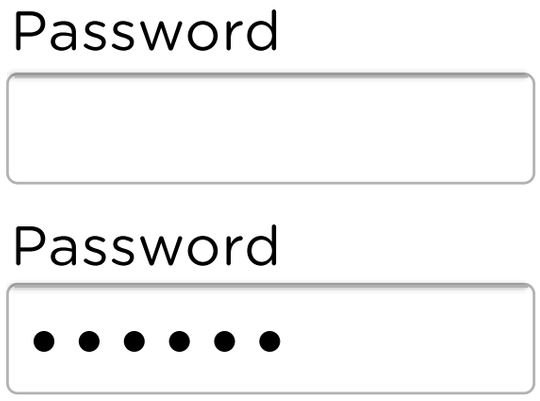
Hide / Show Passwords



Passwords on the Web have long been riddled with **usability issues.** From overly complex **security requirements** to difficult to use input fields, passwords frequently result in frustrated customers and lost business. The situation is even worse on mobile where small screens and imprecise fingers are the norm. But what can we do?

Once again, the **constraints and capabilities** of mobile devices challenge us to to rethink long-standing design standards. In this case, the password input field. You're all intimately familiar with this interaction because the average person logs at least **15 times** a day and **86% of companies** in the United States use password

authentication for their services. But just in case, here's how a typical password field works: you enter a character, it displays a "secure" response in the form of a •.



What's wrong with that, you may ask? Very simply put, there's no way for you to check your work by seeing what you entered. Which turns out to be very useful when you're **forced to use** a minimum amount of characters, some punctuation, and the birthdate of at least one French king for your password. So people often submit incorrect passwords and head into downward usability spirals.

Around **82%** of people have forgotten their passwords. Password recovery is the **number one** request to Intranet help desks and if people attempt to recover a password while checking out on a e-commerce site, **75% won't complete** their purchase. As usability advocate **Jakob Nielsen** puts it:

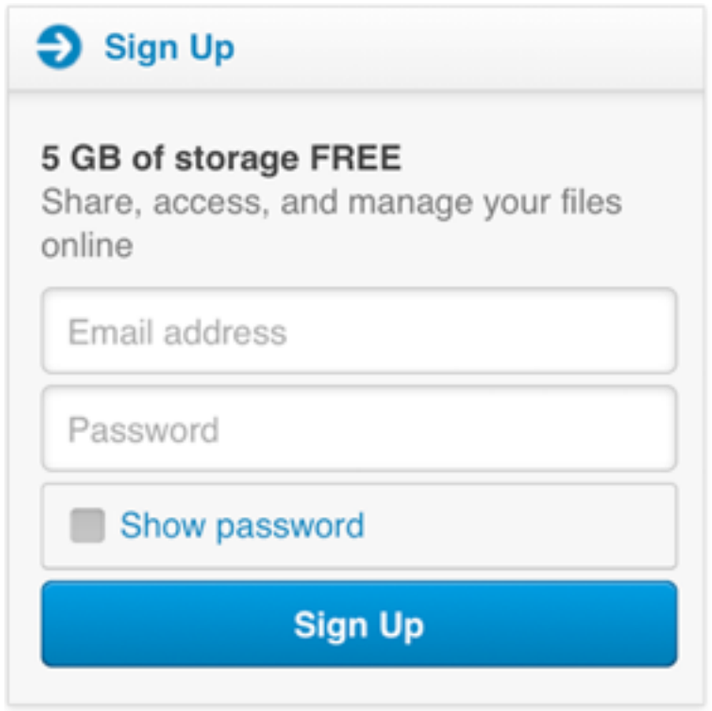
"Masking passwords doesn't even increase security, but it does cost you business due to login failures." ...and it's worse on mobile."

– Nielsen Norman Group

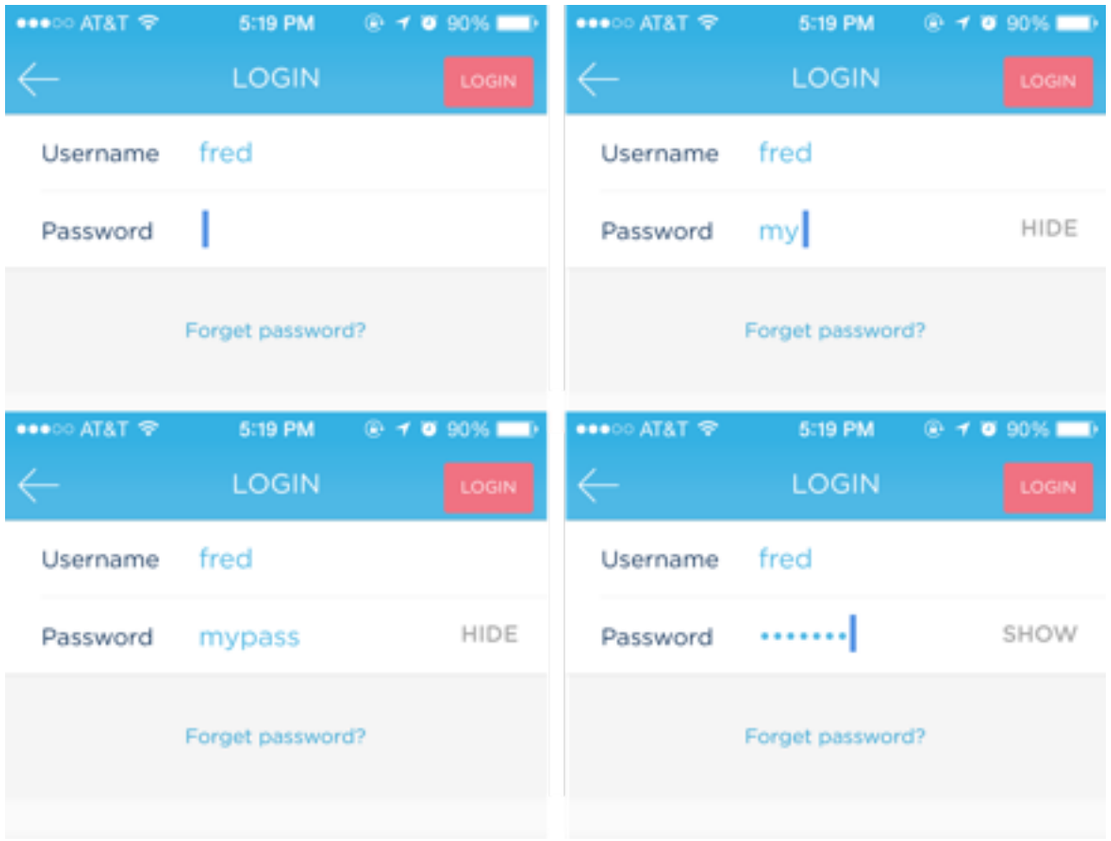
Some mobile operating systems have recognized that imprecise fingers and small screens make it even harder to accurately input complex characters into a password field. So they've made a small adjustment to the way password fields work: they show you the character you've entered for a brief moment before replacing it with the standard "secure" •.

But since many of us look at our touch keyboards while typing, when we do look up at the the character we entered, its no longer visible and inconveniently replaced by a •.

Recognizing this well-intentioned change isn't enough to fully counter password usability issues, some companies take the extra effort to provide a "Show Password" action. When activated, this action displays the contents of a password field in readable text but still sends it to a server securely.



This solution gives people the option of making their password visible and thereby readable but by default it appears as a string of ••••. Since most people **stick with default choices,** it may make sense to instead make the password visible by default but easily hidden if necessary. This is how we implemented things in our newest app, Polar.



By default Polar displays your password on our Log In screen as readable text. A simple, Hide action is present right next to the password field so in situations where you might need to, you can switch the password to a string of •••• instantly.

Wait... what? You're displaying people's passwords by default? Simply put, yes. We decided to optimize for usability and ease of log in over questionable security increases. On a touchscreen phone, its trivial to move the device out of sight of prying eyes. Or easier still to simply hit the Hide action to obscure a password.

But not that it matters, there's a visible touch keyboard directly below the input field that highlights each key as you press it. These bits of feedback show the characters in a password at a larger size than most input fields. So in reality, the •••• characters aren't really hiding a password from prying eyes anyway. As a result, we opted for usability improvements instead.

We also try to keep people logged in as much as possible by not displaying Sign Out actions front and center in the application. This makes entering passwords a rare occurrence and therefore less of an issue for the people using our app.

Testing One Thumb / Eyeball

People use their smartphones anywhere and everywhere they can, which often means distracted situations that require one-handed use and short bits of partial concentration. Effective mobile designs not only account for these one thumb/one eyeball experiences but aim to optimize for them as well.

It's no secret **smartphones get around:** at home, throughout the day, at work, while watching TV, during commutes, and beyond. **39%** of smartphone users even admit to being on their mobiles in the bathroom, which means the other 61% are liars. But whether we admit it or not, mobile experiences do happen everywhere. And that often includes distractions.

Whether you're on a crowded street corner or on the couch watching TV, chances are you're giving your phone just some of your attention. It's also quite likely that attention doesn't last long. The average person looks at their phone **150 times a day.** Most of these are brief interactions lasting a few minutes at best.

Designing for this reality of mobile use requires a laser-like focus on speed and simplicity. But how do you know if you're hitting the mark with a design? Timed, one-handed tests are one way to tell. When designing our new app Polar for Apple's iOS, we did just that.

The core tasks in Polar are voting on and creating photos polls. So these are the interactions we timed and tested with one-handed use. Our goal was to allow anyone to vote on 10 polls or to create a new poll in under sixty seconds using only one thumb. As you can see in the video above, we were able to do just that. In fact, we're often closer to the thirty second mark.

A first time user may take more time as there's usually some context they need to absorb but once someone goes through the flow once or twice, we found under sixty seconds to be easily achievable. Interesting side note: the use of voice input to create a poll wasn't much faster than one-thumb typing in our tests (also seen in the video above).

"What we need to do to design is to look at the extremes. The middle will take care of itself."

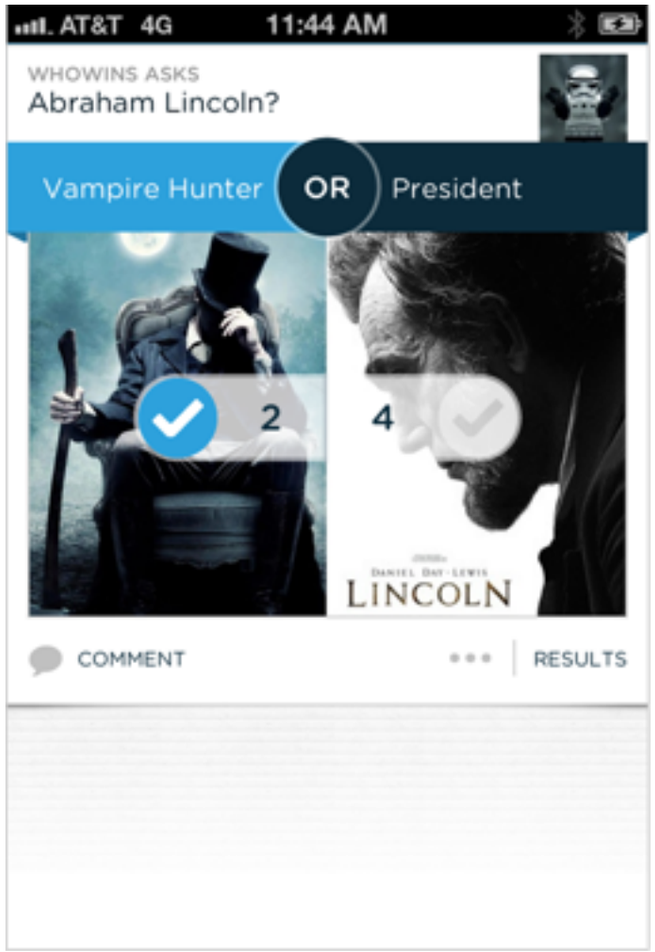
-Dan Formosa

Clearly there's more to testing and optimizing mobile interaction designs than timed, one-handed tests. But personally I consider this form of observation to be a great litmus test. If people can get things done in time sensitive, limited dexterity situations, they'll be even more efficient when we have their full attention and two-hands focused on our designs.

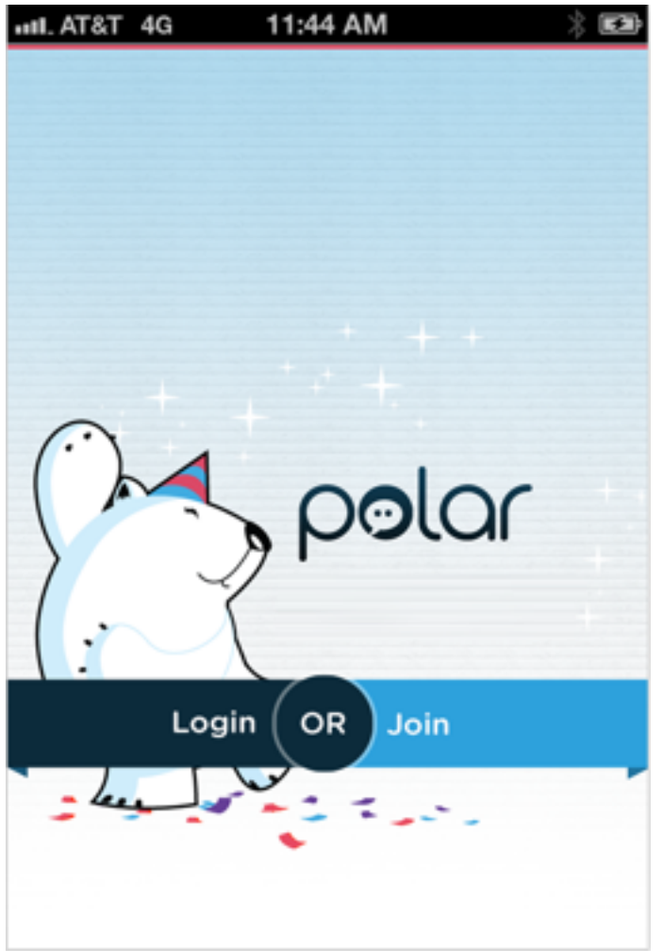
Echoing Core Interactions

There's no shortage of discussion about the value of consistency in interface design. But ensuring things are consistent isn't the goal. Presenting people with predictable and familiar interactions is. One way to get there is by echoing your primary interaction design throughout a product.

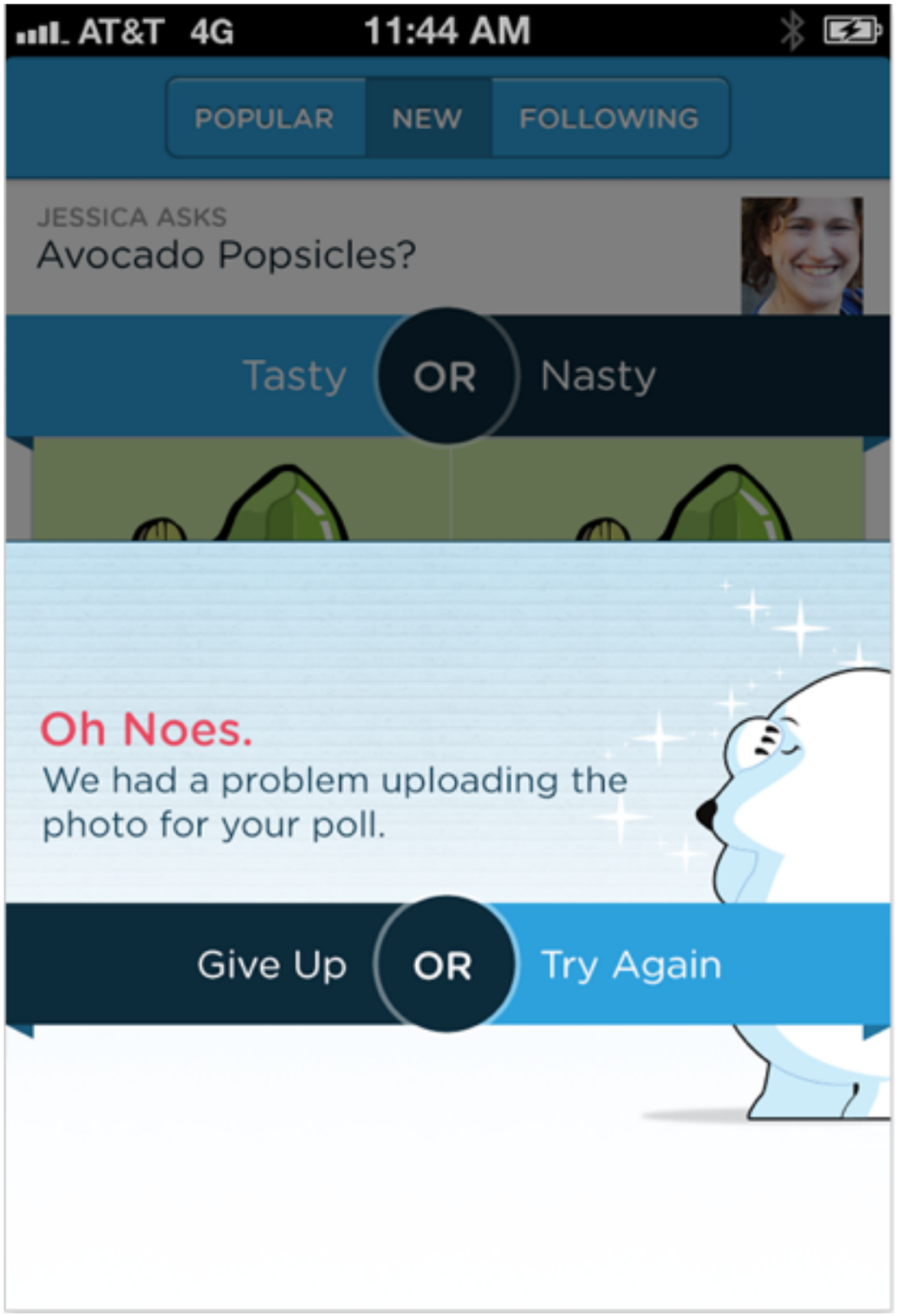
Echoing occurs when a specific interaction design is reused in various but appropriate contexts across a product experience. Since the same interaction model is reused, people can learn it in one place and apply their knowledge elsewhere. We used echoing a lot in our new mobile application, Polar. Let's see it in action.



The core interaction in Polar is voting on polls that feature two options presented in a blue banner separated by the word OR. When someone opens the app for the first time, they are presented with a list of these polls and can start voting right away by tapping on the left or right side of the banner or images below it. When they do, they've learned the core interaction we echo across the app.



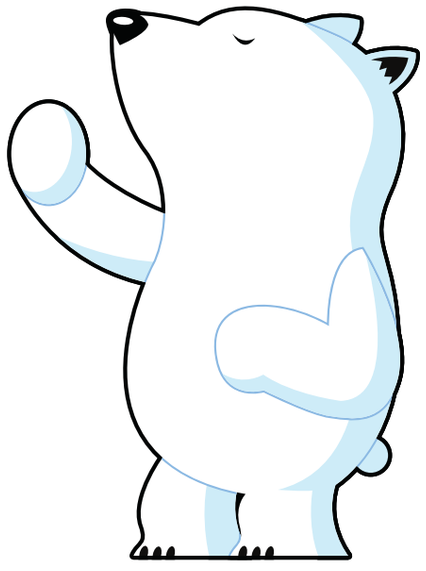
If someone wants to make a poll of their own, they're presented with our Login/Join screen. This design echoes the core interaction of voting by presenting the Login and Join choices as a set of two options in a blue banner separated by the word OR – just like voting on polls.



In the highly unfortunate situation where someone's poll fails to upload (we take lots of precautions to prevent this), their options to Give Up or Try Again will be presented in a blue banner separated by the word OR – just like voting on polls.

At this point you get the idea. Echoing can not only reinforce a core interaction across an experience, it can also unify the design of a product through familiar visual design elements and ideally make mundane interaction like Login, fun.

Mobile Sign Up Forms Must Die



It’s hard to get people to download your mobile application. But if they do, what greets them when they open it up for the first time? A step by step tour? A sign up form? Both might be missed opportunities to get people engaged and interested in your service. In fact, you might want to consider a **gradual engagement** approach instead. Here’s why...

Mobile App Realities

Before diving into gradual engagement, let’s look at **the realities** of mobile app usage. In 2011, the average smartphone user downloaded **about 2.5** new mobile apps a month. iOS users were on the high end with **about 5 apps** downloaded per month. Even if these numbers have gone up over the past year, that’s still not a lot of opportunity to get downloaded.

If you do get a download, chances are most people won't stick around. One study found that 26% of all apps downloaded were opened only once and then never used again. Only 26% were used 11 times or more. Of the remaining 48% of apps: 13% are opened only twice, 9% are opened only three times, and so on.

Of course, there are lots of reasons why people don't use apps on an ongoing basis but your sign-up process shouldn't be one of them. After all the hard work to get a download, losing new users at sign-up is a significant opportunity lost.

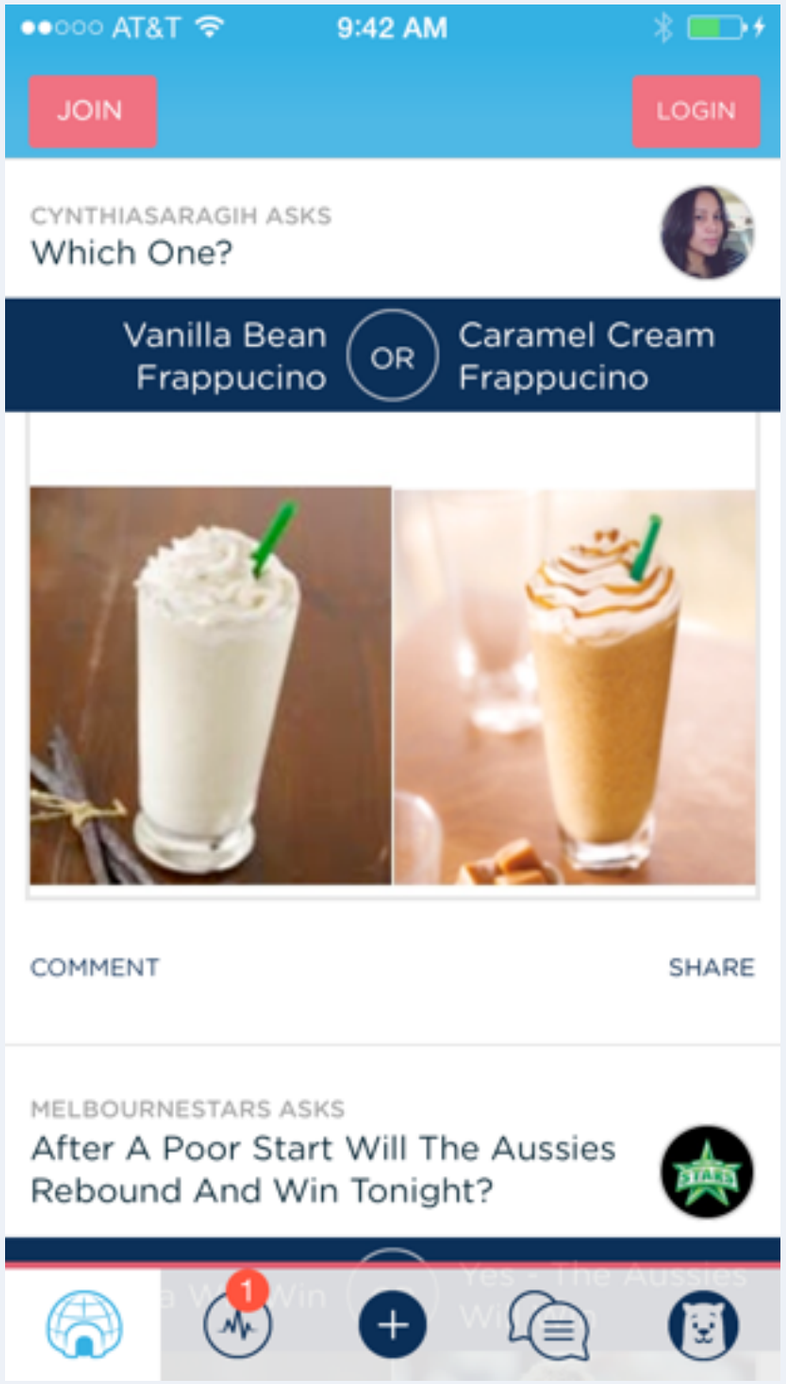
How significant? One very large scale app saw **only 25% completion** of their sign-up process. Vibhu Norby, the co-founder of Everyme wrote in detail how his app (at best) "retained 5% of users through their entire onboarding process". In other words, expect a lot of drop-off.

Gradual Engagement

Gradual engagement is the process of moving a user through an application or service – actually engaging with it, and seeing its benefits. With gradual engagement, new users are not just presented with a registration form and then dropped off a cliff. Instead, registration is either postponed, or handled behind the scenes and the first time experience is focused on giving people an understanding of how they can use a service and why they should care to.

This is the approach we took with our mobile app, Polar. Polar is all about sharing and collecting opinions. So out the gates, that's what we allow anyone opening the app for the first time to do: vote instantly on the polls they see. **87% of people who download the app do.**

Here's how it works. When a new or logged out user opens Polar, they are presented with a list of polls to vote on. We hold on to all their votes so if they ever create a username and profile page, all their votes will carry over.



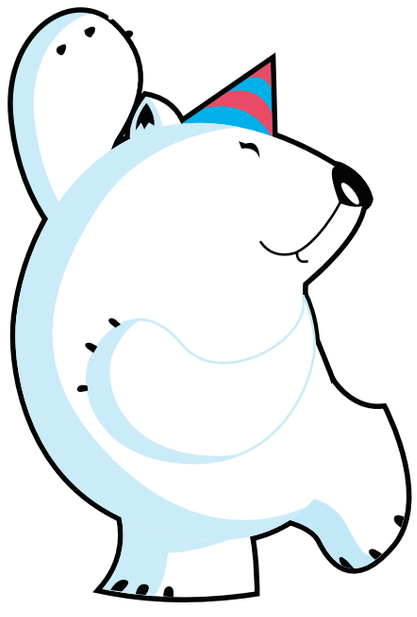
Commenting or creating a poll requires an account and the dreaded sign-up form. But we've made a lot of design decisions to make that process as fast and painless as possible. Massimo Pascotto recently wrote an article about the **design of our sign-up screen** if you're interested in more.

Results

With gradual engagement we can communicate what our mobile apps do and why people should care by actually allowing people to interact with them right away. We can capitalize on all the hard work it takes to get a download instead of turning 75% of our potential audience away with sign-up requirements.

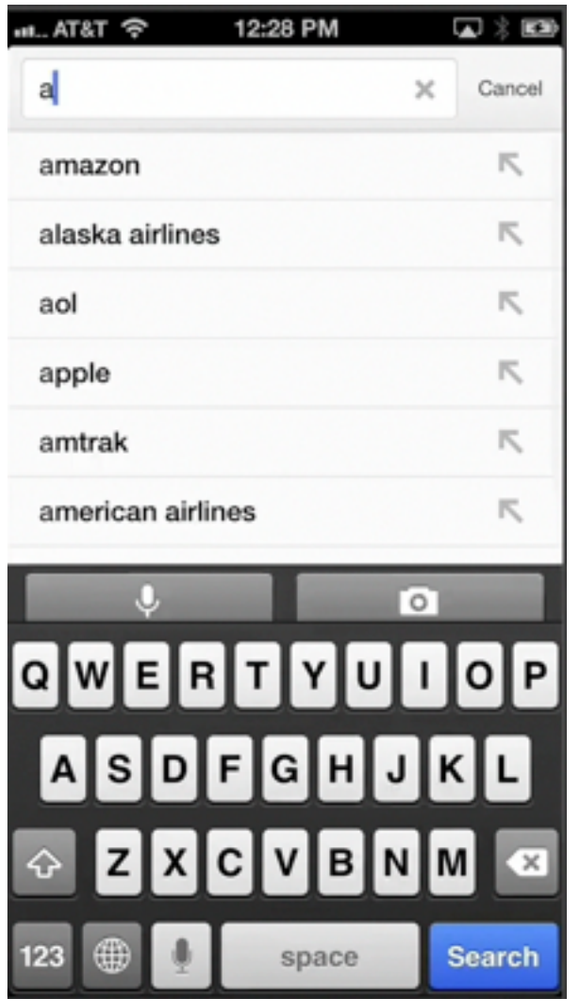
85% of the over 3.5 million votes on Polar have from signed-in users. But another 15% came from people voting without an account. And we're happy to give them a voice as well. No sign-up form required.

Performing Actions Optimistically



The situation on mobile is dire. People expect a faster experience on mobile then they get on the desktop but the networks connecting them to your service are naturally slower. As a result, your Web site or native app ends up fighting performance on both sides. In these situations it really pays to be an optimist.

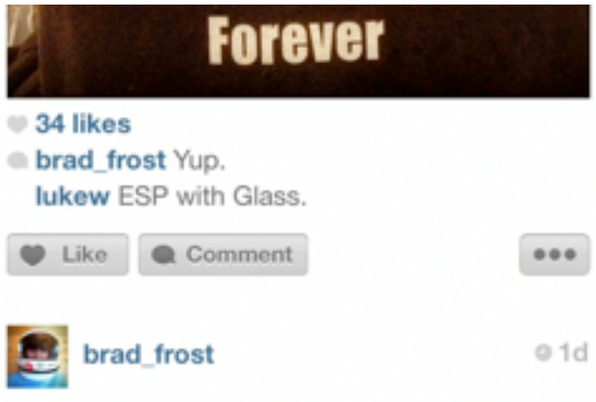
While there's **a lot** you can do to **actually speed things up** -like improving server response times and using inline CSS for above the fold content- you can only go so far. Eventually you'll bump into the **realities of mobile networks.** Luckily, though, perception is not reality and you may be surprised at how much can be done to make apps "feel" faster when they actually aren't.



For instance, Google's search app on iOS uses several animations to make the interface appear more responsive when results are being loaded. Due to the immediacy of these transitions, you get the sense that the app is reacting to your input despite the fact that nothing has actually loaded yet.

The same principle applies to acknowledging touch gestures with subtle UI changes. Right when you tap or swipe, the UI element you just touched changes appearance or moves to tell you something is happening. Techniques like this increase the perception of performance and – alongside actual performance improvements- they can go a while way toward creating fast mobile experiences.

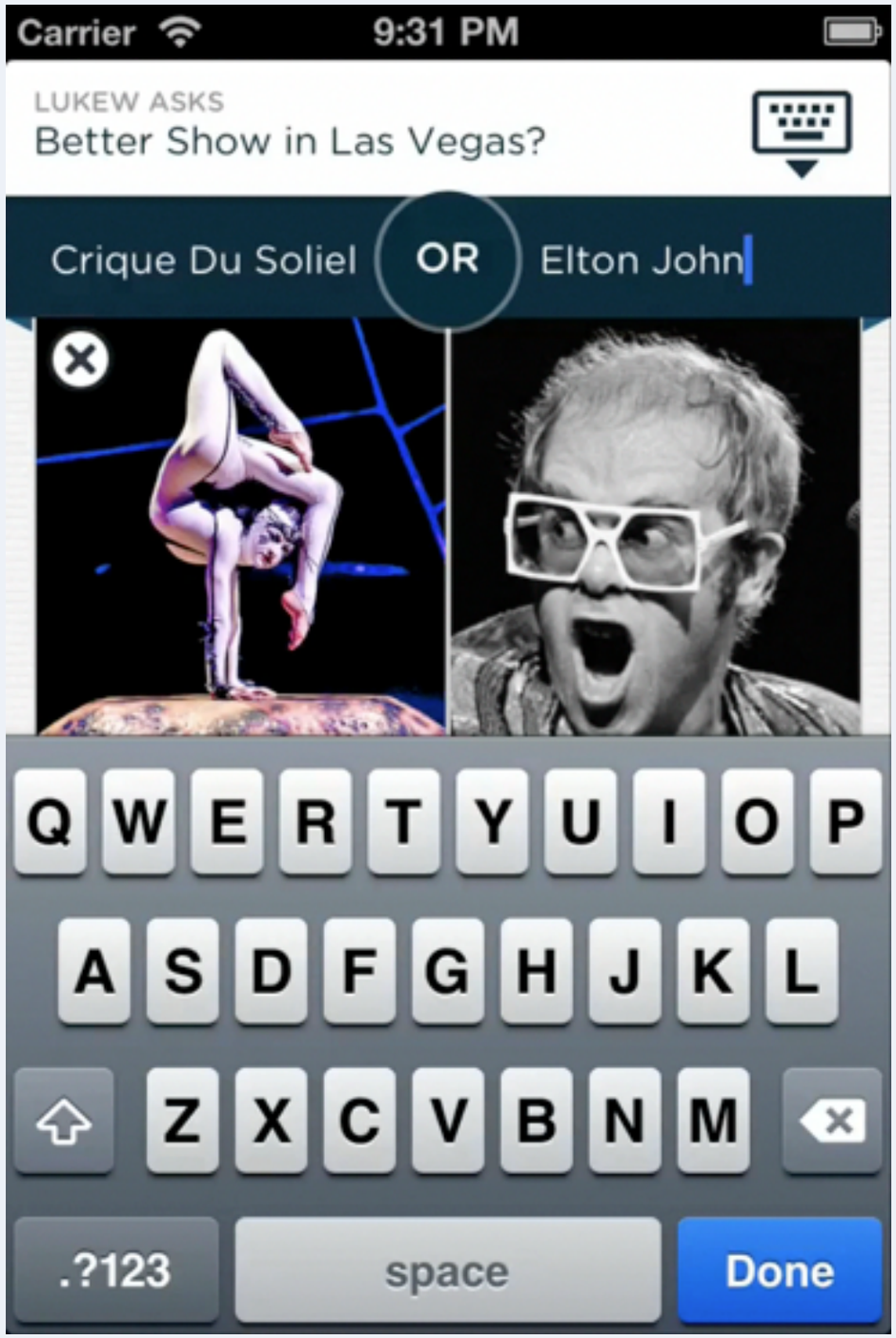
Taking this principle further, we can actually create the illusion that your action has taken effect when in reality it hasn't yet. Instagram co-founder Mike Krieger calls this technique "**performing actions optimistically**." As an example, when you like a photo on Instagram, the button changes instantly telling you that your action is complete.



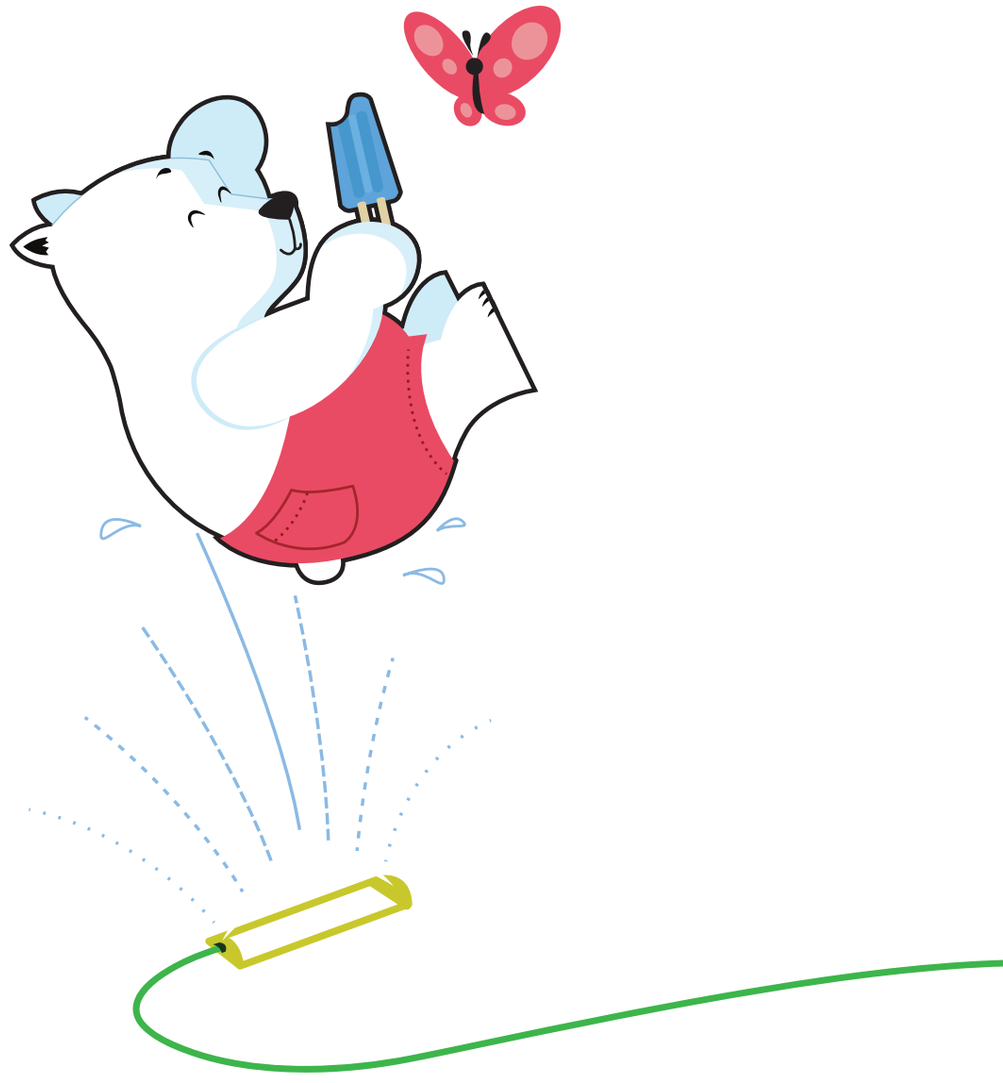
The reality is that a network connection is being made to tell a server what you did. But Instagram's user interface doesn't wait for the server to verify this actually happened. They optimistically assume it happened. If something goes wrong later, they deal with it then rather than incurring a delay up front. Commenting works the same way too.

We've employed similar techniques in our native mobile app, Polar but gone one step further and assumed any new polls you create will make it to our server. So right when you create a new poll on Polar it shows up in your feed. In truth, we've created a temporary local copy of the poll and added it to the front of the list.

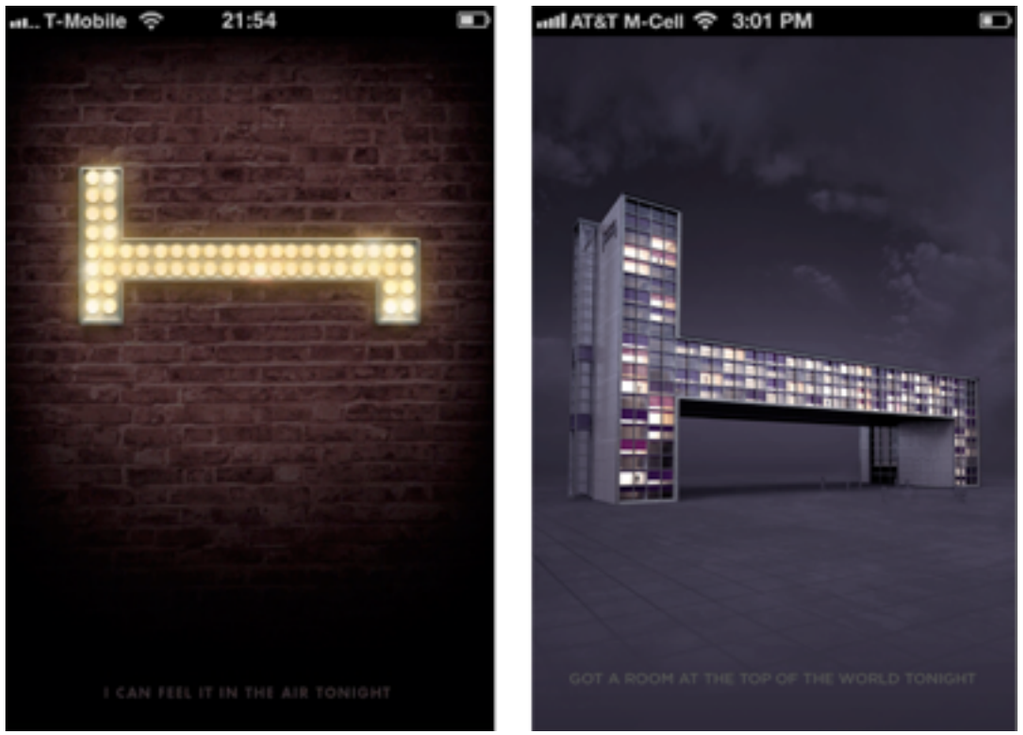
This temporary version of your poll is fully functional. That is you can vote and comment on it and we'll make sure your actions get applied to the actual poll once it is live. To ensure the poll does go live, we hold on to it locally and try to resend it several times before ultimately telling you something went wrong. Occasionally, an upload will make it to our servers but the mobile device won't tell us its done so we have an additional process that checks every few minutes to make sure no images get stuck.



If making temporary versions of polls fully functional and using multiple background processes to make sure uploads are successful sounds like a lot of extra effort to make things feel fast – it is. But the end result is worth it, when people create something on Polar it seems instantaneous. And in this case, perception beats reality.

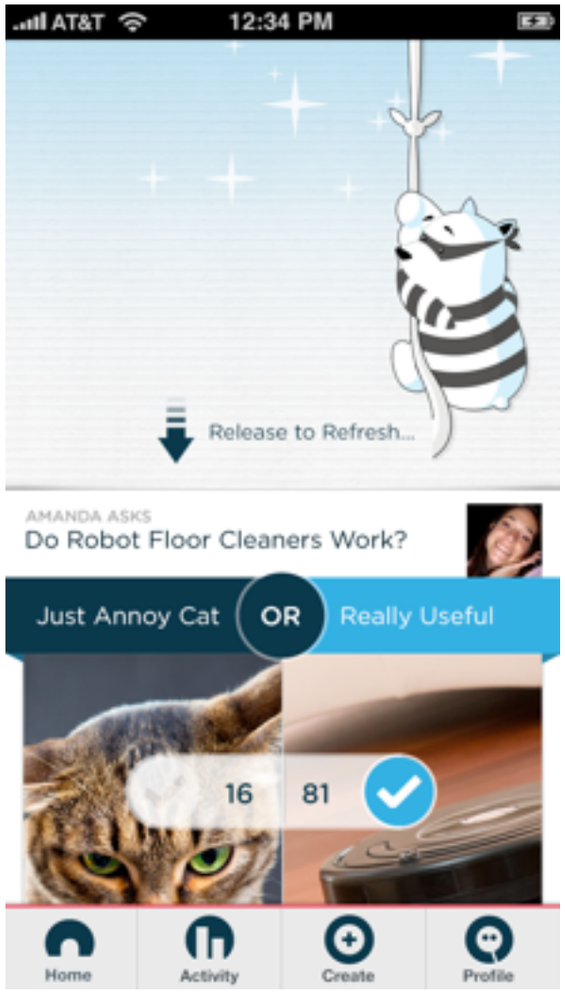


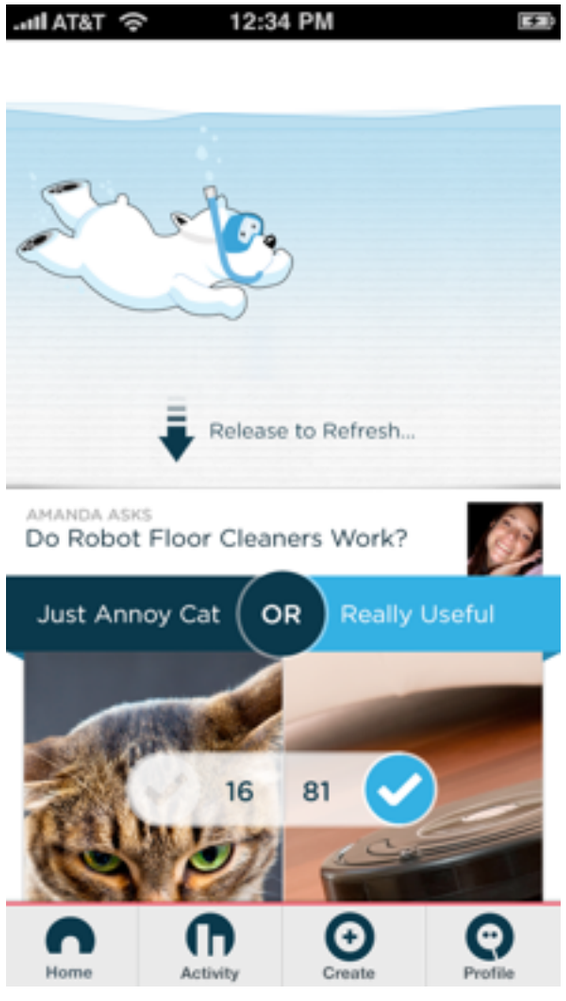
Make Each Release Unique



Designing for mobile makes a lot of people nostalgic for the big screens connected to laptop and desktop computers. These large canvases easily allowed an organization's brand to come to life with big visuals and rich media. In contrast, the slow networks and small screens on mobile devices seem to leave little room for effective brand treatments. In truth they do... but in different ways.

For example, Hotel Tonight includes a unique splash screen with each update of their mobile application. These variations make each release distinct and add some new life to the Hotel Tonight experience.





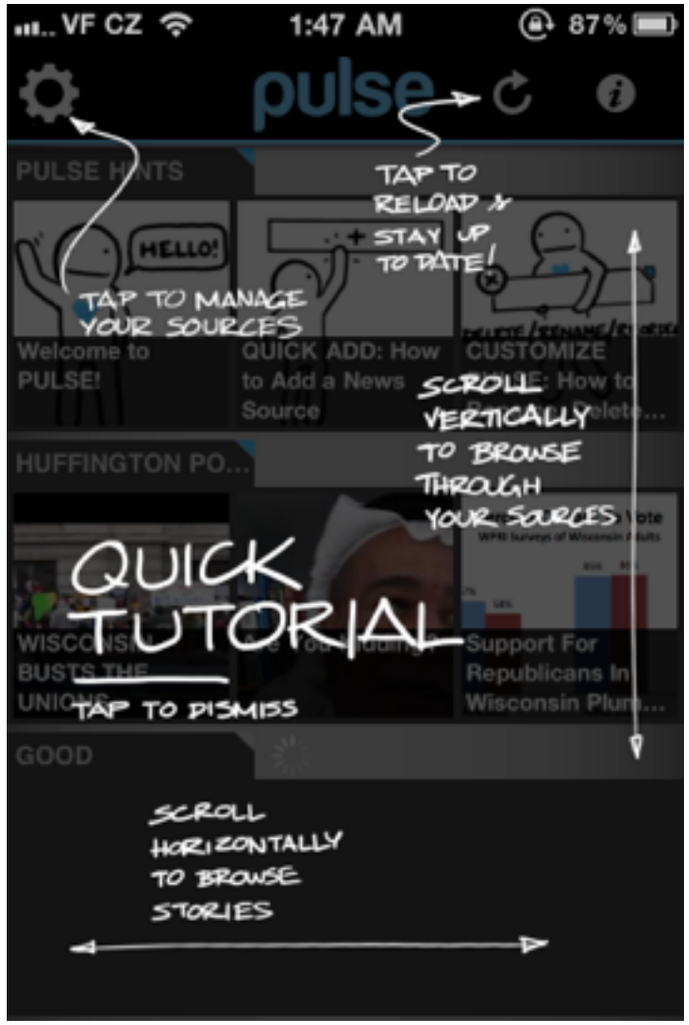
Similarly, we make each update of Polar unique with a continually changing interactive component. Each release of our mobile application features a new “hidden” character behind a pull to refresh gesture. When you pull down on a list of questions on Polar, you gradually reveal a polar bear caught in the act: snorkeling for fish, breaking out, or just hanging around.

Every Polar update has a new bear that comes along with it and our users love to track him down. In fact, they make polls comparing the bears and discuss which one is their absolute favorite. Our aim is to make collecting and sharing opinions fun and we like to reinforce that wherever possible – even in a simple pull to refresh interaction.

Just In Time Education

Without clear affordances, touch-based interactions are invisible to the human eye - we don't know what's possible. To address this, many mobile applications add introductory “tours” that walk you through how to do things using touch gestures. It’s a worthy goal but perhaps not the best way to reach it.

Most people (sometimes over 90%) skip over intro tours as quickly as possible and those that don’t rarely remember what they were supposed to learn. Both these issues stem from that fact that introductory tours show up before you ever get a chance to use an application. Most people are eager to jump right in and as a result, they skip reading the manual. The ones that do read haven’t seen the interface yet so they don’t have any sense of where and how the tips they’re learning will apply.

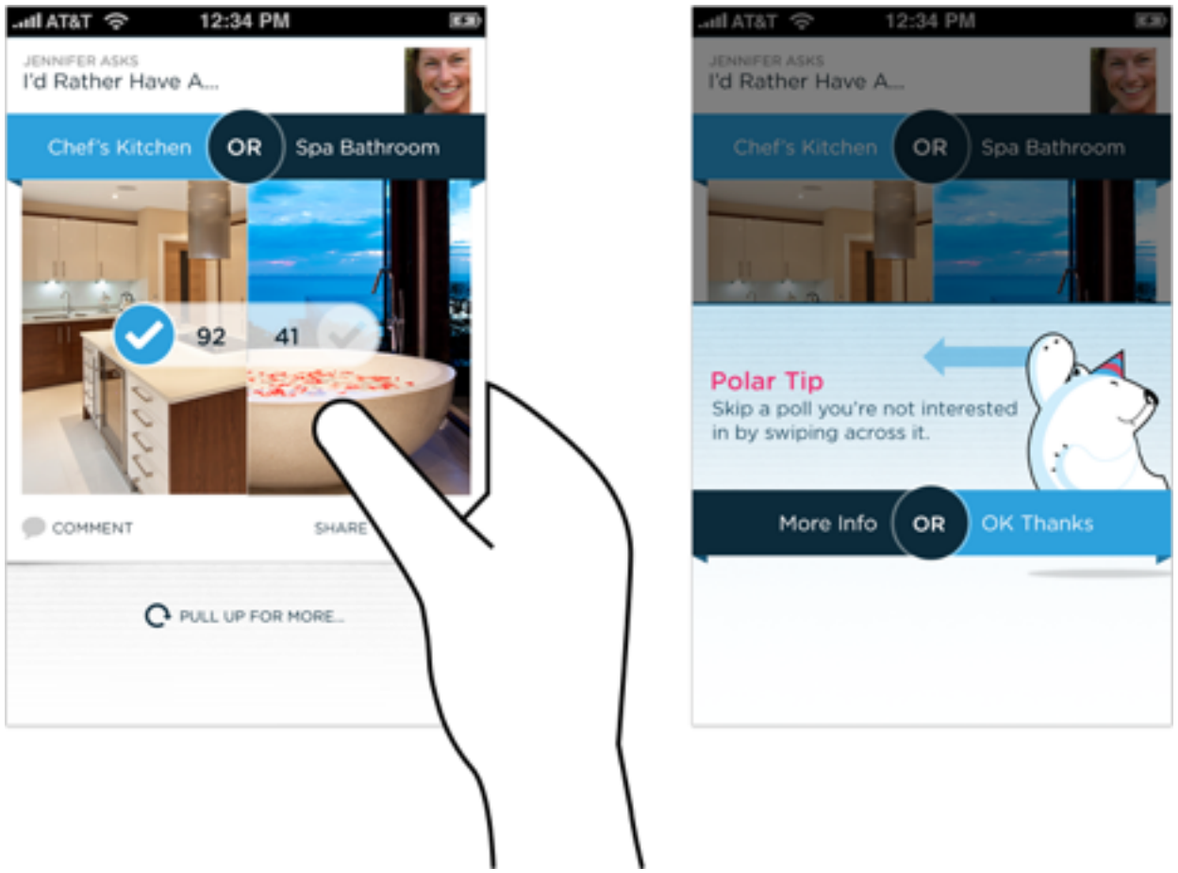


Some mobile apps aim to get around this by overlaying their tour on top of the actual interface design but even with a picture of the interface present, people lack the experience to know which actions will be useful to them and when. And over time they're likely to forget which interactions are possible in any given app as they switch between them regularly.

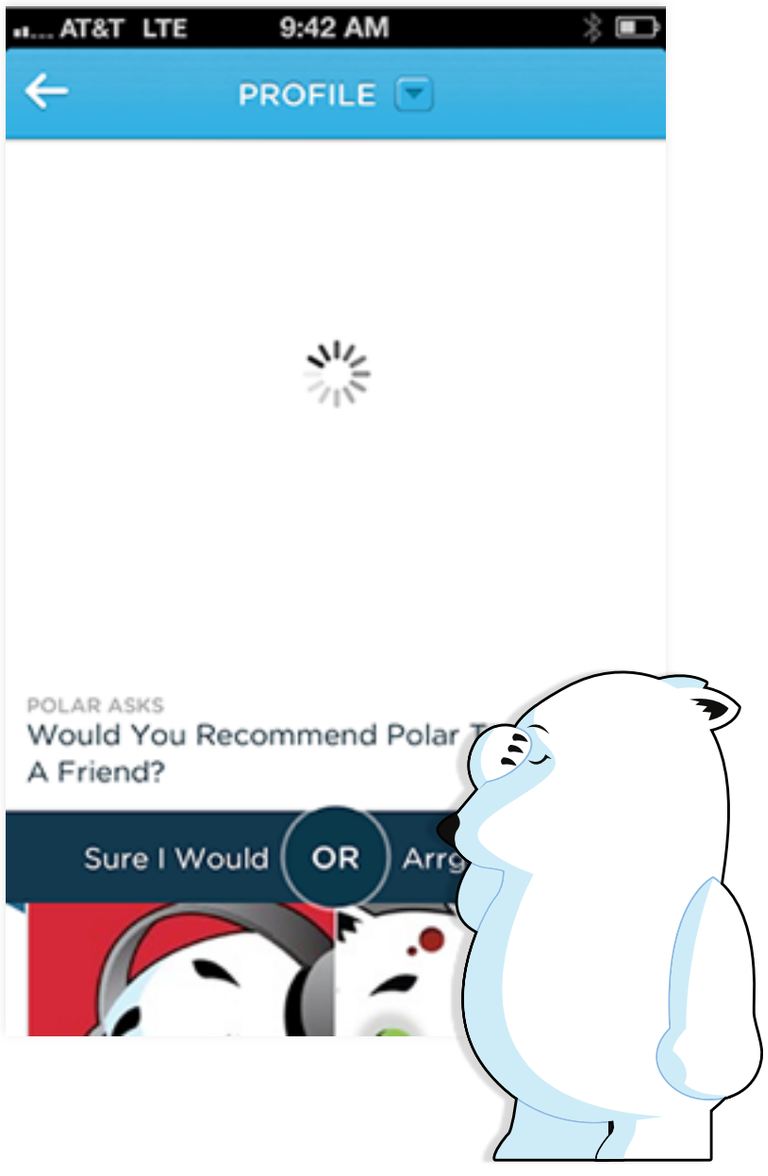
So instead of trying to teach everything up front and all at once, teach in the moment when specific information is actually useful. Author Josh Clark has called this approach "just in time education" and it's an effective alternative to intro tours. We've made use of it frequently in our mobile app, Polar.

After you've scrolled through and voted on a few polls on Polar, pulling **up to load more** at the bottom of the screen brings up a tip explaining how to skip polls you're not interested in **(video demo)**. This message only comes up after you've had a chance to use the application for a bit. At that point you're familiar with the basic interactions and likely ready to learn about a more "advanced" interaction.

Because being able to swipe across a poll is a hidden gesture, we use just in time education to reveal and explain it to you. While everyone sees this tip the first time they use the application, we can also surface it again if we notice people are not using the feature to help remind them of what's possible. The trick to getting just in time education right is revealing useful information **when people actually need it** not when they don't.



Avoid The Spinner



Conventional wisdom tells us that when things are going to take a while, we should let people know. In most mobile applications that translates to adding progress bars or spinners when something is happening or loading. While the intentions behind these progress indicators are good, the end result can actually turn out to be bad.

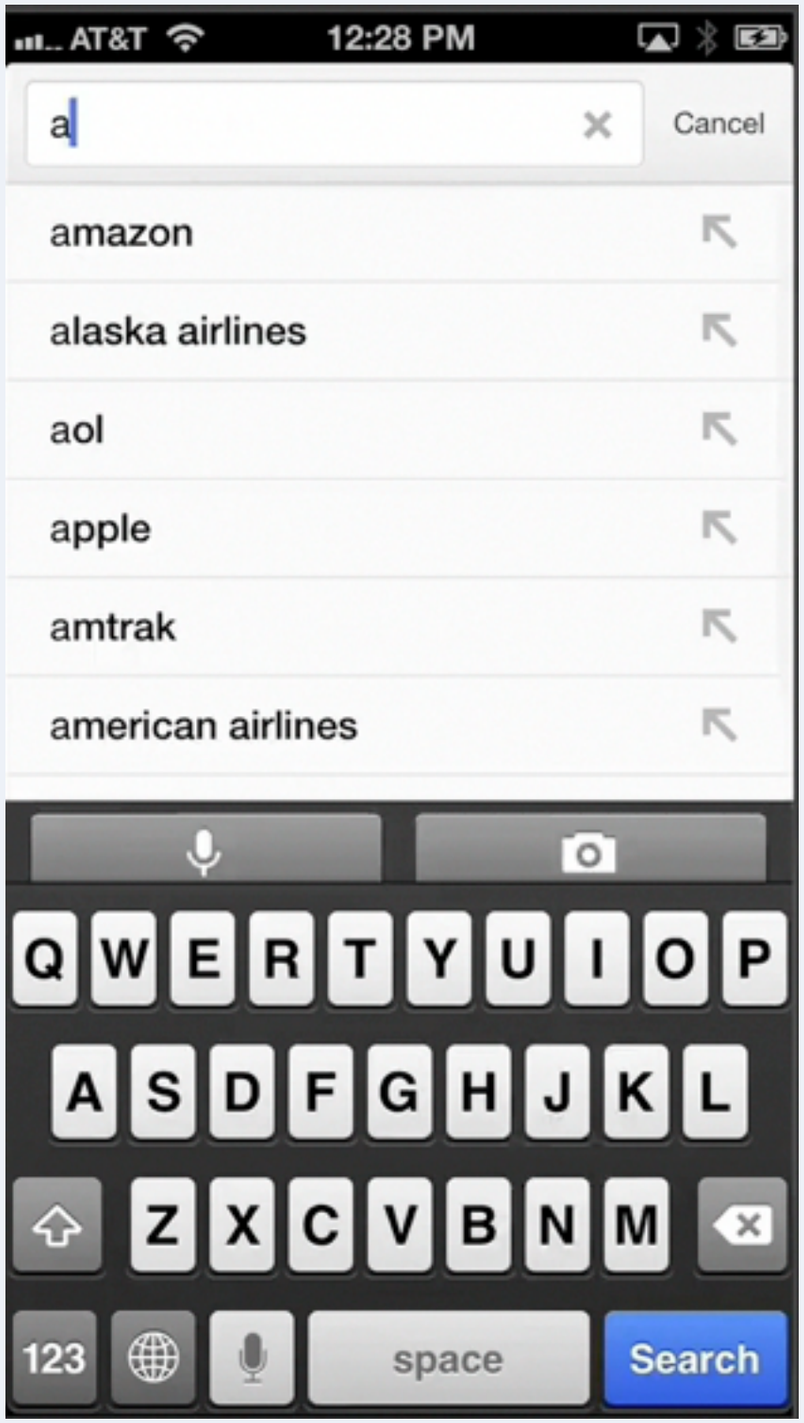
It can be bad because progress indicators by definition call attention to the fact that someone needs to wait. It's like watching the clock tick down – when you do, time seems to go slower. We learned this lesson the hard way on Polar when we experimented with using Web Views to load parts of our native application's interface. Web Views are like little embedded Web browsers that can fetch pages from a server and present them within an app but only after they are loaded.

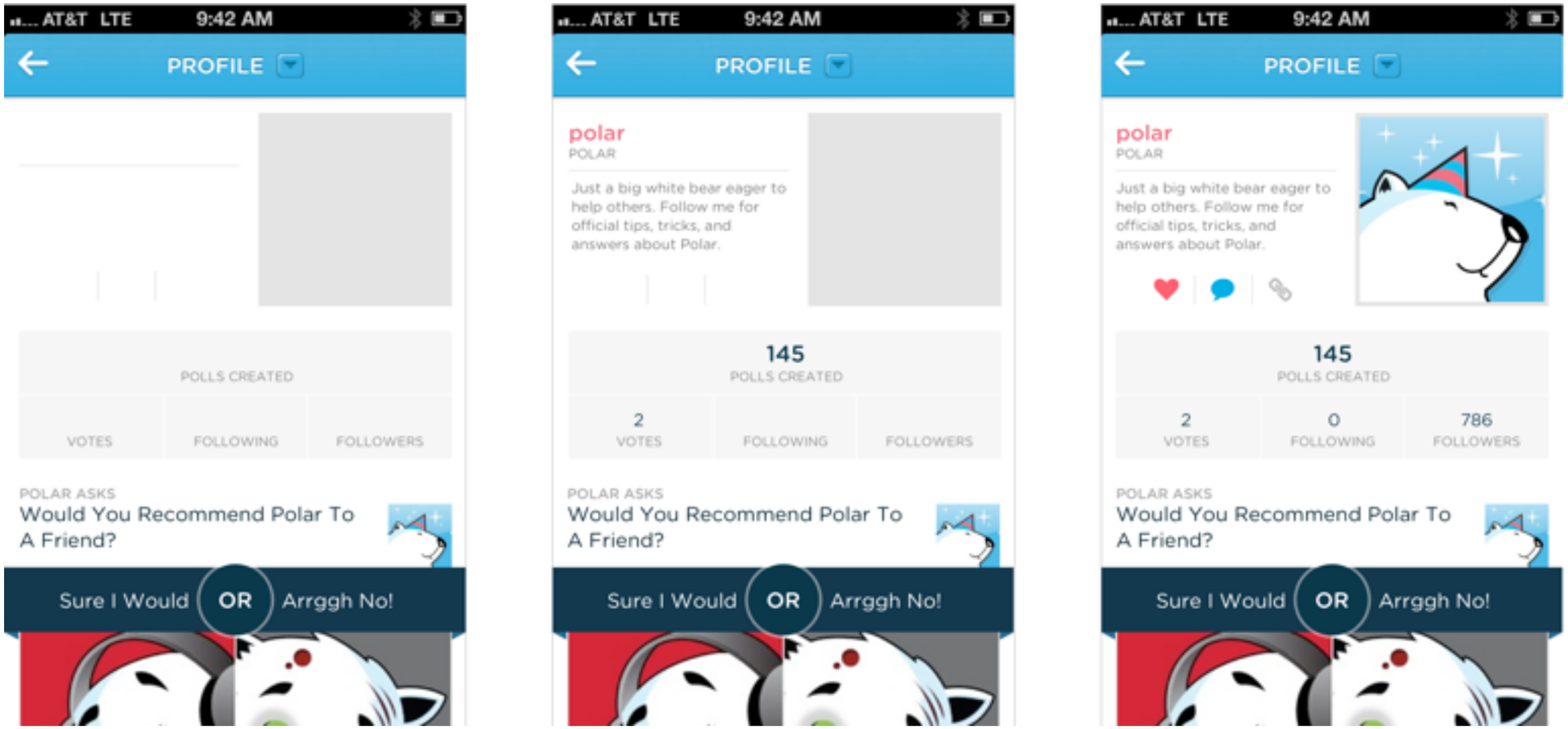
To let people know these elements were downloading, we added a spinner that showed up as each Web View was retrieved from our server. Since we used several Web Views, people could encounter these spinners in a few parts of the app and when they did we started to get feedback like this:

“There seems to be an excessive amount of waiting around for pages to refresh and load – it doesn't seem as quick as the previous version.”

With the introduction of these progress indicators, we had made people watch the clock. As a result, time went slower and so did our app. We focused on the indicator and not the progress, that is making it clear you are advancing toward your goal not just waiting around.

For example, in Google’s Search application the Web page you are loading slides in from the side, making it feel like content is loading immediately even when it isn’t. Google puts the focus on progress by making the loading indicator part of the transition that brings up the page you requested.





Skeleton screens are another way to focus on progress instead of wait times. We used this technique in several places on Polar to effectively eliminate our spinners. A skeleton screen is essentially a blank version of a page into which information is gradually loaded. This creates the sense that things are happening immediately as information is incrementally displayed on the screen.

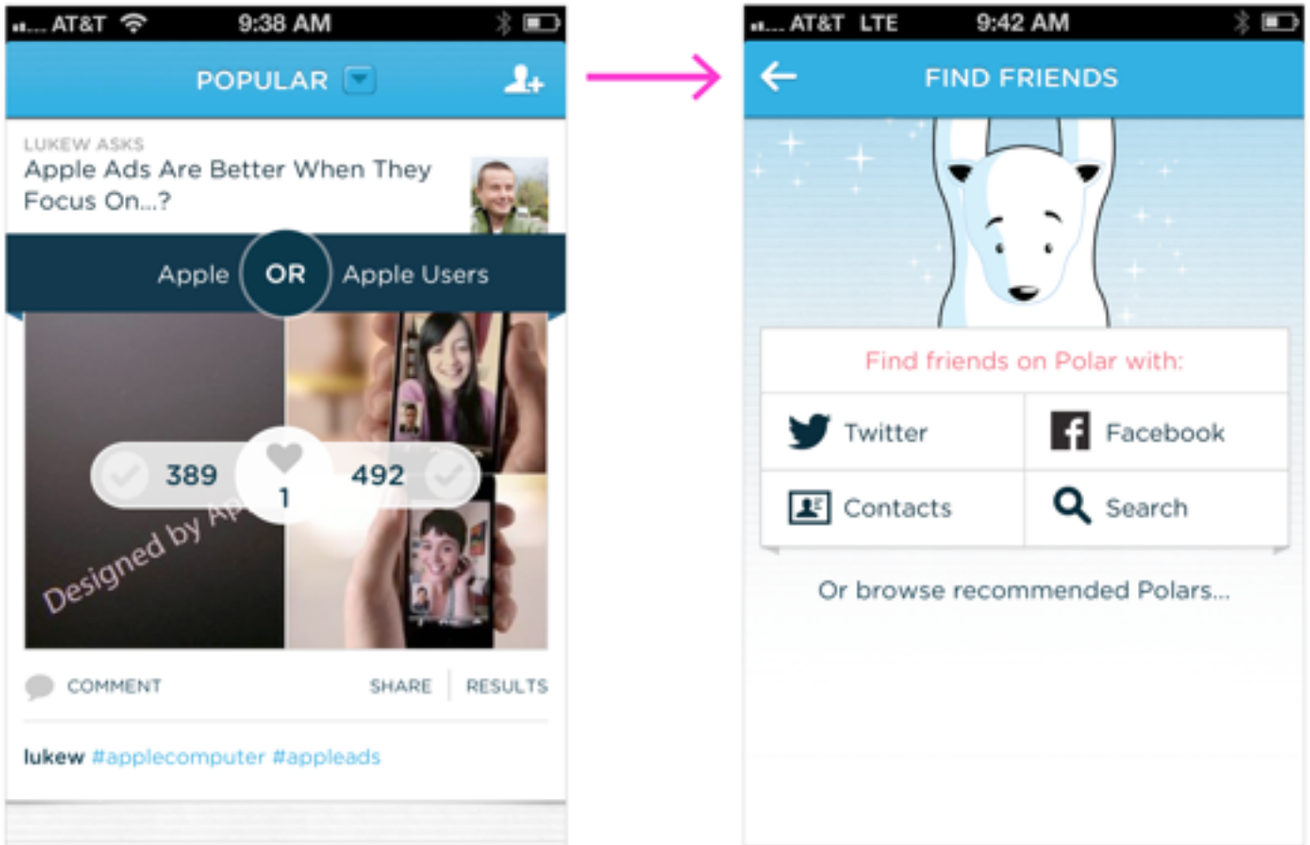
With a skeleton screen, the focus is on content being loaded not the fact that its loading and that's real progress.

Don’t Divert The Train

Outside of Hollywood blockbusters, getting in the way of a speeding freight train usually doesn’t end well. It takes a lot of effort to shift the course of something with that much momentum. So instead of trying, just hop on board.

Believe it or not, this little lesson applies to mobile app design as well. Rather than forcing people to divert their attention from their primary task, come to where they are.

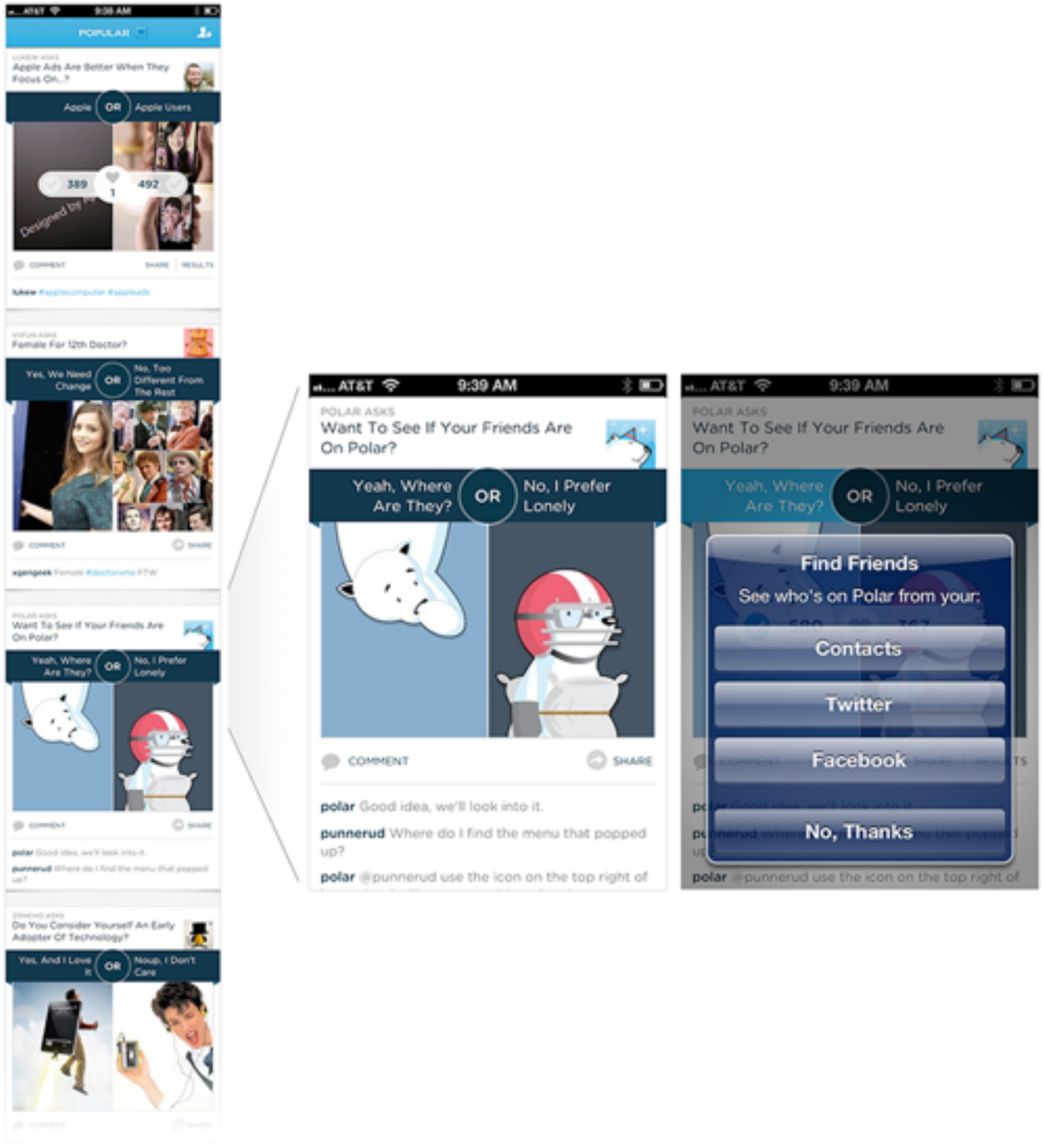
Let me illustrate with an example from our app, Polar. Polar is a fun way to collect and share opinions by making and voting on lots of photo polls. This is our freight train. We get over 40 votes per user on any given day. It’s where people spend the most time in the app and get immersed in the Polar experience.



We knew this experience could be even better if the list contained polls from people you know. So we added a prominent action in the header that allowed you to find your friends on Polar when you tapped it.

But very few people did. As it turned out, we were trying to divert the train by requiring people to go to a different part of the application to do things like find and invite friends.

So we decided to use the forward momentum of our “train” instead of fighting it. Now when someone is voting, voting, voting... the 20th poll we show them asks “Would you like to find your friends on Polar?” If they say yes, we connect them to their Address Book, Facebook, and so on.



When we made this change, use of the Find Friends feature shot up dramatically. Since then, we've redesigned a number of other features this way including setting preferences, requests to rate the app, and more. Treating these actions as part of the main activity of our app, in our case voting on polls, instead of as separate interface elements made a huge difference in their use.

Mobile Design Details

Designing for iOS7: Perils & Pluses

Like many other companies building mobile applications, we've spent a lot of time recently redesigning our iPhone app, Polar, for Apple's newest operating system. Through what turned out to be a rather lengthy process, we learned a lot about the good, the bad, and even the blurry parts of designing for iOS7.

As we began adapting existing elements of the Polar design to work with the overall aesthetic and design language of iOS7, one thing became really clear. We weren't satisfied to **just make things fit into iOS7**, we wanted to ensure we were actually making the design better as a result of these changes. In some ways, iOS7 made it easy to improve our design. In other ways it made things a lot harder – which is where most of our time was spent.

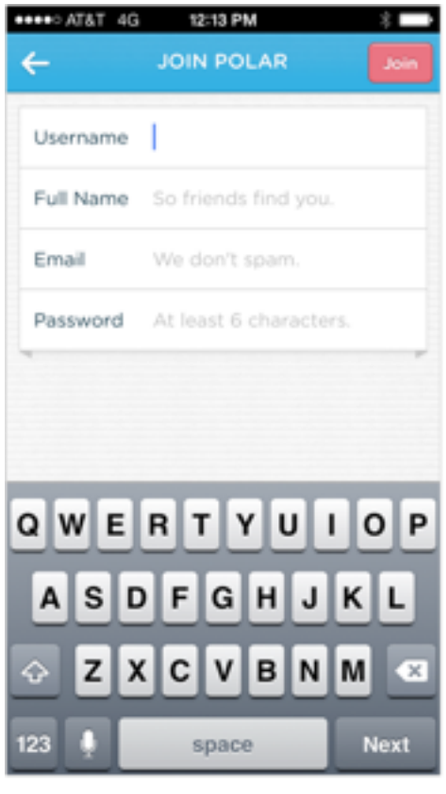
iOS7 Design Pluses

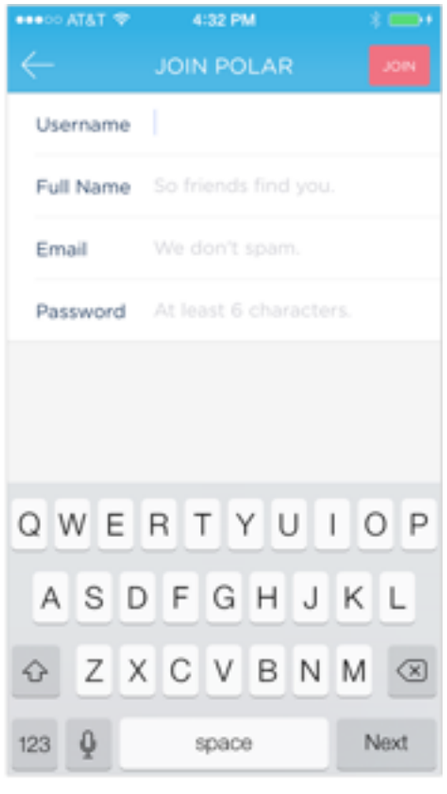
The design language of iOS7 is inherently simpler than the one Apple used in iOS6. On the surface, that would seem to make designing for it simpler as well. But in reality you end up needing to do more with less, which is not easy.

"True simplicity is, well, you just keep on going and going until you get to the point where you go... Yeah, well, of course."

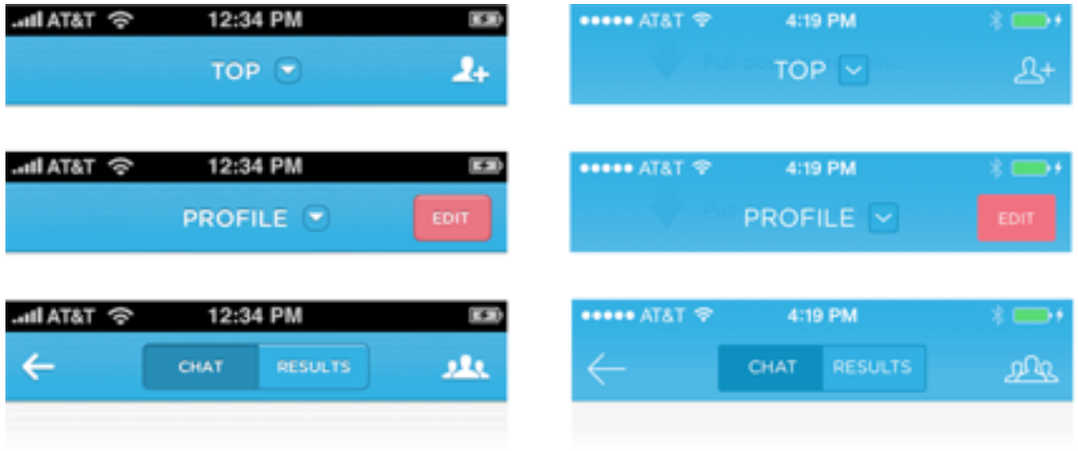
-Jonathan Ive

While we needed to use Ive's process of continual iteration for several of our design elements (details later), the work done by Apple's team also allowed us to quickly get to a better design with other elements. For instance, moving to the iOS7 style of input fields instantly made our forms feel simpler and fit in well with the rest of the operating system aesthetic.



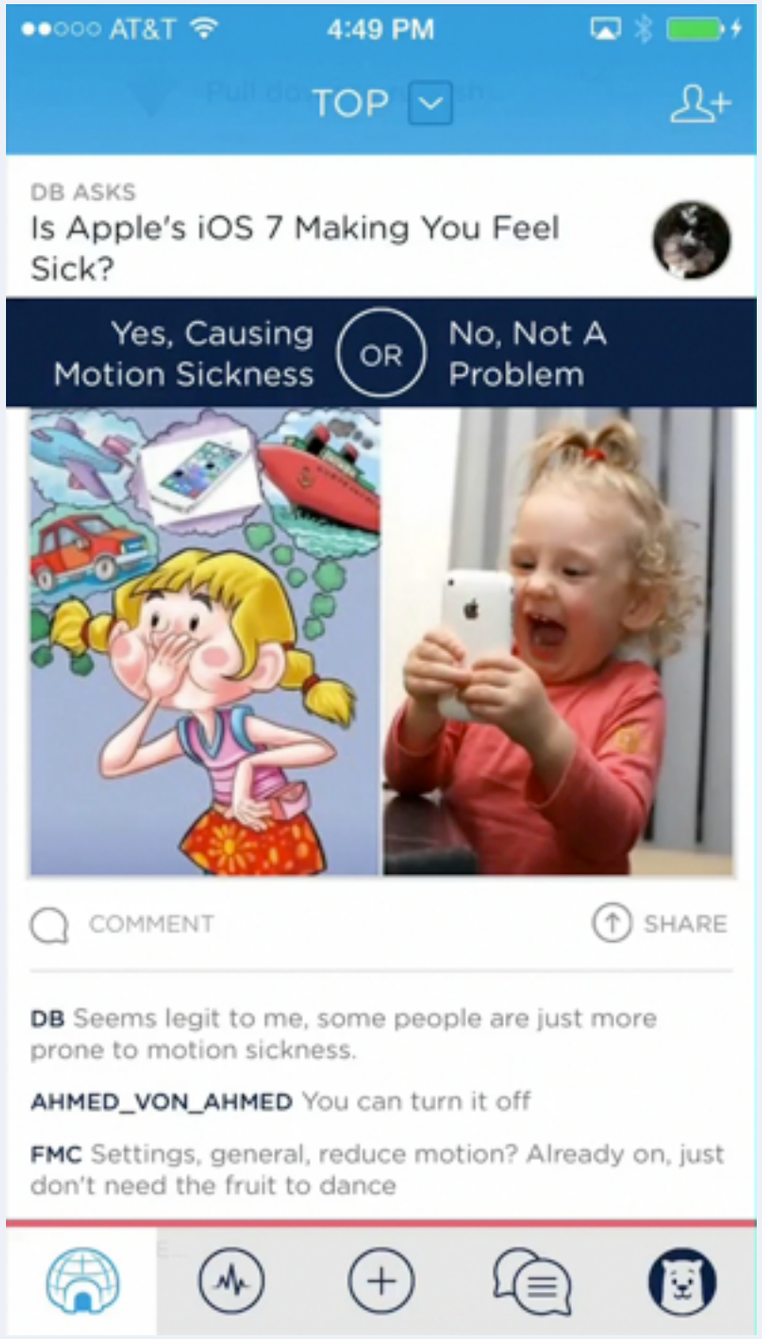


Headers, while requiring more work, also improved our existing design by forcing us to get down to the bare essentials. They also gave us an opportunity to take advantage of the translucency effects that define a lot of the iOS7 experience. By default, app headers are now transparent and can match up with the color of the OS system bar. This creates a single visual element at the top of an application and teases content below the header with transparency – both pluses. But these pluses also come with some new challenges.

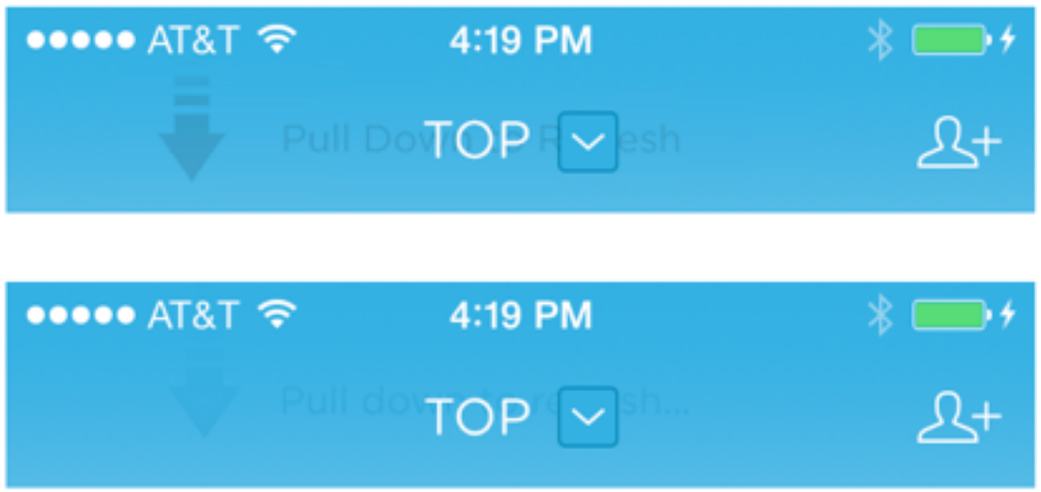


In order to create more screen space for content, Polar has always removed our headers when people scroll down through the list of polls. when they scroll up a certain amount, we bring the headers back so people can navigate around the app again. As you can see in the video below, the default transparency broke this behavior and we had to come up with a new solution.

We ended up sliding the header under a thin blue underlay we positioned below the system bar (see the video above for the full effect). When scrolling down with this implementation, just a thin blue system bar is left thereby maximizing screen space and retaining a bit of the app's style after the header is gone. but we weren't out of the woods yet... because of our custom pull to refresh elements.



Since the first version of Polar, we included a **pull to refresh gesture** that updated the content on our screens. With our new transparent iOS7 headers, these perviously hidden (below the header) UI elements showed through and made the text in our headers harder to read. We got around this issue with blur.



To ensure our **pull to refresh elements** below the header didn't make things harder to parse, we blurred all the elements below the header. This created a sense of depth through translucency without negatively impacting readability. So win/win.

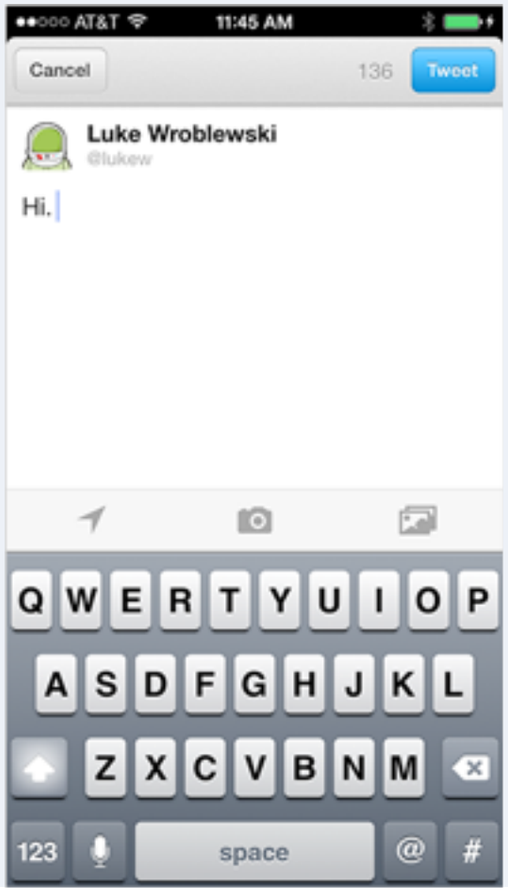
When it came to the forms and headers in Polar, the iOS7 design language made it easy to do the right thing. And I think we did end up with a better design instead of just an iOS7 design. With other elements of Polar, things weren't that easy.

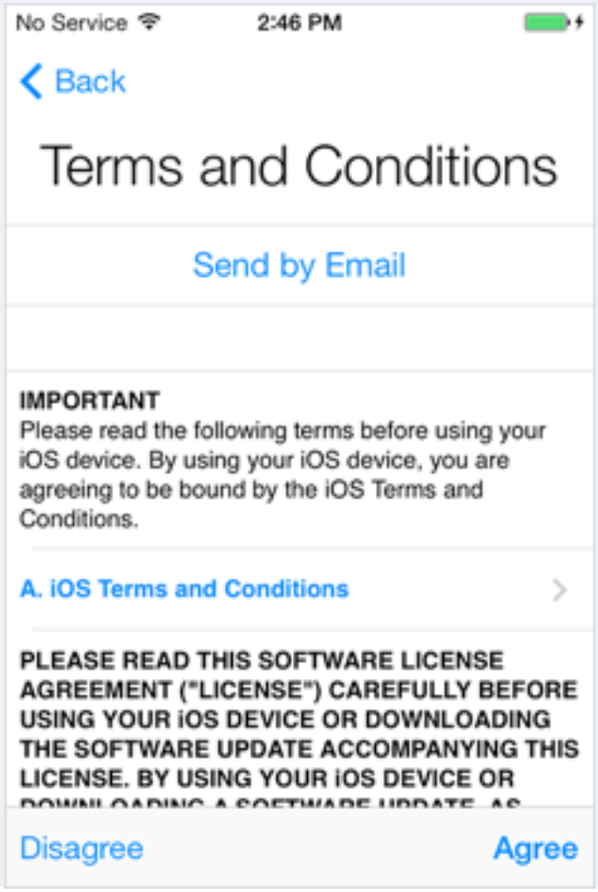
iOS7 Design Perils

The simplicity of iOS7's design language comes at a cost: a reduction in the amount of **visual elements** designers can use to create hierarchy and thereby **understanding.**

To explain that a bit further, how people makes sense of what they see gives designers a set of attributes to play with to create meaning within a design. Elements like color, size, and texture can create similarity, differences, and hierarchy within a layout. When these elements are "flattened", some of this vocabulary goes away. It's like losing a set of words, you have to work harder **to communicate** with a more limited vocabulary.

You can see a lot of places in iOS7 where the flat design style makes the hierarchy of actions less clear. For instance, compare Twitter's compose screen on iOS6 to the one on iOS7, the lack of strong contrast between elements makes it less immediately apparent where the primary call to action (Tweet) is located.



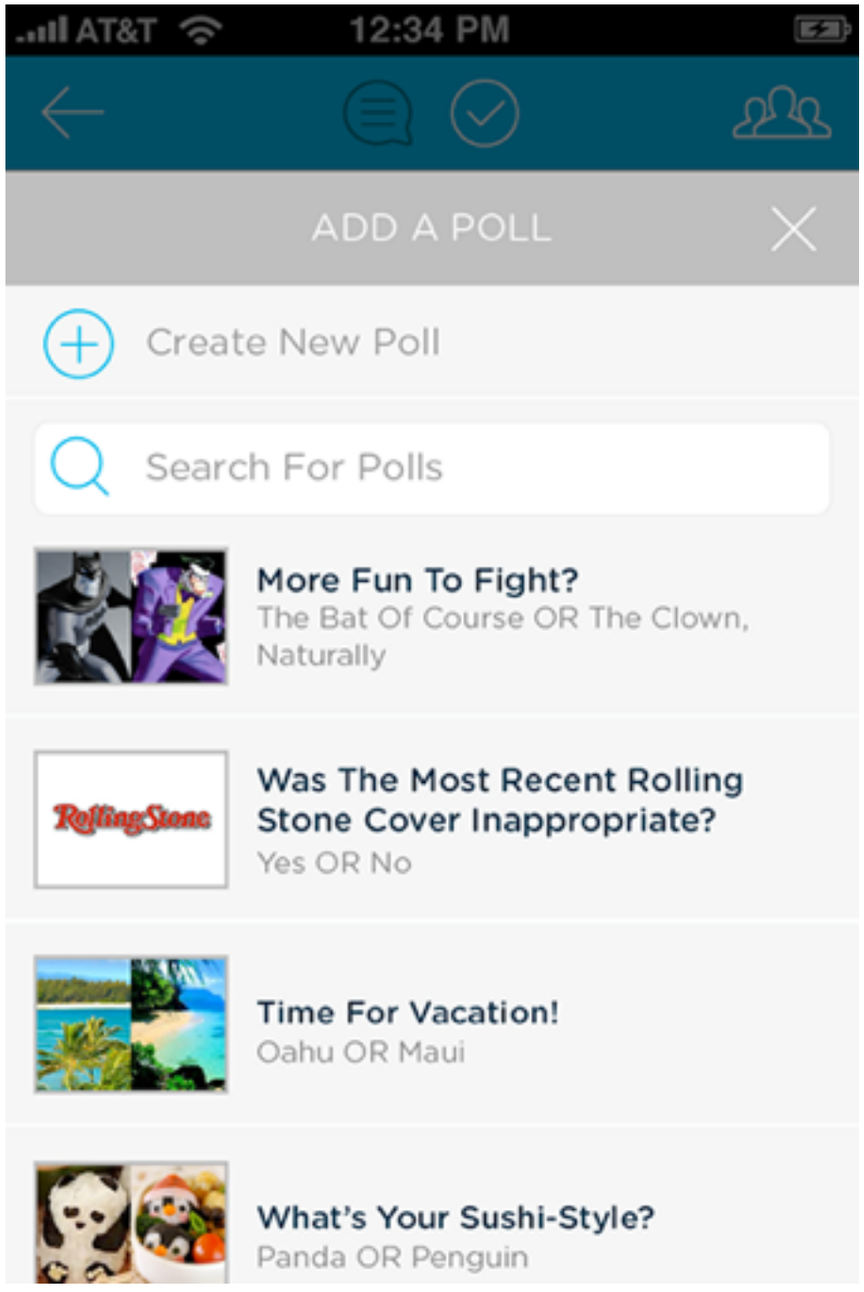


In parts of iOS7 it can be hard to determine what the primary call to action is because it is only distinguished through subtle visual relationship differences. For example, in the Terms and Conditions screen every iOS7 user sees, Agree is just a bit bigger and just a bit bolder than other elements on the screen despite being the primary action.

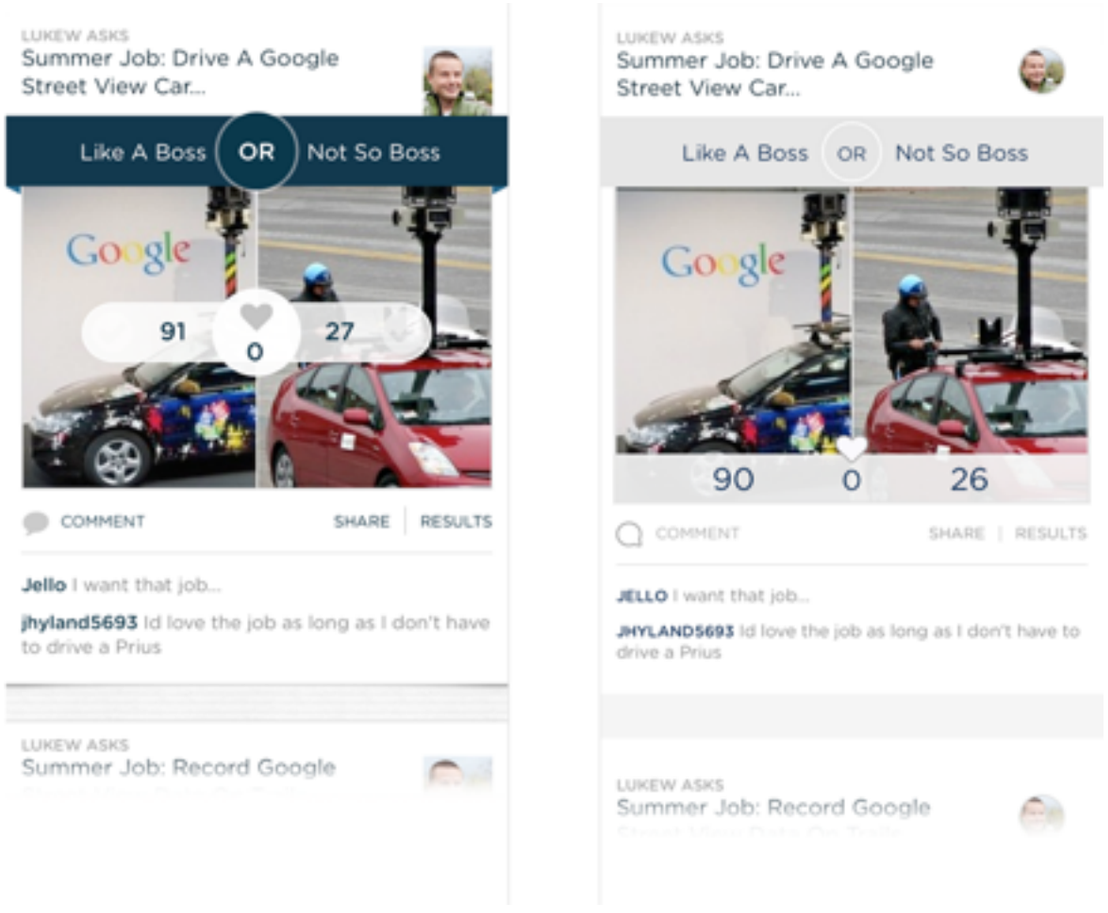
Of course, it's still possible to create effective visual hierarchies with less contrast between visual elements but it's often harder to do so. Which bring us back to Jony Ive's quote at the start of this article: it's all about iteration.

In some of our earliest explorations of an iOS7 design, flattening things out resulted in less hierarchy than we felt was needed to make actions distinct. You can see

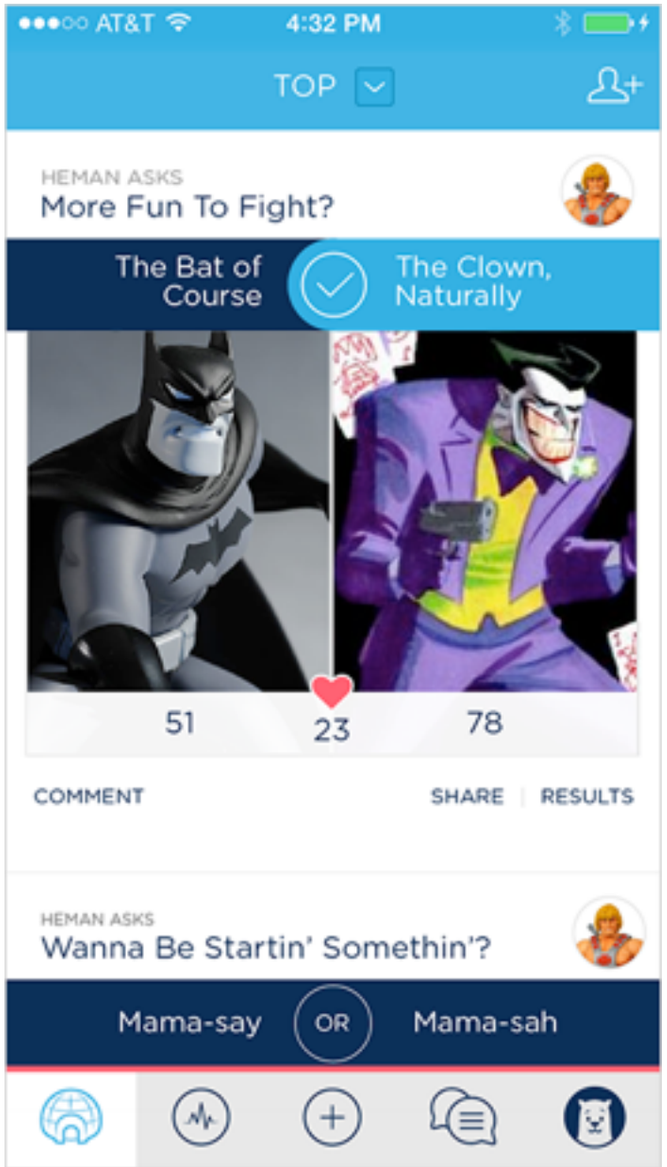
this situation in the example below. The Add, Search, and Create actions all seem to blend together as we're relying on small visual changes to distinguish these actions.



We faced the same challenge in our list of polls. In our iOS6 design, we had relied on depth (shadows) and texture to separate items in the list from each other. When we adapted to an iOS7 design, simplify flattening these elements once again created hierarchy issues. A number of of visual elements blended together too much making it hard to distinguish individual polls in the list.

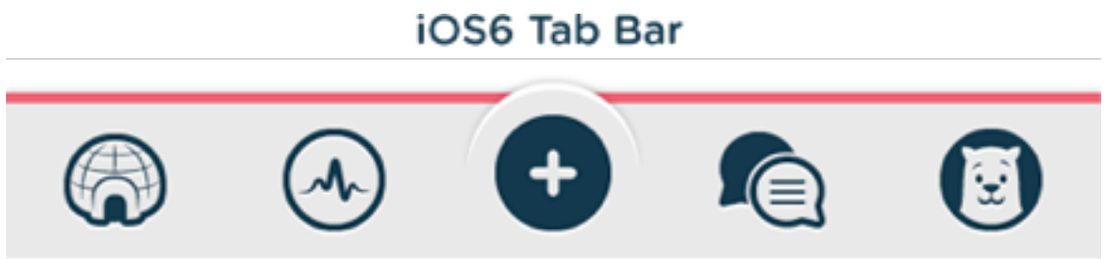


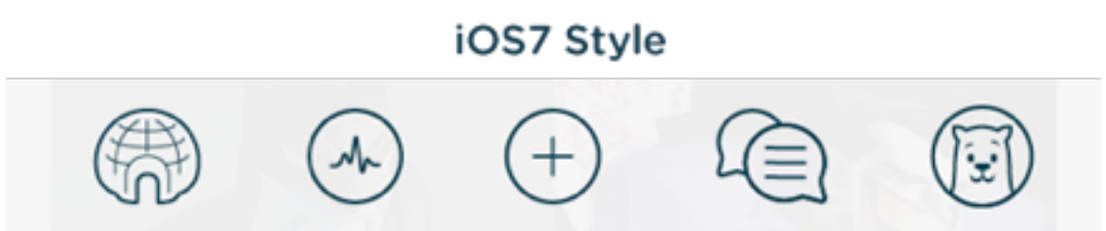
It was only after we started to remove visual elements from the poll list that the flatter, simpler look began to work well. We took out the elements that had been background textures, eliminated icons, and introduced a bit of color to separate out actions like Comment and Share.



Removing texture and depth forced the rest of the visual design to work harder to create meaningful distinctions between the various elements on screen. I think this is a key reason why designing for iOS7 is harder. It forces you to simplify in order to provide the same clear visual communication using less visual relationships.

Another area that required significant iteration was our Tab Bar. Thanks to **Thanh's** amazing icon work, our Tab Bar not only provided quick access to key features inPolar but strongly reflected our personality as well. When we simply tried to adopt iOS7 styled outlines for our icons, two things went wrong.



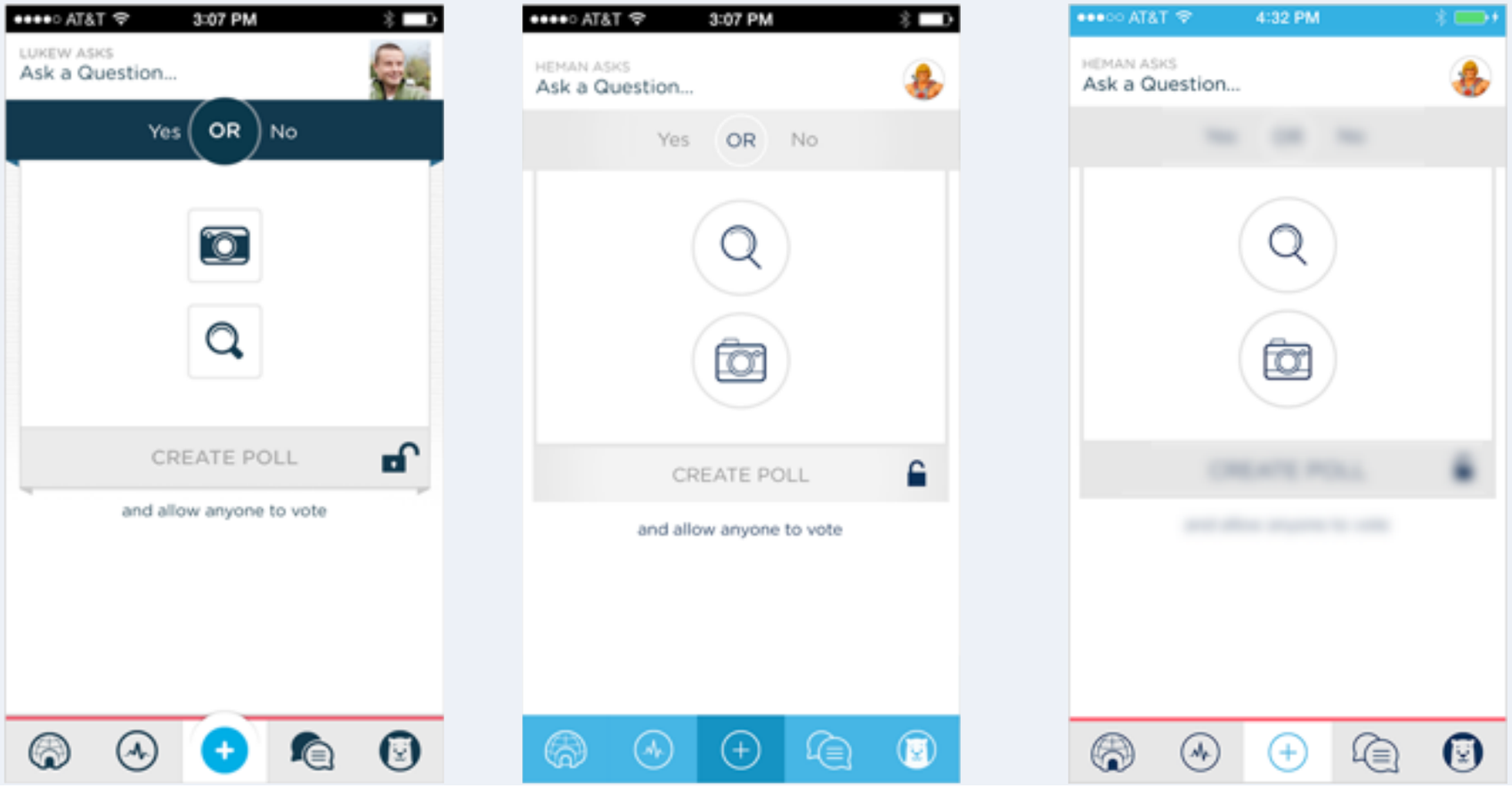


First, it made the icons harder to parse quickly. Aubrey Johnson recently pointed out **hollow icons take more effort** to process and we ran a series of polls that seemed to prove out his hypothesis.

But even without these theories and data, it was clear the Tab Bar icons were communicating less effectively. Secondly, we lost a lot of our personality. So it was back to iterating until again we found a Tab Bar design that retained our personality and felt at home on iOS7.

The balance of your application's personality and the personality of iOS7 is a great reason to not simply change over to an "iOS7 design". Take the visual vocabulary iOS7 provides as a language but find your own voice.

We also encountered an over-abundance of "flatness" in another one of the key screens on Polar: create a poll. When we first adapted this screen to an iOS7 style, we lost the priority of actions that we counted on depth and texture to establish in our iOS6 design.

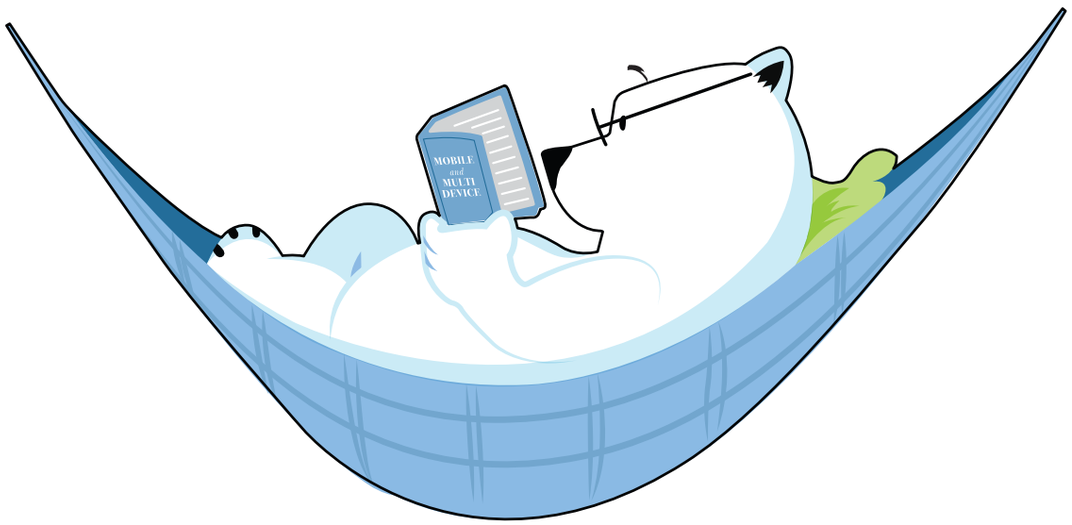


To create a clearer hierarchy of what to do first when creating a poll, I suggested blurring out the secondary elements and putting the focus on the things people need to do first. This was an attempt to use the translucency and blurring effects found in other parts of iOS7 to add some much needed hierarchy to a critical screen design.

Ultimately, we backed away from this approach based on **usability concerns** and the time it would take to fully explore and build. But we're still iterating, so this and few more iOS7-designed elements might make it into Polar soon. Install the app to see how our current iOS6 design morphs over to iOS7 in the coming days. We've certainly enjoyed the journey and think you will too.

Cross Device Design for the Web

Linking Mobile Web & Native App Experiences



It's often said that the power of the Web lies in the links that enable connections between people, places, and things. At first glance, those same links don't seem to work with native mobile applications. But mobile Web and app experiences can be connected – it just takes a bit more work.

"Links don't open apps."

-Jason Grigsby

While **Jason's right**... a standard Web link won't open a native mobile application on its own, we can set up a system that allows people to go from the mobile Web to

a mobile app fairly easily. Here's how we did it for our latest iOS app, Polar.

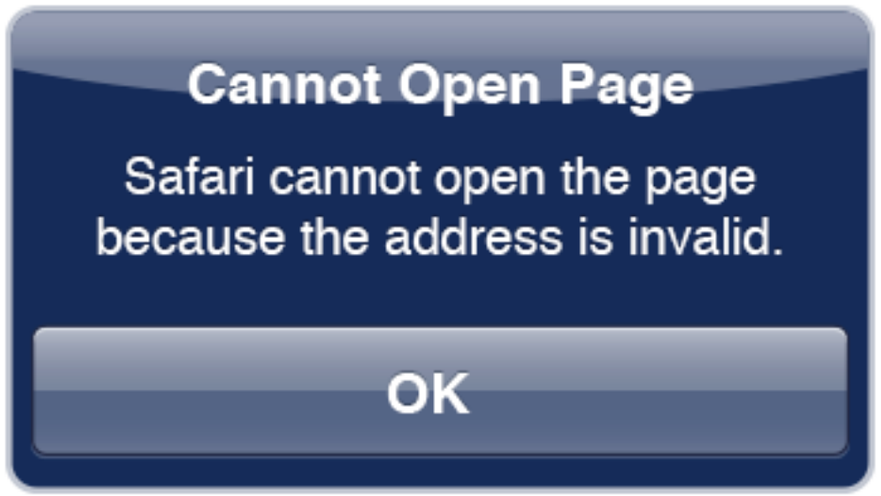
Every poll created in the native Polar iOS application is also accessible on the Web. We created a mobile first responsive Web design that allows people to view and vote on polls using the Web browser and device of their choice. This means anyone who doesn't have our native mobile app can still vote and share their opinion quickly and easily.

However, if someone has already downloaded our iOS application we take steps to get them into the app where they're more likely to be logged in and can have the full Polar experience.

A Custom URL

To start, we set up a **custom URL scheme** for Polar that allows us to launch the app from a Web browser (or any other app on iOS). This link takes the form of polar://polls/2246/.

While this custom URL scheme allows us **to open an iOS app** to any screen we like from the Web browser, it doesn't work on all mobile platforms and actually throws ugly errors on iOS when an app is not installed.



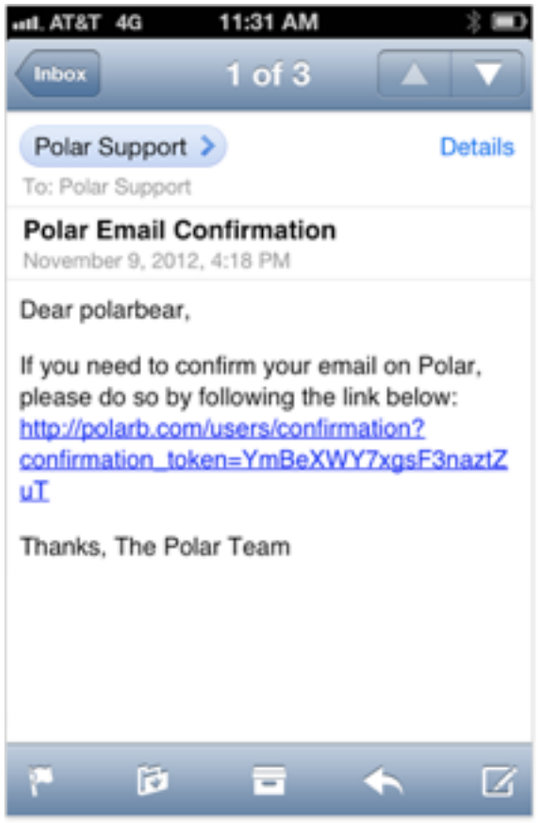
So we can't simply replace standard links with custom URLs -even on the platform where our native mobile app resides. We need to know if someone has our application installed and **only redirect them** to it then.

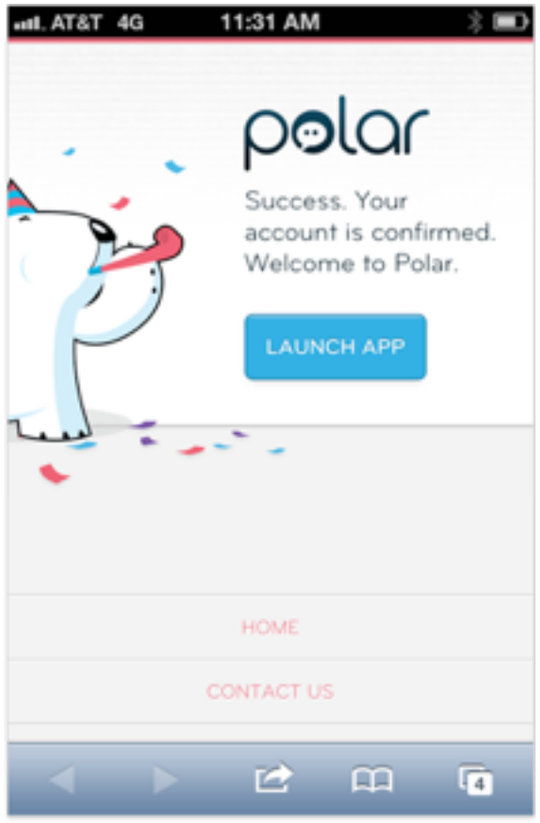
We also can't check to see if someone has our app installed. Imagine if any Web site could figure out exactly which apps you've got on your phone -not good from a privacy perspective. So no operating systems allow it.

Setting Cookies

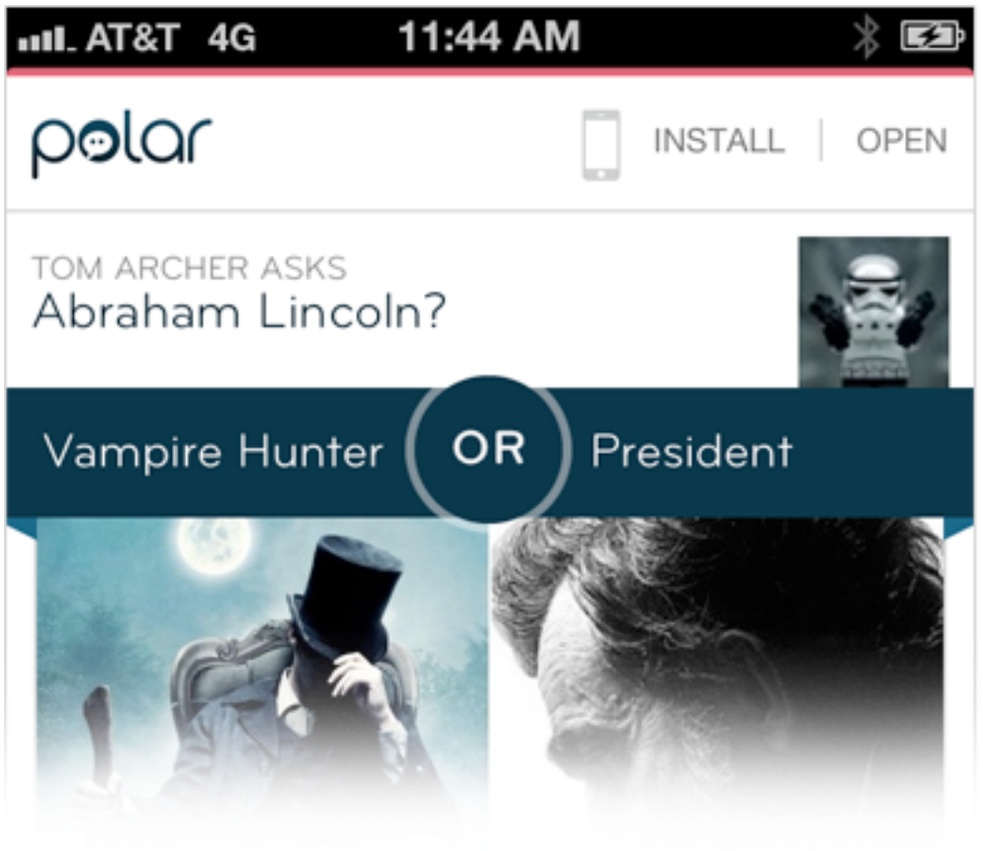
Instead we have to determine if you have our app ourselves. We do that on the Web **through cookies.** When someone signs up for Polar, we have them confirm the email address they used to sign up for the

site. To do this, we send an email with a confirmation link that opens in a Web browser and tells us they are in possession of the email address they provided. Along the way we also set a cookie in that browser that tells us this person has installed our app.



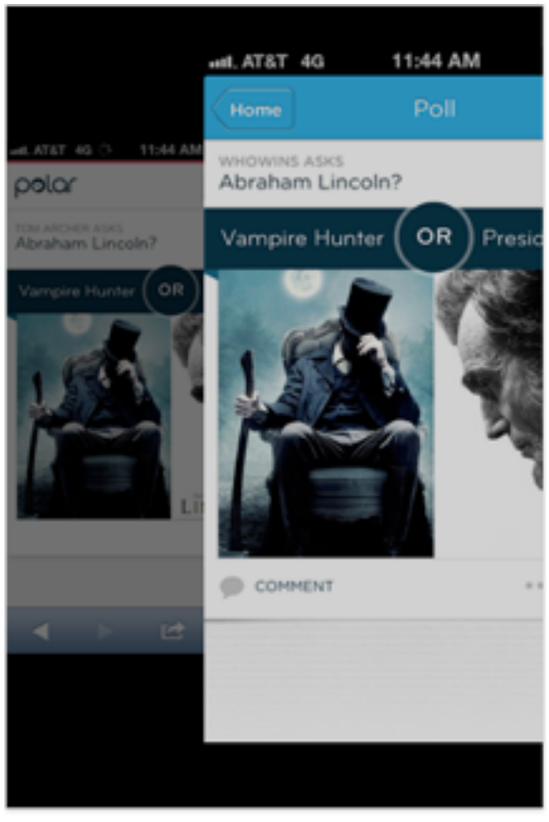


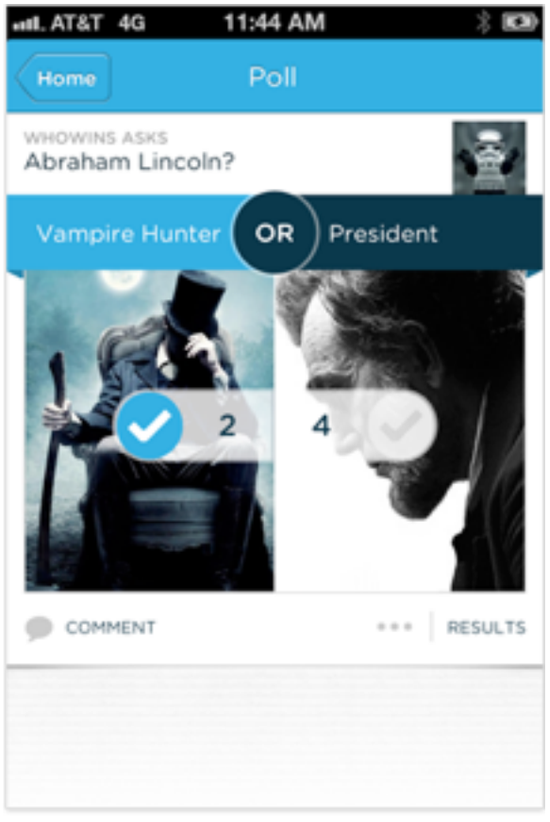
Since not everyone that downloads Polar will confirm their email address and many people are likely to use more than one browser and device to respond to email, we can't only rely on setting the cookie during email confirmation. We also have to allow people to tell us when they have the app. For that we include an OPEN link on poll pages when they are viewed on iOS. Tapping this link, not only opens the poll someone is viewing in our native application (using our custom URL scheme), but also sets their cookie to tell us this person has our app installed.



Redirecting to Native

Because we set a cookie under these two conditions, we can now check for the presence of that cookie when someone opens a Polar link in their Web browser on iOS. If our cookie tells us that they have the app, we'll **redirect them automatically** to our native application using our custom URL scheme.





This technique also works for embedded Web browsers inside of other native applications, albeit a little differently. For example, if someone encounters a Polar poll in the native Twitter application on iOS and views the link in Twitter's embedded Web browser, they'll also be redirected to our app. In order to not switch people between apps without warning, the embedded Web browser uses a simple confirmation dialog to let people choose to open Polar or not. (I wish Apple made this message a bit clearer.)

Like many solutions on the Web, this technique isn't foolproof. We might end up with an inaccurate cookie because someone deletes the app from their phone or taps OPEN on accident. In this case, they'll be presented with the cryptic iOS error we saw earlier, need to scratch their head, and then dismiss it. (Again it would be great if Apple provided something a bit more informative in the error message.)

Android

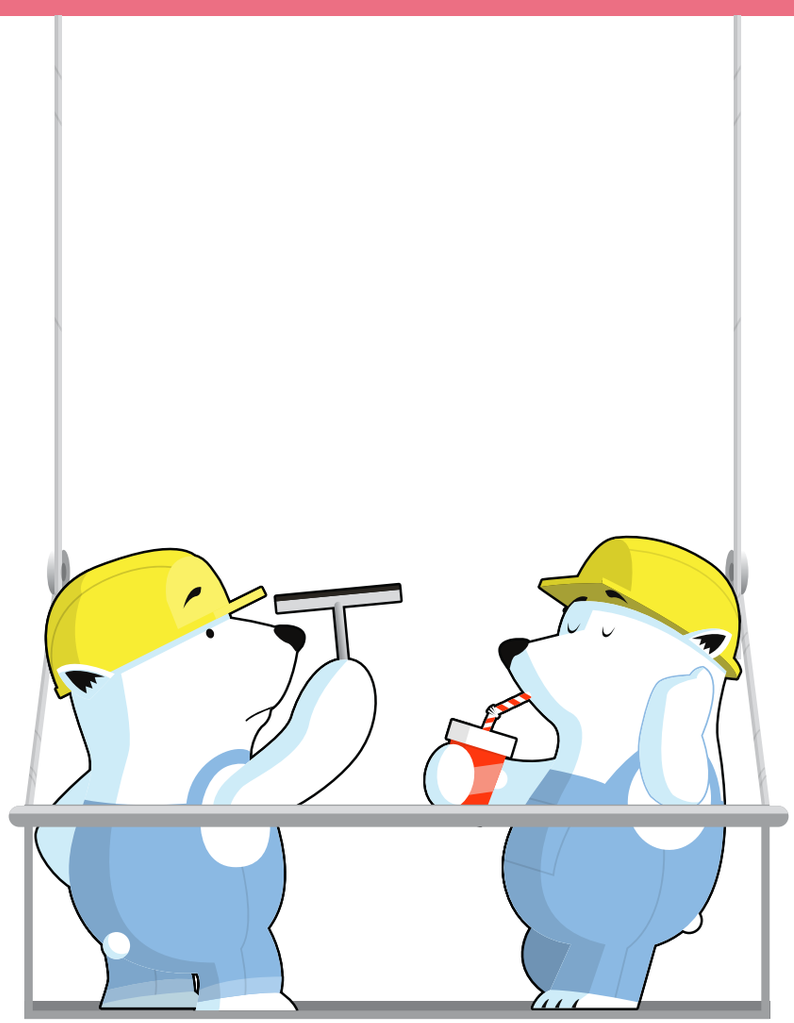
Does this approach also work with Android apps? Yes. We did something similar on Bagcheck to open barcode scanning apps on iOS and on **Android.**

Making links open apps on Android doesn't require cookies, you can use **intent filters** to intercept a link and **open an Android app from the browser.**

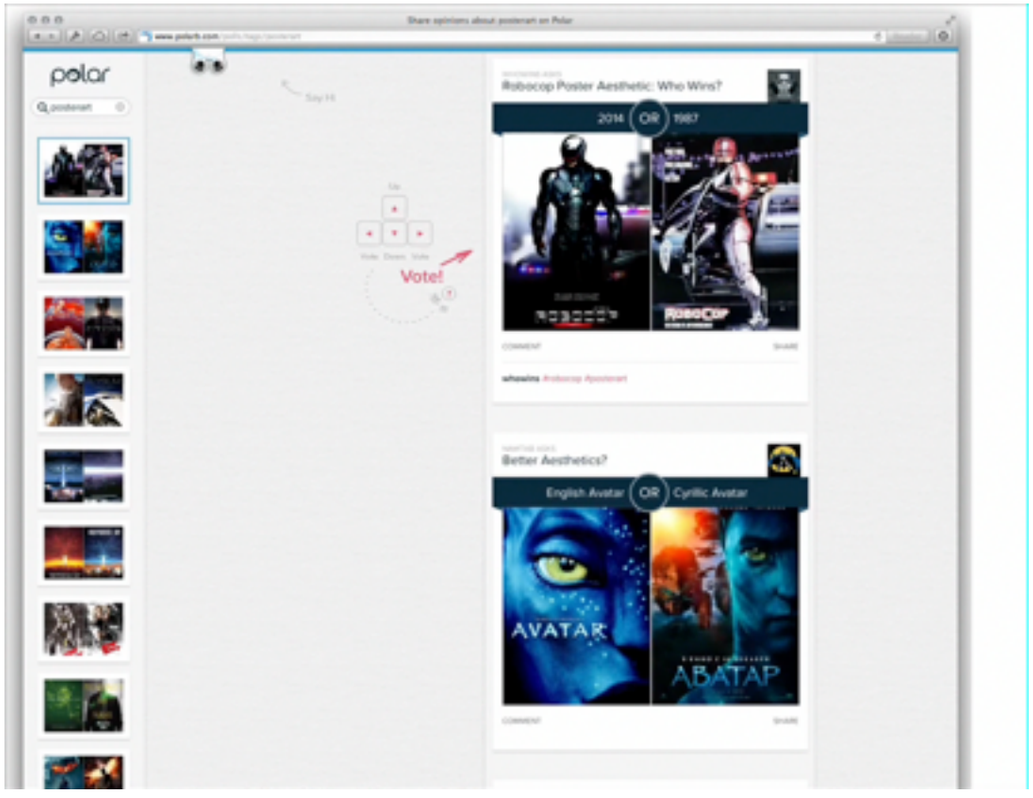
Smart Banners

Why not use Apple's Smart Banners to open iOS apps instead? Smart banners have some advantages over our method. They actually know if an app can be installed or is installed (but won't tell us) and can adjust the action within the banner or hide the banner accordingly.

But smart banners only show up on iOS6, so we can't reach all the iOS devices we'd like nor reach people on desktops and laptops with iTunes. We also can't adjust the size or contents of smart banners ourselves. If your primary goal is getting people to install or open your native app, that might be fine. Our primary goal is allowing people to vote on a poll and Apple's banners get in the way. Despite these issues, smart banners might be a real good solution in other situations.



New Layouts for the Multi-Device Web



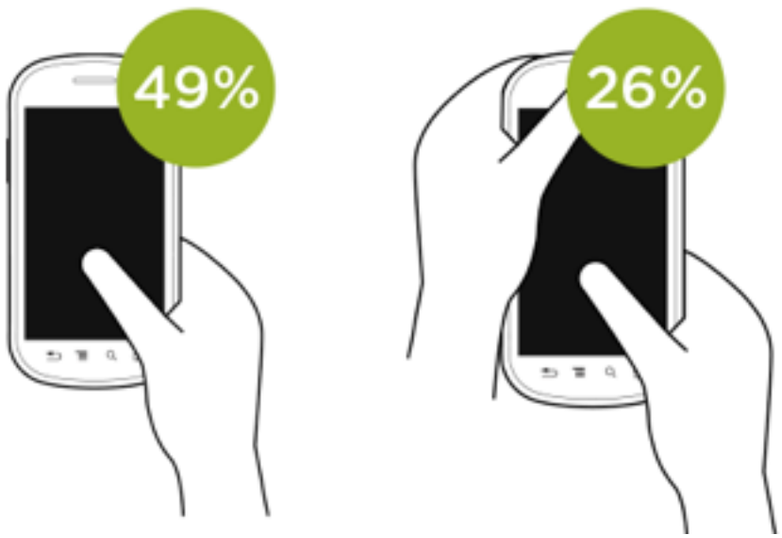
Most Web page layouts rely on design patterns created for laptop and desktop computers equipped with a mouse and keyboard. As the **variety of devices** being used to access the Web has grown, these patterns haven't been keeping up. Designing for today's Web means considering single-handed thumb use on smartphones, two handed touch interactions on tablets, mouse and keyboard input on traditional PCs, hybrid devices, and more. Web layouts have to evolve to support this new reality.

The New Reality

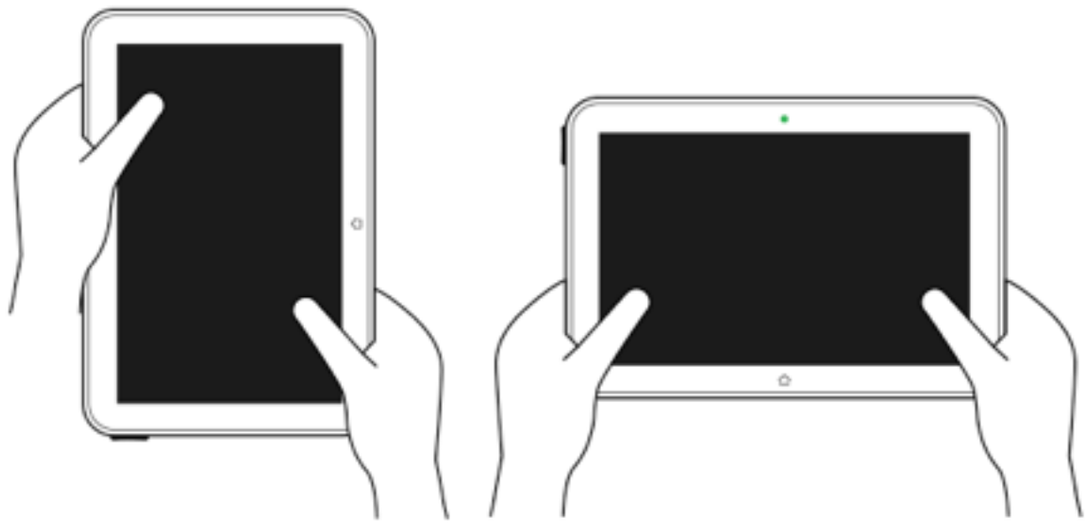
As device diversity increases, so does the number of ways people interact with the Web. To understand the impact of a

specific device on interaction design and layout, we can look at three things: output as mostly defined by a screen, the input types available, and common postures or modes of use (strongly influenced by input and output capabilities). For example, consider the modern smartphone.

Todays' smartphones are defined by palm-sized screens (usually 3-5 inches diagonally) of varying pixel density, multi-touch input, and predominately one-thumb use with the device about a half arm's length away. A recent study of 1,333 people using smartphones on the street found that about 75% of smartphone use is one thumb. Web layouts need to take this reality into account.

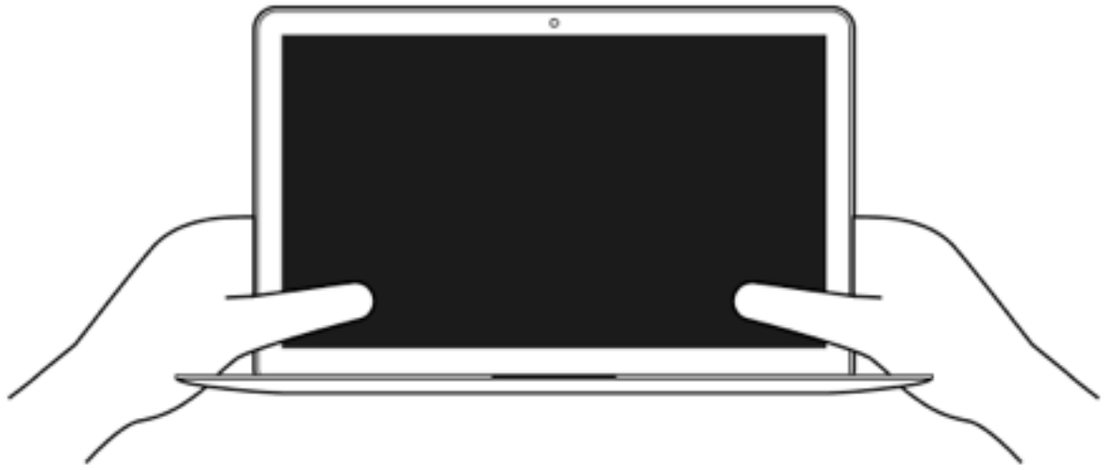


Tablets also feature multi-touch input but they have larger lap-sized screens (7-10 inches diagonally) that have an impact on how they get used. With a larger screen one-handed use is less comfortable so two-handed use is more common. With two-handed touch interactions, the sides of the screen are the easiest to access with simple finger gestures. As tablets continue to grow, Web layouts also need to take this reality into account.



Hybrid devices that feature touch, mouse, and keyboard input are increasingly common as well. On these devices, touch interactions are more frequently used than most people assume. A study by Intel found 77%

of interactions on these devices used the touch screen, 12% used the mouse, and 8% used the keyboard. Once again the size of the screen and people's posture influences interaction design and layout. To avoid the fatigue that comes from holding your arm up in the air, people rest their arms or elbows on a surface and once again rely on the sides of the screen for touch input.



Alongside these diverse devices, we still have lots of laptops and desktops with traditional mouse and keyboard inputs and design considerations.

So how do Web layouts adapt to this new increasingly diverse reality? Here's one attempt we just launched for Polar.

Multi Screen & Multi Input Design

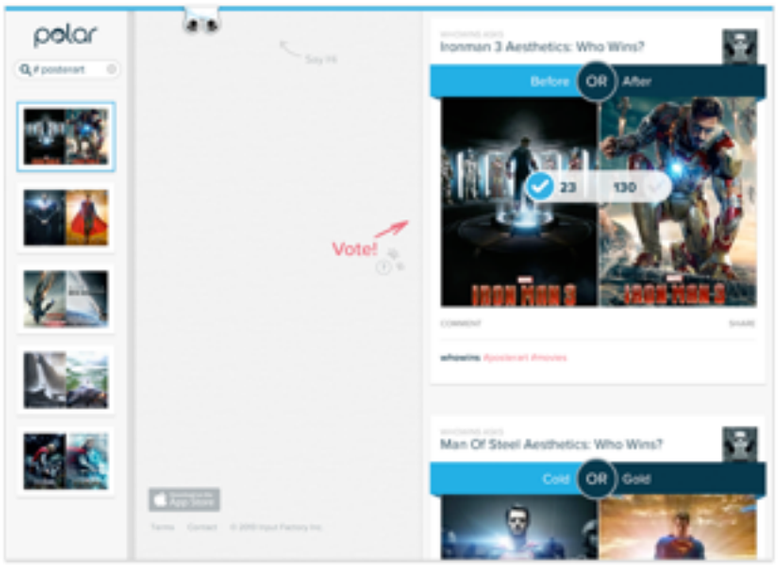
Topic pages on Polar were designed to adapt to not only different screen sizes but to different input types as well. The end result is a Web interface that aims to fit into the reality of Web use today. In particular, the human ergonomics of how people interact with different devices: one-thumb use on smartphones; two-handed use along the sides on tablets; and mouse scrolling, clicking, and keyboard shortcuts wherever they make sense.

On a smartphone, Polar topic pages focus on **one-thumb use** through large touch targets and a single scrolling column of content. We also aimed to minimize mobile download times by only loading more content when people ask for it. On other devices, we automatically load more content as you scroll.

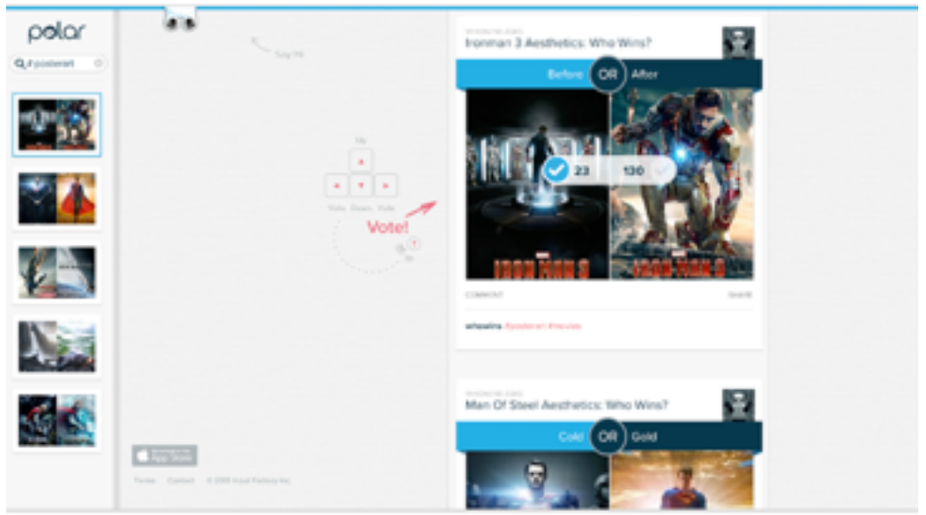
On larger smartphone screens and small tablets (call them phablets if you must), we increase the size of the content and maintain an emphasis on touch input. However, we also include keyboard support for voting as well. But we're not promoting it… not yet.

On tablet-sized screens, we introduce a browsing column on the left side of the screen for quickly exploring content and move the content column to the right side. This is an attempt to accommodate two-handed use of tablets where people's hands sit comfortably on the sides of the device.

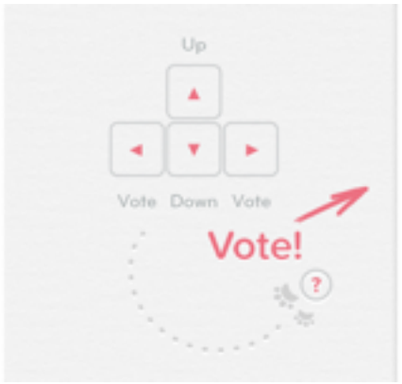
This arrangement carries over into landscape tablet views and the larger screens you're more likely to find on laptop and hybrid devices. Once we cross a screen size more likely to be present on a laptop, we introduce an affordance that let's people know they can use their keyboard to vote as well. The up, down, left, and right arrow keys allow you to move through the content list and take action.



Our initial goal was to provide a more significant promotion for keyboard voting if we detected a keyboard was present on a device. After going down this path for a while, however, we found no reliable way to detect if a keyboard was attached. So instead we downplayed the keyboard promotion and made it visible to everyone that fit within a screen size threshold. Using screen size as a proxy for keyboard presence is not ideal but sadly its all we have to work with today.



On very large screens, we actually do promote keyboard voting more aggressively. Even if touch is available on a desktop-sized screen, chances are the screen is a decent distance away from the viewer, whereas as the keyboard is right next to them, and thereby more comfortable to use. As a result, we feel more confident promoting the keyboard voting feature.



Comfortable to Use

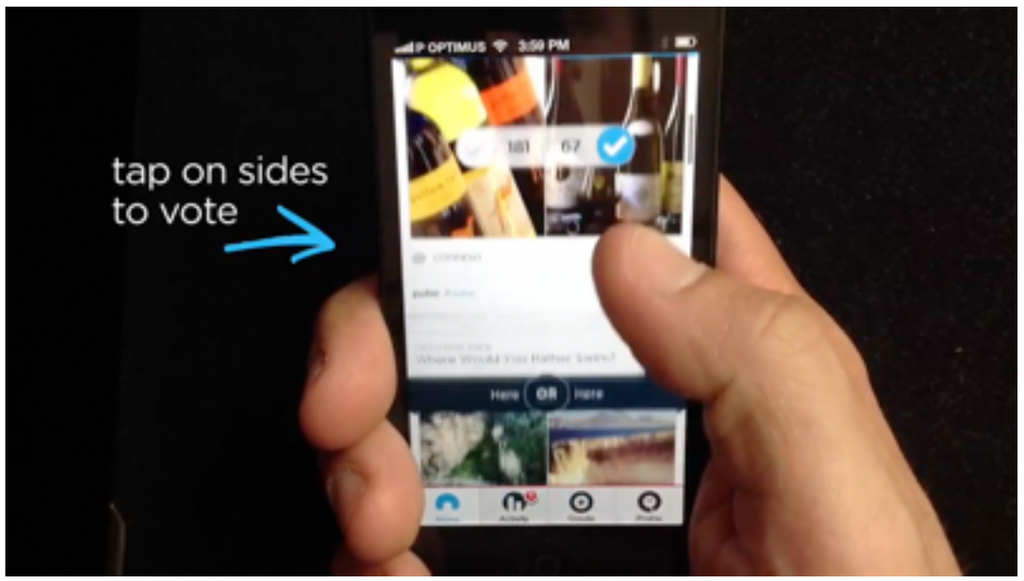
Across all these devices from smartphone to desktop, our criteria for the Polar interface was “comfortable to use”. That is we emphasized human ergonomics over typical visual design conventions. We wanted a design that was comfortable for phone, phablet, tablet, hybrid, laptop, and desktop users and adapted the interface as needed to align with how people use these distinct devices.

The potential downside of this approach is that “comfortable to use” doesn’t come through unless you are actually using the application. Looking at the Polar interfaceon a laptop can be a bit disconcerting because we’ve essentially left the middle of the page “blank”. Just about every other Web page online centers their page layout and leaves the sides as empty columns.

MULTI INPUT DESIGN DESKTOP DESIGN

But is this a desktop convention that needs to change? Are we better off adapting our interfaces to the way people actually use devices instead of clinging to “best practices” defined nearly twenty years ago? And just what is the difference between tablets and laptops in a world of hybrid devices? Time will tell. In the meantime we’ll keep experimenting on these devices and **many more...**

Designing for Thumb Flow



Lately I've become increasingly interested in the ergonomics of software design. That is, how human factors influence both the interactions and layout of an interface. In New Layouts for the Multi-Device Web I outlined the impact touch interfaces can have on layout across diverse screen sizes. This time I want to focus on interaction and designing for the flow of a thumb.

The Ergonomics of Software

Traditional graphical user interfaces (GUIs) are controlled by indirect manipulation through a mouse, keyboard, or joystick. So we design software for GUIs to work with digital representations of our movements by relying on interactions and layouts that accommodate mouse cursors not hands.

Human factors do play a role in GUI design as we need to consider a user's distance from the screen, the range of motion they can get from a mouse, proximity of keyboard actions, and occasionally environmental conditions like lightning. But the importance of ergonomics increases dramatically when indirect manipulation gives way to direct manipulation.

Interfaces that support direct manipulation through multi-touch, gestures, and voice are different. These **natural user interfaces** (NUIs) make direct use of our hands, fingers, and bodies. So human factors have a much more direct impact on software interactions and layout.

When you add in the diversity of device sizes out there today and their impact on user posture and its not surprising that a company like **Netflix refers to their software experiences** in terms of human ergonomics: 10 foot user interface guy (for TVs), two foot guy (laptops) and 18 inch guy (tablets).

In a world of diverse devices and direct manipulation, human scale matters. And human scale on mobile devices is increasingly measured in thumbs.

Thumb Flow

In **a study** looking at over a thousand people using mobile devices in the street, Steven Hoober found that about 75% of people's interactions with a smartphone were managed with a thumb. Whether holding the device with one or two hands, it was the thumb that was doing all the work.

So when it comes to designing for mobile, it makes sense to follow the thumb. Let me illustrate with an example from our mobile application, Polar, which was **designed and tested for one-thumb use.** Polar is a fun way to collect and share opinions so we wanted to make that process as fast and easy as possible - especially with one thumb. You can how we went about it in the video below.

Dragging your thumb up on the screen reveals new questions you can vote on. Tapping on either side of a question with your thumb allows you to respond. Once you get to the end of a list, just pull up on the screen with your thumb again to reveal more questions.

The first time time you do this, a tip slides up from the bottom letting you know that questions you might not

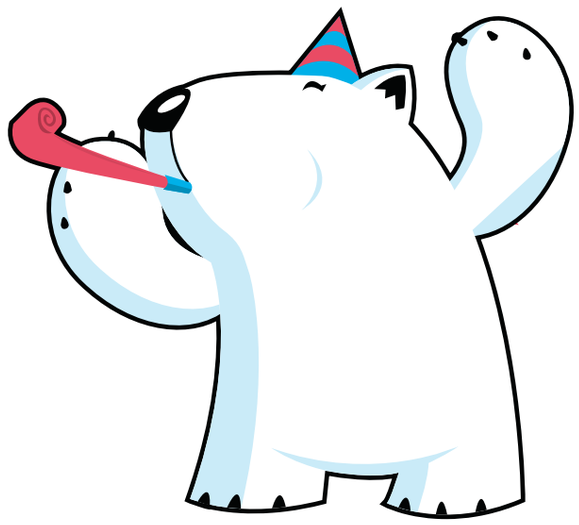
be interested in can be skipped by swiping across them with another simple thumb gesture. It's important to note that we're showing this tip to you only after you've scrolled and voted on a few things –not before you ever use the application.

Now that you've made it down the list a ways, you might want to explore other parts of the application. To do so just drag your thumb down on the screen. Once you scroll the list a bit, we slide the main navigation menu down for quick access sparing you the need to keep scrolling in order to reach it.

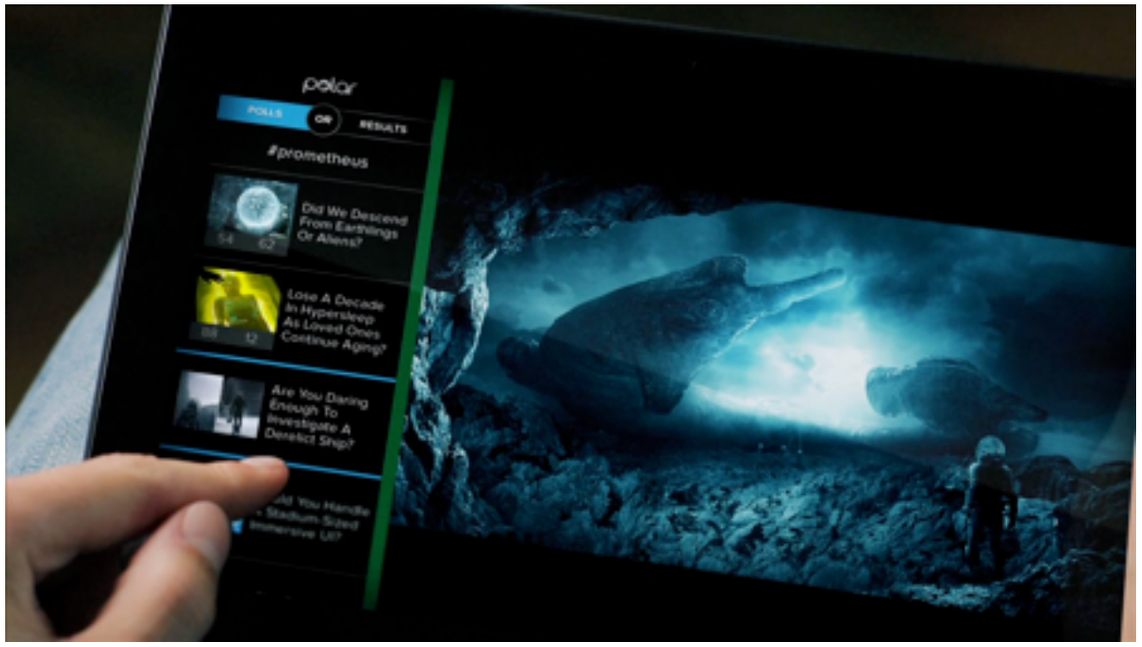
Want to refresh the list to see the latest questions being asked? Just pull down on the list to reload it. Looking at all these interactions together, you can see just how much you can get done with one thumb:

1. **Swipe up to move down the list**
2. **Tap on either side of a question to vote**
3. **Swipe up to load more**
4. **Swipe down slightly to reveal the main menu**
5. **Swipe down more to refresh the list**
6. **Swipe across to skip questions**

Making all this possible requires comfortably sized controls and a lot of **one-handed testing**. But in return, people will be able to easily use your application anywhere and everywhere they are – all they need is their thumb.



Designing a Companion Web Experience



These days, the Web is always with us. Thanks to an **ever-increasing** number of always-on devices with great browsers, the Web is a constant companion in our lives. This continuous connectivity allows us to build new experiences that rethink how the Web can work -not just on one device but across many. To explore these possibilities we teamed up with Microsoft to create a **Companion Web** experience for Polar.

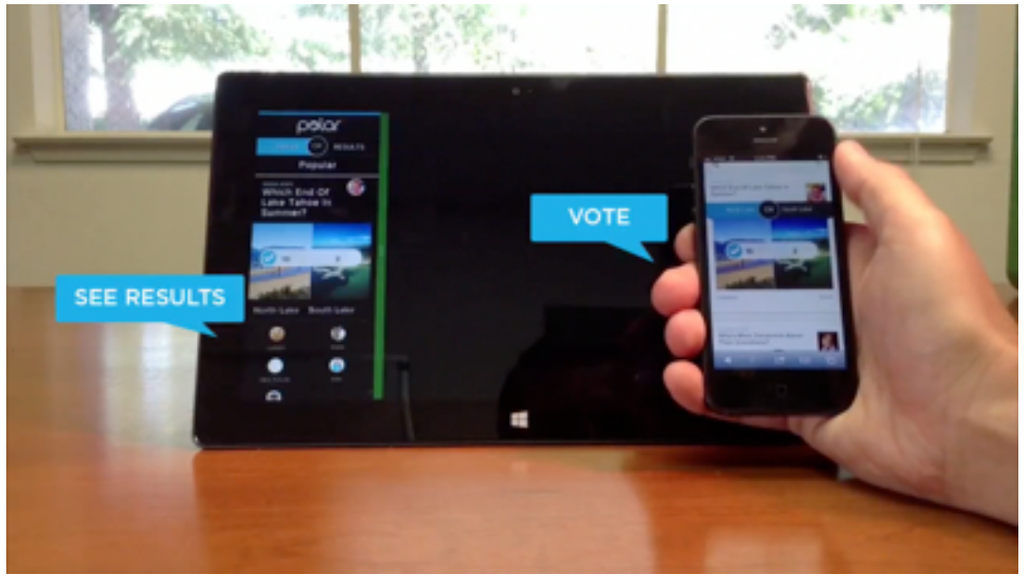
The Companion Web

From hyperlinks to social networks, the Web has always been about making connections between information, people, and now... devices. Whether tablets,

smartphones, or TVs, today's networked devices ship with the best Web browsers we've ever had to work with: Internet Explorer 10, Chrome for Android, and Safari for iOS for starters. The combination of better browsers, new devices, and how they're used allows us to rethink what the Web can be – to create new experiences for this new reality.

Consider **more than 80%** of smartphone owners in the United States use their device while watching TV and even back in 2011, **20% of time spent on a smartphone** was in front of a television. Despite people's frequent simultaneous use of these screens, they've remained separated by different operating systems, input devices, and apps. So while the amount of connected devices in our lives has grown, their connections to each other have not kept pace.

The Web can help by connecting our screens and applications so we can build cohesive experiences that take advantage of what each device in our lives does best. These Companion experiences represent the next evolution of the Web and we're excited to be working with the Internet Explorer team at Microsoft to explore this new terrain for Polar.

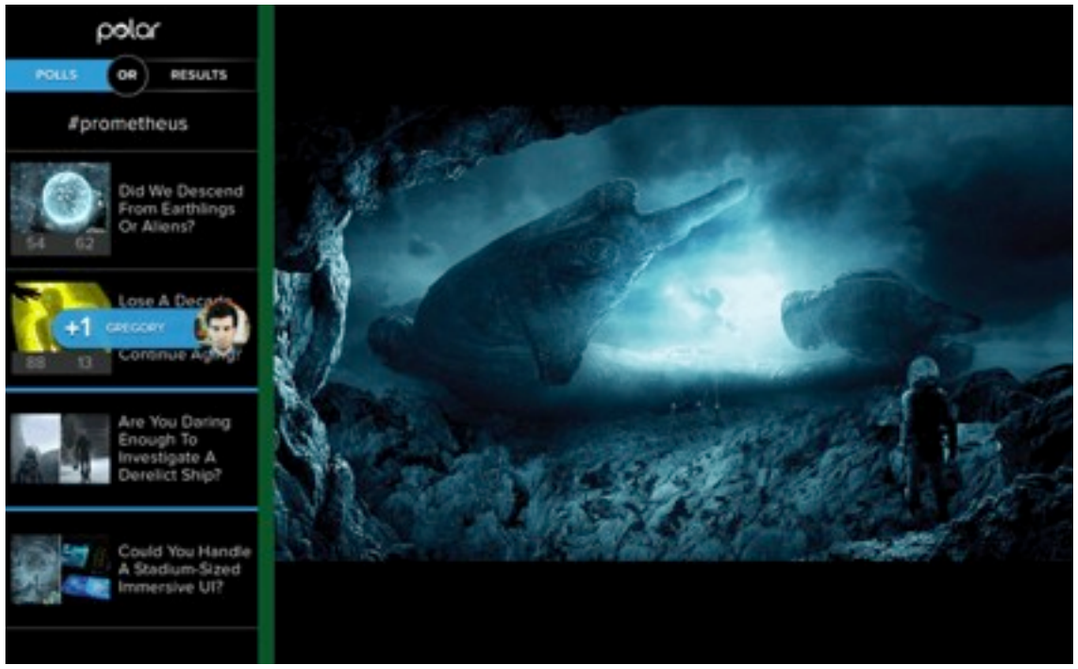


The Polar Companion Experience

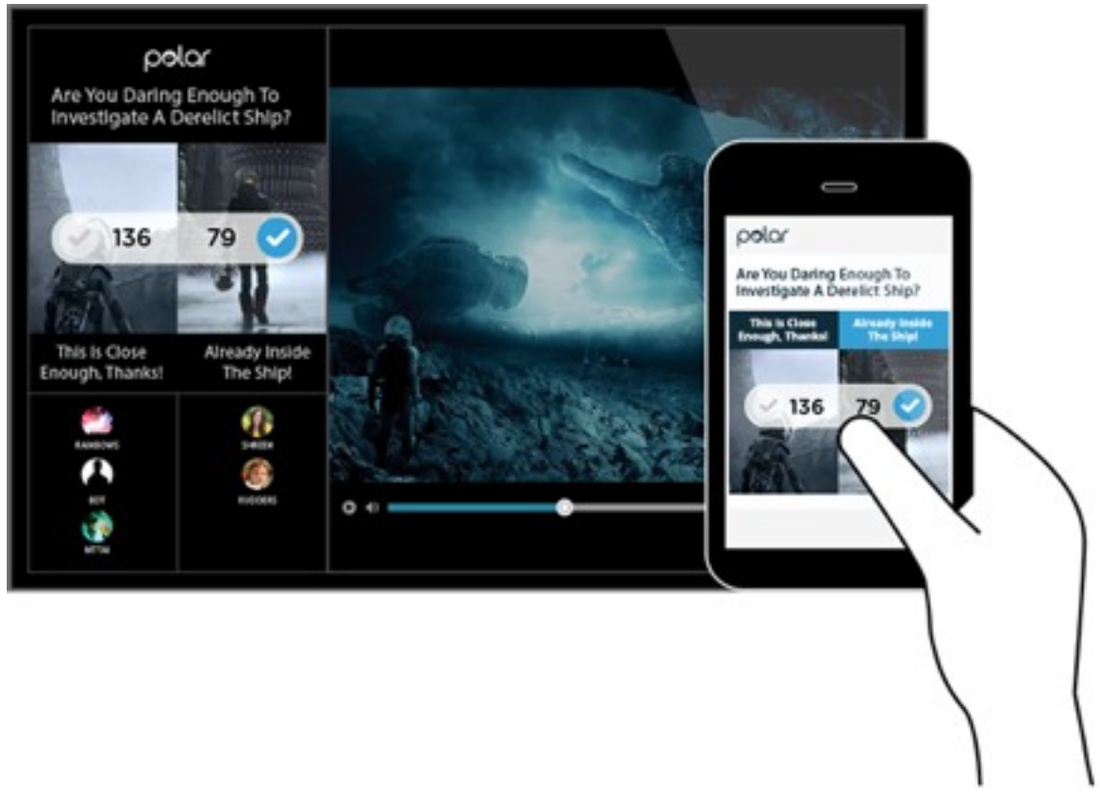
Polar allows you to easily share your opinion on any topic from iOS7 icon design to summer foods. We built a Web experience that not only looks good **on different sized screens** (smartphones, tablets, laptops, etc.) but also works great **with different kinds of input** (touch, mouse, keyboard) as well. So you can use Polar on the Web wherever and however you like.

But where and how people use the Web keeps changing and increasingly involves more than just one device. So our new Companion Web experience for Polar isn't just designed to be accessed on a wide range of devices, it's designed to work simultaneously across these devices as well.

As you can see in the video, the Polar Companion Web experience complements what you're watching on the big screen. You can leave it up alongside any film, TV show, or video you're viewing and keep up with new opinions and questions, as they appear in real-time.

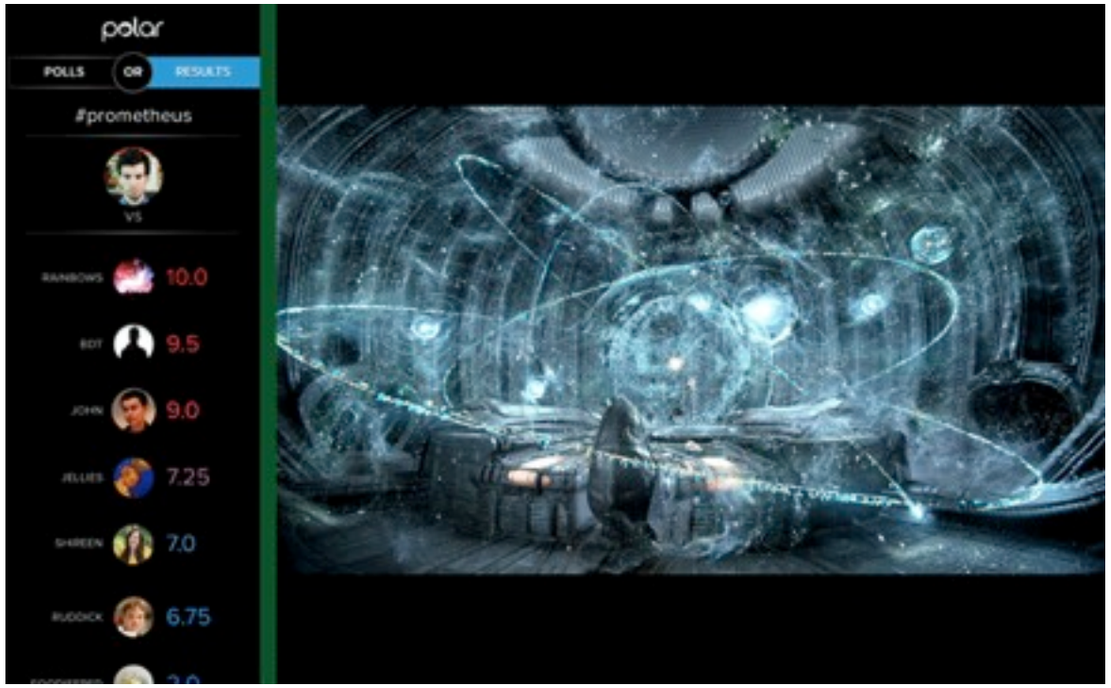


Or use any device with a Web browser to control the big screen display. When you scroll, vote, or switch topics with your connected device, the big screen experience responds to your actions: vote on a poll and see results from others instantly on the big screen; start scrolling on your connected device and the big screen automatically switches to list mode showing you totals for the polls you've already voted on.



We've even made it possible to have multiple devices connected to the screen at once so you can take turns controlling the big screen experience.

Once you are done voting on a topic, you can switch the big screen display over to the Results view to see how your opinions stack up against others who have voted on the same topics. This display updates in real time as others continue to vote so you can keep it alongside whatever you're viewing to follow along.



Because this is the Web, you're not limited to just one way of using our Companion Web experience. Turn on a video on any Windows 8 laptop or tablet and snap Polar's companion experience alongside for a single-device split screen view. Or connect any two Web browser windows you like whether they are on the same device, same operating system, or not.

As you can see, the Web is full of possibilities and Companion Web experiences like Polar allow you to connect the diverse screens in your life in new ways that put control in your hands. It's an exciting time to be building online and we're happy to have worked on it together with the Internet Explorer Team.

You can get more information on the Companion Web on **Microsoft's site.**

Responsive Web: Relying Too Much on Screen Size

As people continue to go online using an ever increasing diversity of devices, responsive Web design has helped teams build amazing sites and apps that adapt their designs to smartphones, desktops, and everything in between. But many of these solutions are relying too much on a single factor to make important design decisions: screen size.

What's Wrong With Screen Size?

It's not that adapting an interface to different screen sizes is a bad thing. Quite the opposite. It's so important that key metrics like conversion and engagement usually **increase substantially** when Web sites adjust themselves to fit comfortably within available screen space. For proof, just

look at how mobile conversion rates **increase significantly more** in responsive redesigns than PC conversions do.

So if adapting to different screen sizes can have that kind of positive impact for a business, what's the risk? As the kinds of devices people use to get online continue to diversify, relying on screen size alone paints an increasingly incomplete picture of how a Web experience could/should adapt to meet people's needs. Screen size can also lead to bad decision-making when used as a proxy for determining:

- If a browser is running on a mobile device or not
- If Network connections are good (fast) or bad (slow)
- If a device supports touch, call-making, or other capabilities

There's still no relationship between screen size and bandwidth. Instead, we should ensure our work's as light as possible *for everyone*.

– Responsive Design (@RWD)

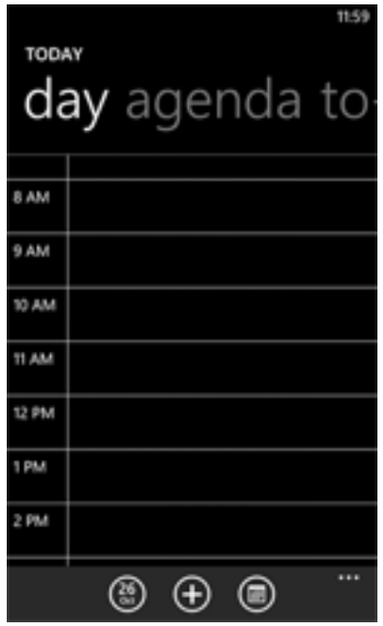
None of these can actually be accurately inferred from screen size alone but they are comfortable assumptions that make managing device diversity substantially easier. The harsh truth however, is that output (screen size and resolution) is only one third of the equation -at best. Equally important to determining how to adapt an interface are input capabilities and user posture, which sadly screen size doesn't tell us anything about.

Let me illustrate with a few specific examples.

Screen Size Limits

On tablets, PCS, and TVs, Microsoft's Windows 8 platform allows any app, including the Web browser, to be "snapped" to the side of a screen thereby letting people interact with it while using another application in the primary view. As an example, the Windows 8 calendar application can be snapped alongside the weather app when making your daily plans.

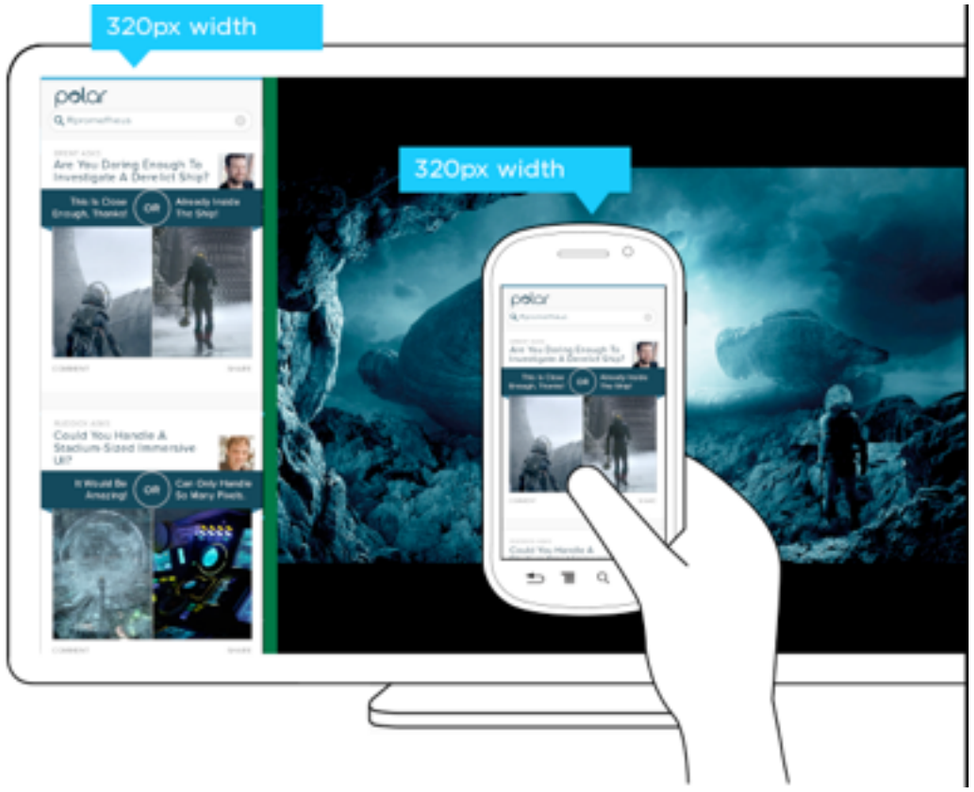
Notice though, that the default view of the calendar application on Windows Phone 8 is quite different than the snapped view of the same app on a tablet, PC, or TV. They are both using the same amount of screen width (in relative pixels), but the mobile interface starts with a daily agenda instead of a small month view by default. The controls are also adjusted to the mobile form factor as you can see in the image below.



We can debate about why these differences exist and if they should or not but the bottom line is there's more than screen size being taken into account in these application designs.

This simple example illustrates the challenge for Web designers. On Windows Phone devices, Internet Explorer uses 320 pixels for its device-width (the width it renders content at). On Windows 8 tablets, PCs, and TVs, snap mode uses the same 320 pixel device-width to lay out Web pages docked alongside other apps.

So with a responsive Web design, people get the same interface on a smartphone that they get in snap mode on a TV screen due to the same device-width (320 pixels). You can see this illustrated in the image below.



But should the interface be the same? A TV is usually viewed from about 10 feet away, while the average smartphone viewing distance is about 12 inches. This has an obvious impact on legibility for things like font and image sizes but it also affects other design elements like contrast. So a user's posture (in this case viewing distance) should be taken into account when designing for different devices.

The input capabilities of a TV (D-pad) can differ wildly from a those of a mobile device (touch) or in some cases be the same (voice). Designing a simple list interface for d-pads requires a different approach than a creating a similar listing for use with touch gestures. So available input types should also be considered in a multi-device design.

When you take user posture and input capabilities into account when designing, an interface can change in big or small ways. For instance, contrast the design below for Windows 8 snap mode on a TV compared to a mobile version of the same feature.

While the screen size (320 pixel device-width) has stayed consistent, the interface has not. Larger fonts, a simplified list view, inverted colors, and a lot more have changed in order to support a different user posture (10 ft away vs. 12 inches), and different input types (d-pad vs. touch). As you can see, screen size doesn't give us a complete picture of what we need to know to design **an appropriate interface.**

Before you dismiss this as an isolated use case on Windows 8 devices, note that Android smartphones and tablets also offer the ability to interact with **multiple applications side by side** and Android-powered TVs won't be far behind. In fact, we've already got Android eyepieces like Google Glass that pose similar challenges.

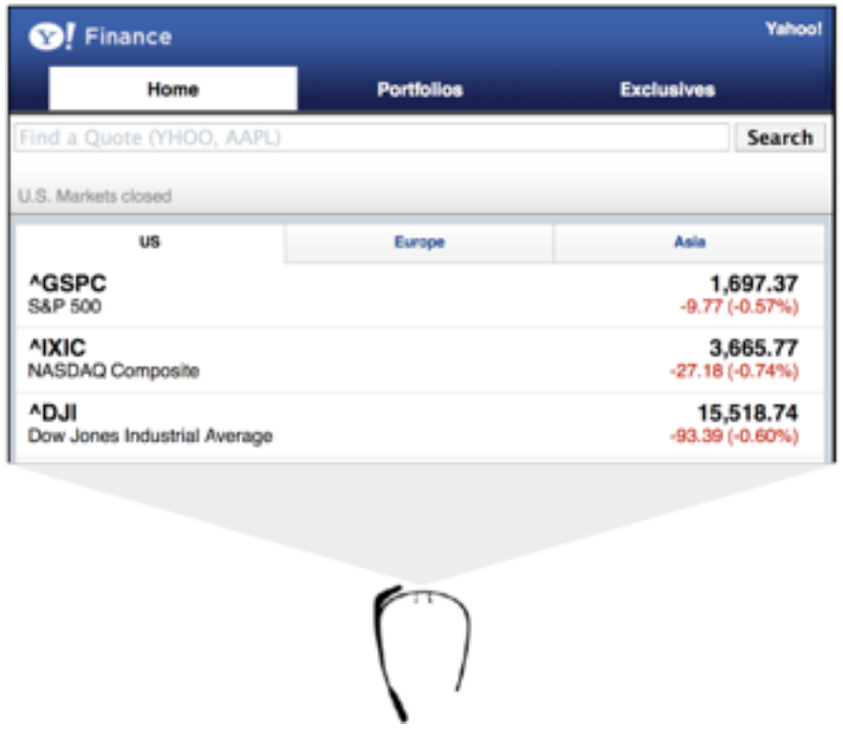
Google Glass allows you to view applications and Web pages using a display that projects information just above your line of sight. The official specs describe the Glass display as a "25 inch HD screen viewed from 8 feet away." So right up front, viewing distance matters.

Like most mobile Web browsers, Glass uses a dynamic viewport to resize Web pages for its screen. On Glass the default viewport size is set to 960 pixels and pages are scaled down accordingly. So if someone is viewing the Yahoo! Finance site, it displays like this in the Glass browser (below). Essentially, it is shrunk down to fit.

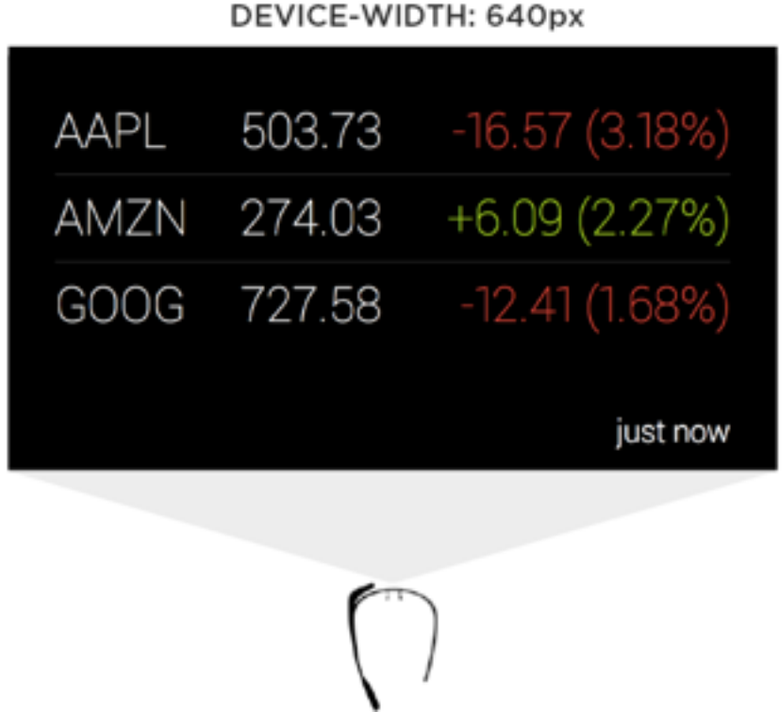
VIEWPORT: 960PX

The Web browser on Glass also allows pages built responsively to adapt to a more suitable device-width. In this case, 640 pixels. So a Web page designed to work across a wide range of screen sizes would render differently on Glass. Given that 600 pixels is a common device-width for 7 inch tablets, the page you'd see on Glass would look more like the following -adapted for a smaller viewport size.

DEVICE-WIDTH: 640px



In addition to the Web browser, Google Glass also includes a number of “glassware” applications built with the same Web technologies used to create Web pages. One of these apps provides access to stock price changes -very similar to what you see displayed prominently on the Yahoo! Finance site. However, the presentation of this information is very different. As you can see in the image below it’s been designed as if you are viewing a 25” screen from 8 feet away. This design is much more suited to a wall-sized display than a small tablet screen.



This Glassware interface is also designed to make scrolling through information using the touchpad on the side of Google Glass (which comfortably supports sweeping left/right and up/down gestures) fast and easy.

So again user posture and input capabilities inform how to design for a specific device. Screen size alone doesn’t tell us enough.

Supporting Everything

In order for an interface to adapt appropriately to different output, input, and user posture, we need to know what combination of the three we’re are dealing with at any given time. On the Web that’s been notoriously difficult. We **can’t tell TVs from smartphones** or **what devices support touch** without relying on some level of user agent detection, which is often looked at dubiously.

Because of this, Web developers and designers have smartly decided to simply **embrace all forms of input**: touch, mouse, and keyboard for starters. While this approach certainly acknowledges the uncertainty of the Web, I wonder how sustainable it is when voice, 3D gestures, biometrics, device motion, and more are

factored in. Can we really support all available input types in a single Web interface?

A similar approach to user posture is increasingly common. That is, an interface can simply ask people if they want a lean-back 10 foot experience, a data dense 2 foot experience, or something more suited for small portable screens. This makes user posture something that is declared by people rather than inferred by device. Once again, this kind of "support everything" thinking embraces the diversity of the Web whole-heartedly. However it puts the burden on each and every user to understand different modes, when they are appropriate, and change things accordingly. (Personally I feel we should be able to provide an optimal experience without requiring people to work for it.)

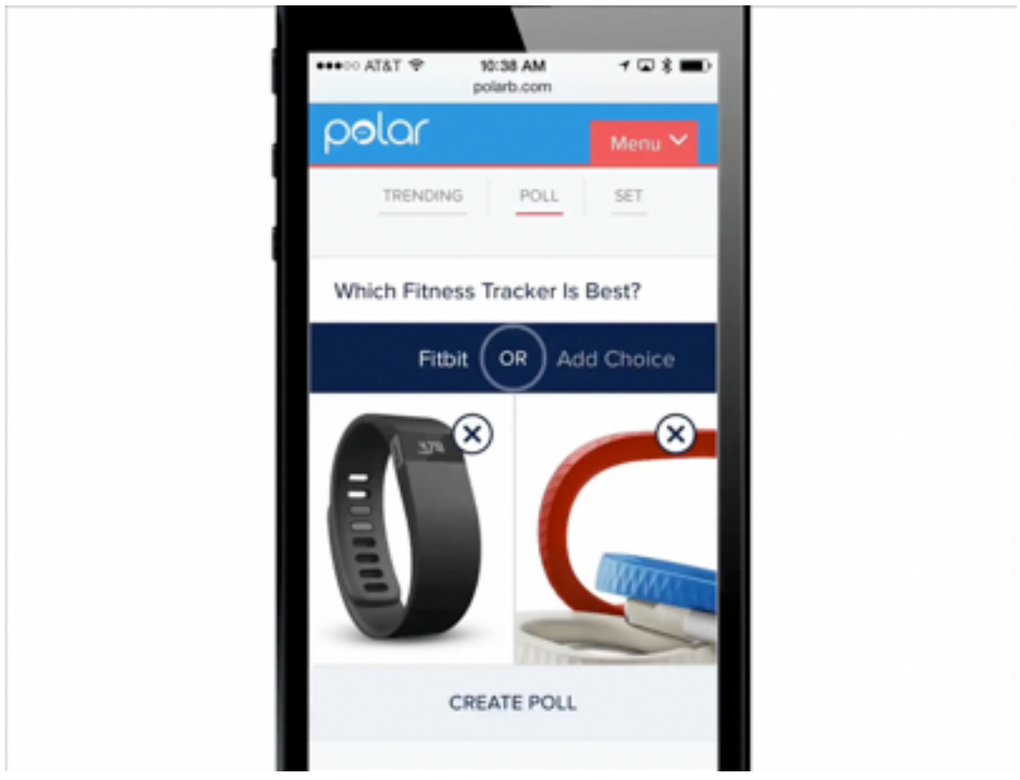
Ultimately trying to cover all input types and all user postures in a single interface is a daunting challenge. It's hard enough to cover all the screen sizes and resolutions out there. Couple that with the fact that an interface that tries to be all things to all devices might ultimately not do a good job for any situation. So while I embrace supporting the diversity of the Web as much as possible, I worry there's a limit to the practicality of this approach long-term as the amount of possible inputs, outputs, and user postures continues to grow.

Don't Assume Too Much

These examples are intended to convey one important point: don't assume screen adaptation is a complete answer for multi-device Web design. Responsive Web design has given us a powerful toolset for managing a critical part of the multi-device world. But assuming too much based on screen size can ultimately paint you into a corner.

It's not that adapting to screen size doesn't matter, as I pointed out numerous times, it really does. But if you put too much stock in screen size or don't consider other factors, you may end up with incomplete or frankly inappropriate solutions. How people interact with the Web across screens continues to evolve rapidly and our multi-device design methods need to be robust enough to evolve alongside.

Customizing Help & Tips By Input Type



On today's **multi-device Web,** your audience might be using a mouse, keyboard, touchscreen, trackpad, or increasingly, more than one of these input types to interact with your service. Given all these possibilities, how do you let people know how they can get things done in your app?

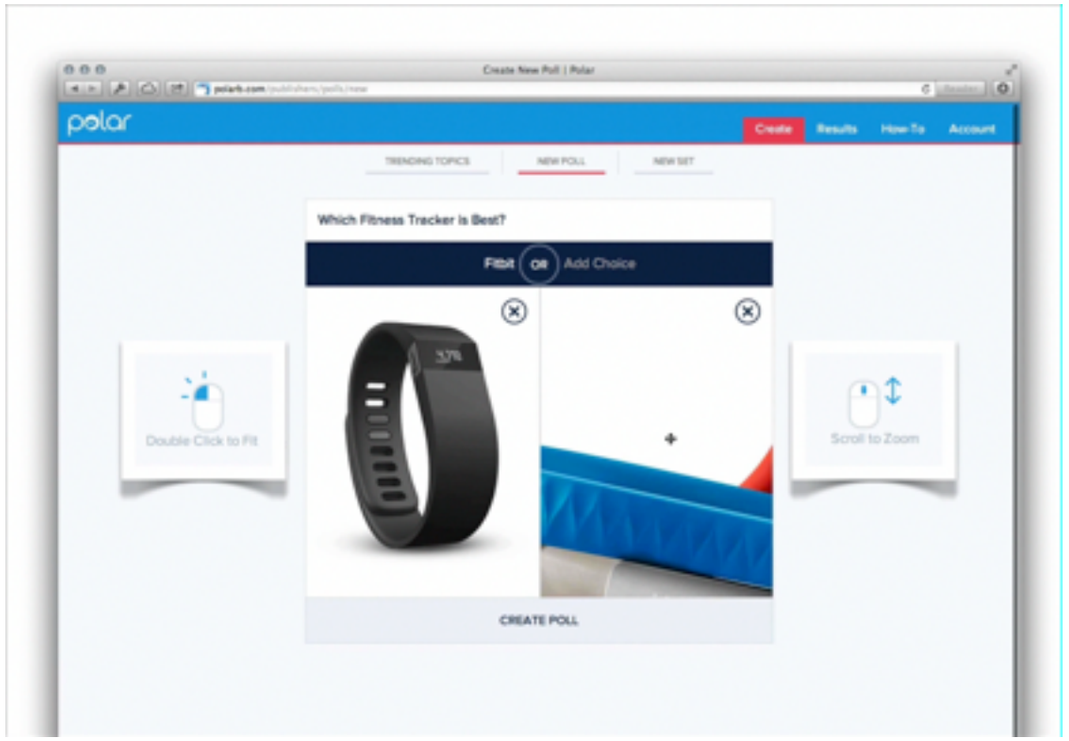
A common way to provide relevant bits of guidance inside an application is through **inline help.** Inline help is positioned where it's most useful in an interface and made visible by default so people don't have to do anything to reveal it. This makes it an effective way to tell people how to use an interface. But what happens when those instructions vary by input type?

For instance, we recently built a fully responsive Web application that can be used on smartphones, tablets, desktops, and more. In this application, people can add and manipulate images by zooming, resizing, and moving these images around. Depending on the input type their device supports, however, they have different ways of accomplishing these tasks.

To position an image using touch, you simply drag it with your finger. Zooming-in and zooming-out is accomplished through pinch and spread gestures. Sizing an image to fit happens through a double-tap action.

These same features are supported on mouse and keyboard devices as well. To move an image around, click and hold to drag it. Zooming in and out happens with the scroll wheel or a multi-touch gesture on the trackpad. Sizing an image to fit takes a double-click of the mouse.

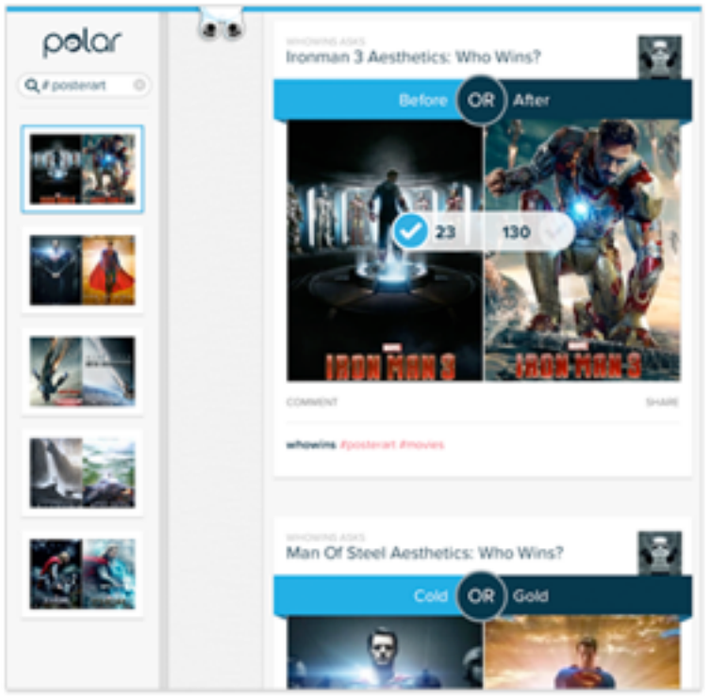
As you can see, the touch and mouse actions are similar but not the same. We were also concerned that while touch users are quick to pinch and spread when trying to zoom an image, mouse users are less familiar with using scroll wheels and two-finger trackpad gestures to accomplish the same thing. So we wanted to let our mouse users know what's possible with a few simple bits of inline help.

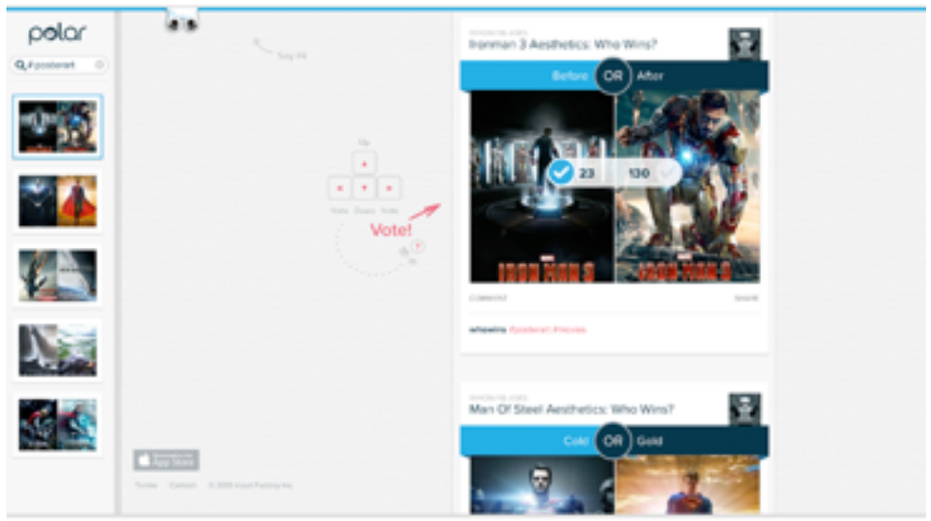


Easy, right? Just **check if** someone's device has a mouse or trackpad attached and reveal the tip. Well, **no**.

On **an earlier project**, we faced a similar situation. While our smartphone and tablet users could easily

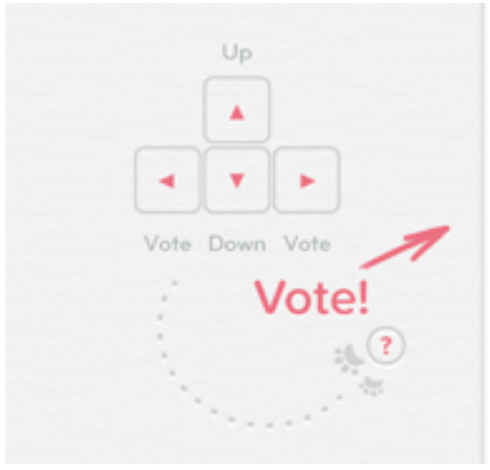
interact with our lists of polls by scrolling and voting with their thumbs, mouse and keyboard users had more work cut out for them. They couldn't simply keep their fingers in one place and vote, scroll, vote.





So we developed a keyboard interaction that allowed people to vote with left/right keys and move through polls with up/down keys. While this made keyboard users much faster and effective, we again were faced with a hidden interface: people didn't know keyboard voting was possible unless we told them.

That meant we had to detect if a keyboard was present and surface a simple inline tip that explained the voting interface. After a few failed attempts at doing this, I reached out to friends on the Internet Explorer team at Microsoft. After all, if anyone would have a good handle on how to manage different input types on the Web, it would be the company with a Web browser that not only runs on phones, tablets, laptops, and desktops but hybrid devices and even game consoles as well.



Sadly, we learned there's not a great way to do this in Web browsers today and browser makers are hesitant to reveal information like this in a navigator.hardware object because of concerns it could be used to fingerprint users. So instead, we opted to proxy the presence of a keyboard using screen width. That is, if the screen is wide we assume there's a higher chance of a keyboard being present and show the keyboard interface tip by default.

This is the same solution we now have in our new publisher tool. When the screen crosses a certain width, we reveal two inline tips explaining how to zoom and fit an image using a mouse.

As I've written before, using **screen width as a proxy** for available input types is not ideal and increasingly unreliable in today's constantly changing device landscape. But the alternative solutions don't seem much better.

For instance, we could provide a help section that explains how to do things with every input type. But then we loose the immediacy and effectiveness of concise inline help. We could wait until someone interacts with the app to determine if they are a touch or mouse user, save that information and display device-appropriate tips from that point on. But even if we get this info up front it can change as people switch between touch screens and trackpads or their mouse.

While the need for input-appropriate help text in a Web application may seem like a small detail, it's reflective of the broader challenge of creating interfaces that not only work across different screen sizes **but across many different capabilities** (like input type) as well. Inline help is just one of many components that need be rethought for today's multi-device Web.

Thank You

The Polar Team: Jeff Cole, Winfield Peterson, Thanh Tran, and Paula Wirth.

Special Thanks to: Dmitry Dragilev, Ashley Streb, Chris Butler, Jason Weaver, and everyone who used Polar. You helped make it so much better than we ever could on our own.