

# iOS Accessibility: Doing Well by Doing Good

| John C. Fox (@djembe)  
Senior Software Engineer, iOS UI

NETFLIX

My name is John Fox, I'm a member of the iOS UI team at Netflix, and I'm very happy to be here in Sunny St. Petersburg. I'm going to talk to you about Accessibility for iOS. I was very purposeful in my title "Doing Well by Doing Good". There are many moral reasons for caring about accessibility: that's the the "doing good" part. There are also many benefits to caring about it, for design, engineering and customer happiness reasons: that's the doing well part.



In January 2016, we launched Netflix globally (minus China & Syria, because of reasons beyond our control). This means we have to care about how to make our app feel comfortable to people with all sorts of different cultural backgrounds, who speak different languages, use all sorts of different devices in different network conditions, etc.





Fundamentally, this means making our service work well for everyone, which means we have to care about more than just languages, network conditions, devices. We have to care about Accessibility as well.



**What is Accessibility?**

**Why should you care?**

**Implementation on iOS**

In my talk, I'm going to cover three main topics: what is accessibility? Why should you care? How do you implement it in your iOS app.

# What is Accessibility?

At a high level, Accessibility refers to the ability of a person to get along in the world, despite physical or mental challenges. There are many challenges which might come to mind. For example, a person who is blind, or deaf, or cannot walk and must use a wheel chair.

# Accessibility: a definition

- The ability for a person to get along in the world despite any physical or mental challenges, be they permanent or temporary.

Here's a useful definition. I want you to think about the notion of temporary disabilities, it will be important later on.

# Facing challenges



Imagine you don't have use of your legs and had to move about the world in a wheelchair? Just crossing the street used to be an incredible pain in the ass!

Imagine if you couldn't go to school because there was no way for you to get there, or get into the class room.

How would lack of education affect your choices in life?

# Ed Roberts



Ed Roberts was born in Burlingame, CA which is near San Francisco. He contracted Polio at age 14. As a result, he spent most of his day in an Iron Lung which helped him to breathe. He eventually developed the strength to breathe on his own, but couldn't walk.

He was a bright student, and was accepted to UC Berkeley, but when he showed up, they said he was too profoundly disabled to be there.

How would you feel? Would you be angry? I certainly would.

# Many others



There were many many others. Disability rights were likened to Civil Rights.

I love this sign. One way of thinking about it is that if you deny people the right to achieve all that we can, it damages our society as a whole, both morally and practically.

We have plenty of challenges, and we need all the brainpower we can get!

## 1990: ADA was signed

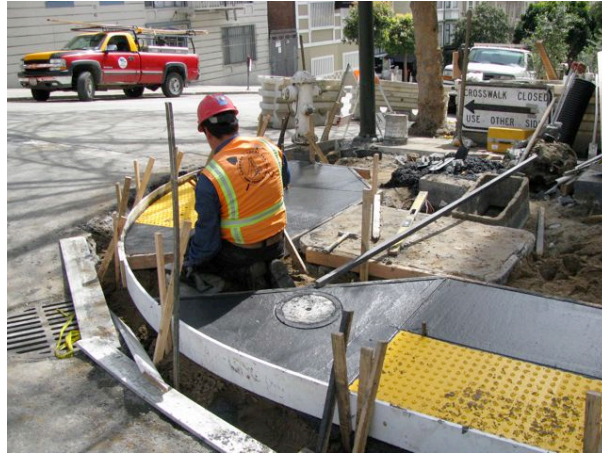


It took a long time, and a lot of struggle, but in 1990, George Bush, Sr. signed the ADA, the Americans with Disabilities Act. When he signed the act, it was not long after the fall of the Berlin Wall. He said that it the purpose of the law was to remove walls that served to block people from living their lives.

It essentially mandated that our physical spaces be made accessible using reasonable accommodations.



# Curb cuts: required by law



In cities of all sizes around the world, it's now common to find wheelchair ramps, sometimes called "curb cuts".

For a long time, people complained about the expense of tearing up streets, but too bad: it was the law.

Sometimes, important changes need to be mandated.

# Helpful to all of us



Eventually, all kinds of people, without any conditions that would be called a disability found them useful.

Perfect example, is a parent who has to push a child around, oftentimes, only having one hand free.

This is an example of a situational disability. There are lots of people in wheelchairs, how many parents of small children are there in the world?

I can assure that as a parent of a teenage son, I sometime consider his existence the source of situational disabilities such as sleep deprivation.

## Even fun, for some



But even beyond serving needs, curb cuts can provide amusement to your local grown men who act like teenagers.

I hope you appreciate the effort I had to make to get a random skater dude to appear in a video for my presentation.



# **Accessibility is really a design problem**

So, Accessibility, is really a design problem. When you care for people with challenges, you end up caring for everyone.

I have a great example for the world of consumer products...

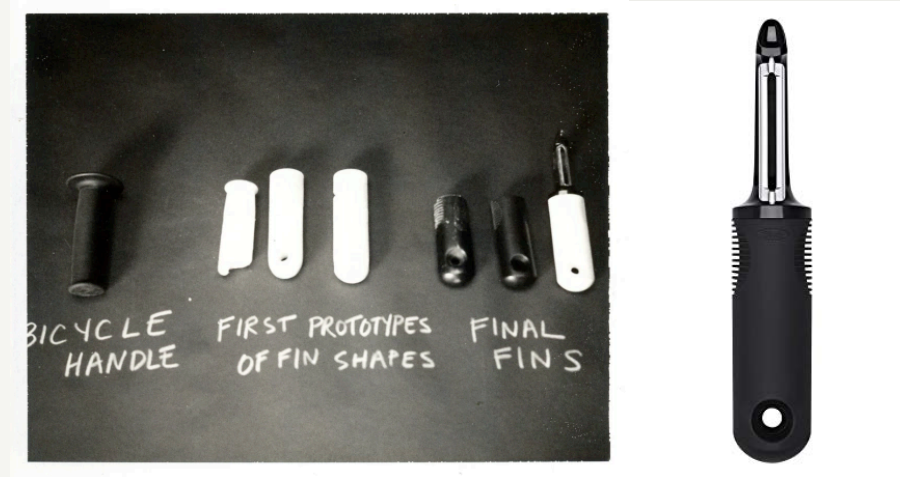
# Get a grip



This is a potato peeler. Have you ever used one?

There's a fellow named Sam Farber, whose wife suffered terribly from arthritis. She had a hard time gripping peeler with their small, hard metal grip.

# Designing a Grip



He tried many many different prototypes, and eventually landed with the design on the right.

# A grip on the market



OXO is a company that makes consumer tools that are much loved for their super comfortable grip. Their original product was a new version of a potato peeler.

There's no an entire line of kitchen tools which are super comfortable and are huge sellers.



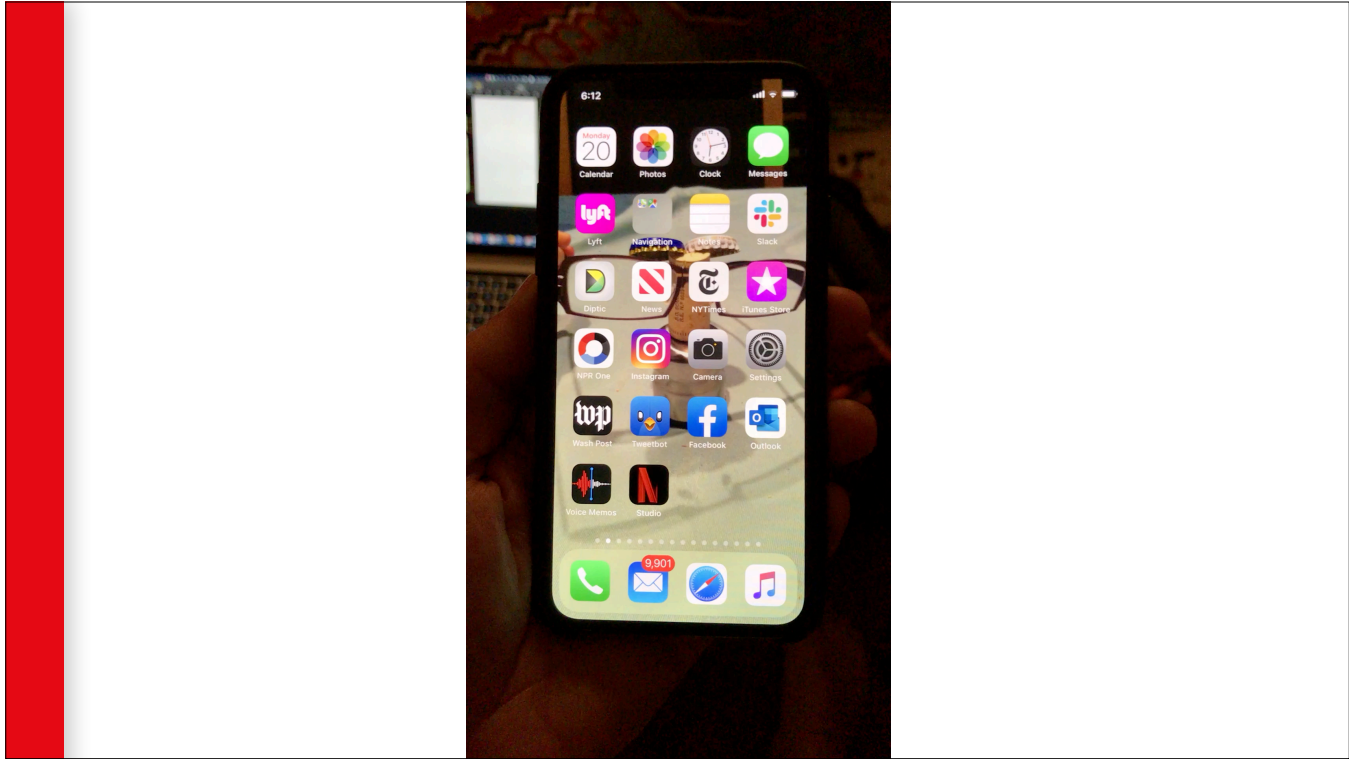
# What is *Computer* Accessibility?

So, we've talked about accessibility for the physical world. Remember when I talked about access to education.

Since the advent of computers connected to the Internet, can imaging what life would be like if you couldn't use them.

Would any of us be here right now?





Let's ask Siri



Fortunately, Siri doesn't understand my own style of humor.

# Accessibility for iOS

In that example, you saw the use of Siri, which is a Voice Interface. When it works, it's awesome. Siri's technology, with its synthesized voice is intertwined with the greater story of Accessibility for iOS.

# UIAccessibility:

- Traits: what the element is and how it behaves (button, text, image, selection state)
- Label: a concise, localized description of the element
- Hint: an optional bit of text which gives an idea about what will happen next

Accessibility on iOS is accomplished through the UIAccessibilityProtocol.

There are several methods to the protocol, but for the purposes of this talk, we're going to focus on just three.



Step 0: a quick and dirty guide for setting up Accessibility Shortcuts on your device.

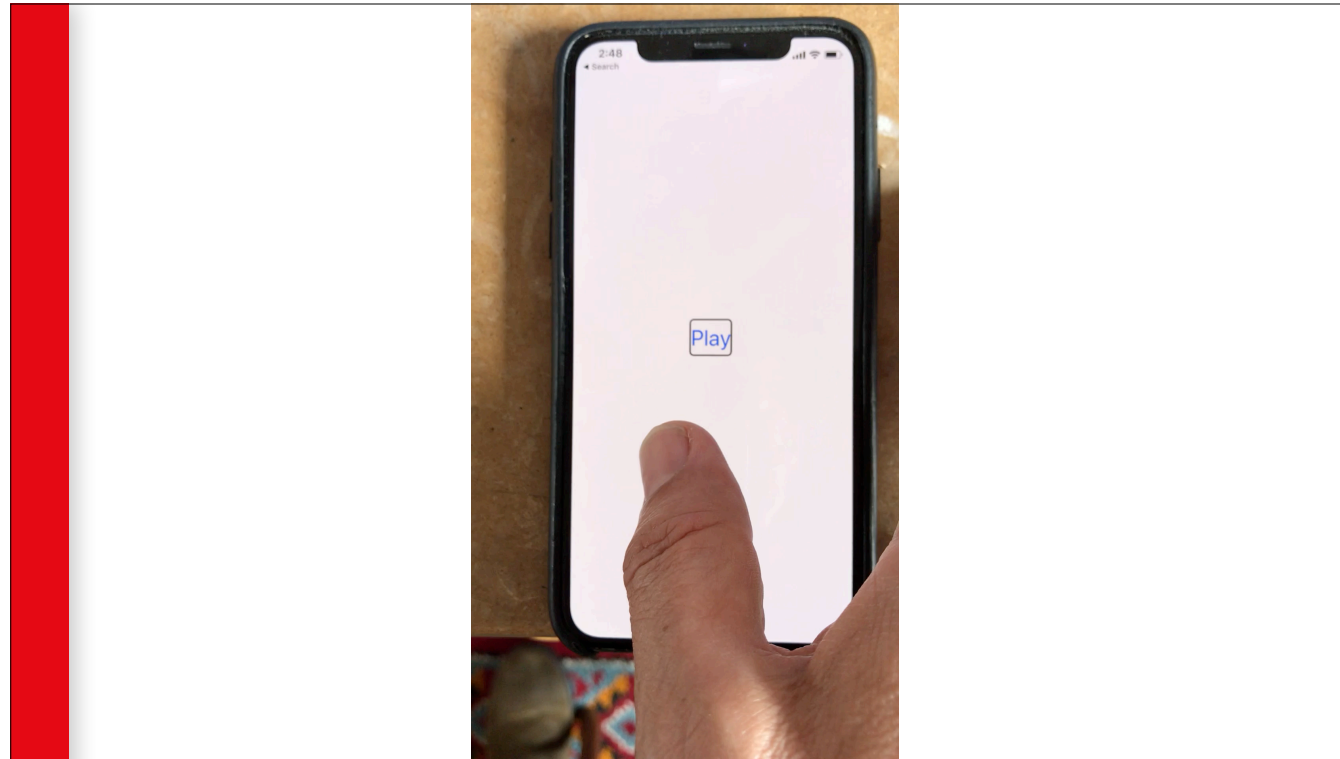


# Hello (Accessible) World

We're going to look at the simplest example of UIAccessibility. No work is required by the programmer!

# Accessibility: UIButton

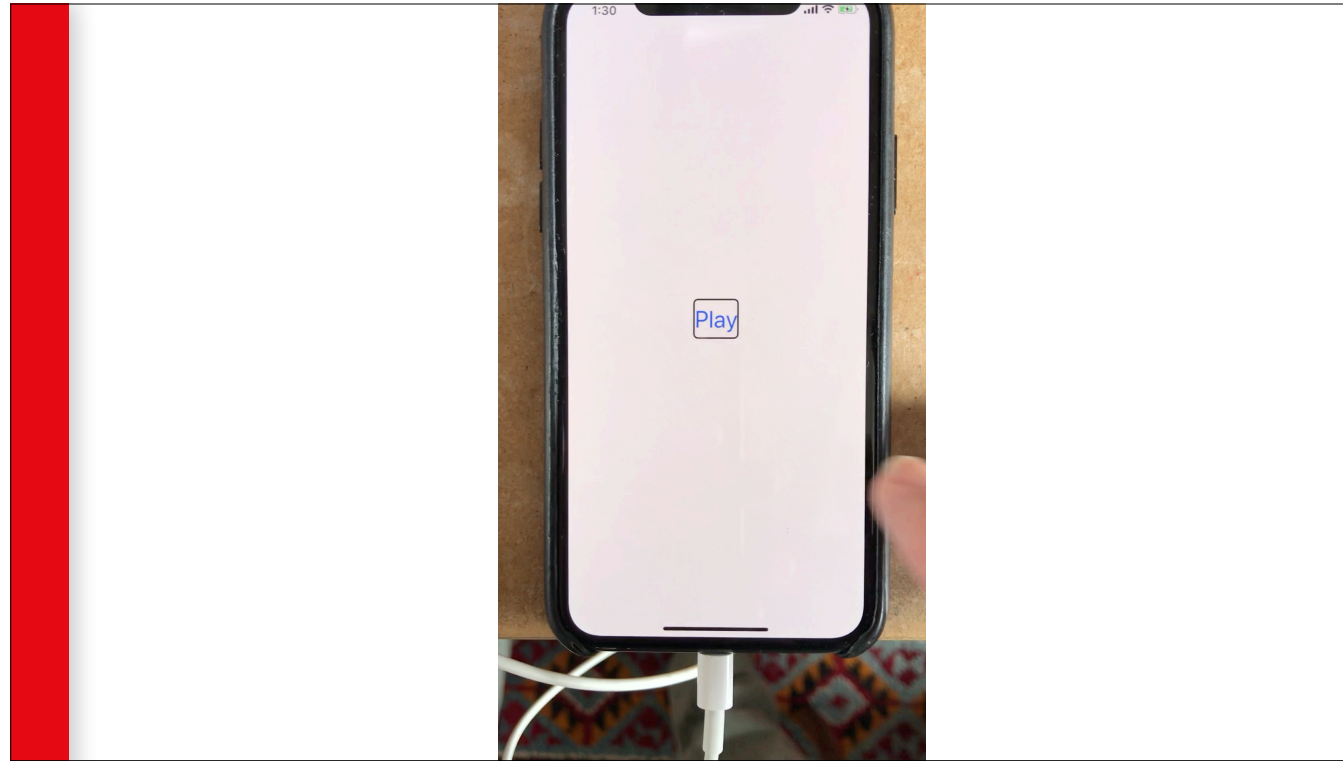
Where would we be without UIButton? We'd just stare at the screen and nothing would happen. If you can't see, how would you where to tap on the screen and what the elements are?



When VoiceOver is running, you can navigate through all the elements on the screen by swiping left or right, or by just moving your finger around.

In this example, there is only one element on the screen. VoiceOver will announce the accessibilityTrait for each element, which for a UIButton, is “Button”. This lets the user know that there’s some action that can be invoked using the single-finger double-tap...





Notice that you can double tap anywhere on the screen to invoke the action of the selected button. After the button's action is invoked its new title is read.

# The code:

```
- (IBAction)togglePlayback:(UIButton *)sender {  
  
    // Next state is the opposite of current  
    NSString *newTitle = self.isPlaying ? @"Play" : @"Pause";  
  
    // UIButton's default accessibilityLabel is its titleLabel's text  
    [sender setTitle:newTitle forState:UIControlStateNormal];  
  
    // Update state  
    self.isPlaying = !self.isPlaying;  
}  
  
@end
```

UIButton's default implementation of accessibilityLabel just returns the text of the titleLabel. This method is no different than what might otherwise be done. You get your accessibility for free.

# UIButton with Icons



Text labels are pretty dull, so we more typically use icons.

# The code:

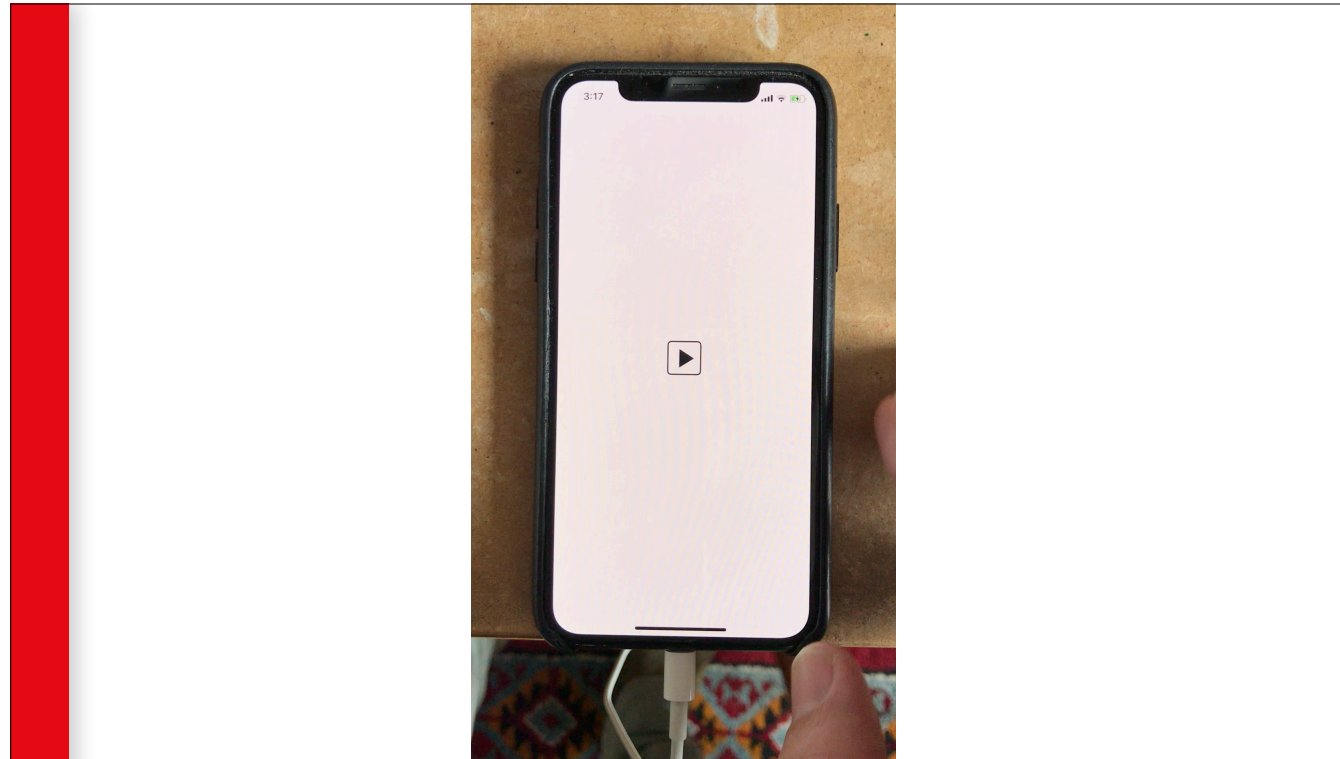
```
- (IBAction)togglePlayback:(UIButton *)sender {  
  
    // Next state is the opposite of current  
    NSString *newIconName= self.isPlaying ? @"playIcon" : @"pauseIcon";  
  
    // Update the icon  
    [sender setImage:[UIImage imageNamed:newIconName]  
                forState:UIControlStateNormal];  
  
    // Update state  
    self.isPlaying = !self.isPlaying;  
}
```

You might do something like this in your code: programmatically swap icons or perform animations.



# Does it play well?

Let's see how that works for VoiceOver.



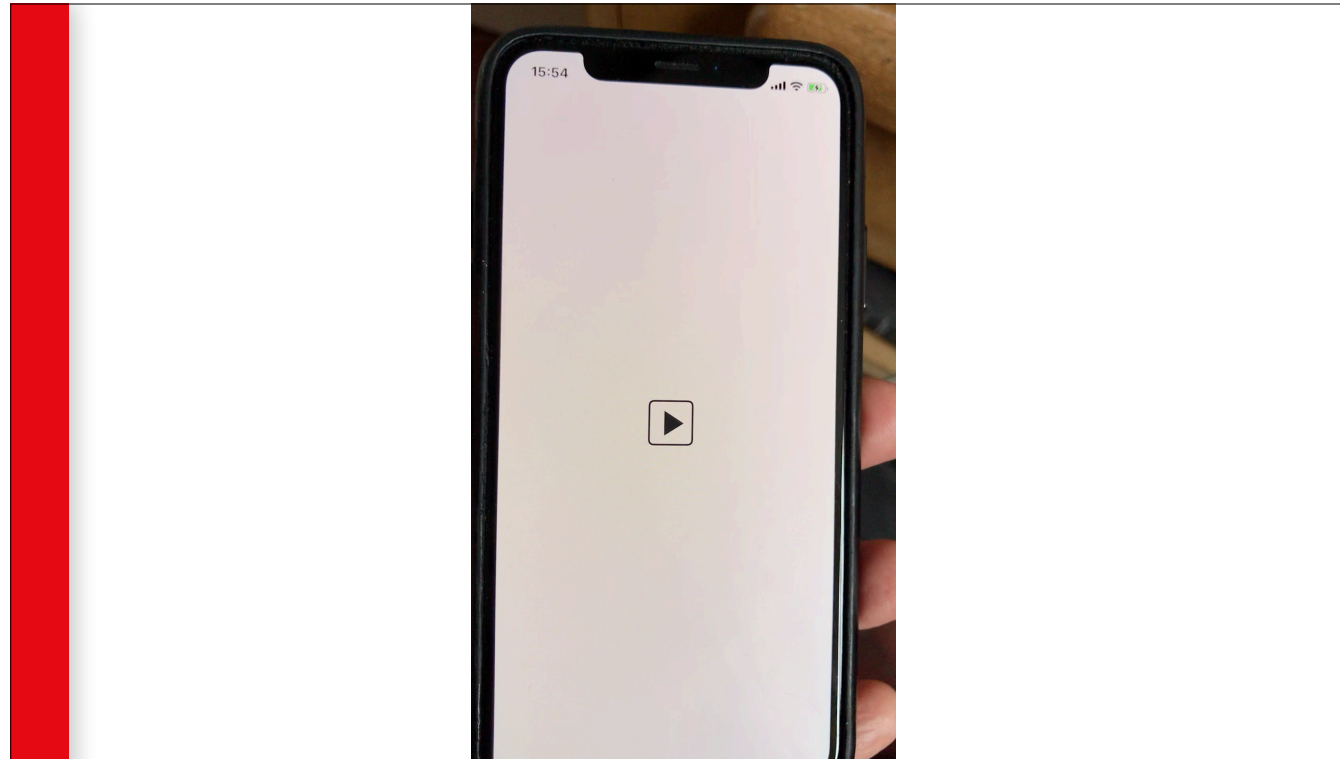
Let's see how VoiceOver handles this...

Not very well. It's left to read the name of the image, which in this case kinda, sorta works because the file name resembles the function, but only in English.



**А по русски?**

And in Russian?



Not so great: it's reading the word "playlcon" and "pauselcon" with a Russian accent.

I found it so fun, I was tempted to break out Garage Band and crank out a new hit Dance Remix for my band Accessibility Playground.





**We can do better!**

Surely we can do better.

# The code:

```
- (IBAction)togglePlayback:(UIButton *)sender {  
  
    // Next state is the opposite of current  
    NSString *newIconName= self.isPlaying ? @"playIcon" : @"pauseIcon";  
  
    // Update the icon  
    [sender setImage:[UIImage imageNamed:newIconName]  
                forState:UIControlStateNormal];  
  
    // We localize, because we have love in our hearts  
    NSString *newLabel= self.isPlaying ? NSLocalizedString(@"a11yPlayLabel", "Play") :  
                        NSLocalizedString(@"a11yPauseLabel", "Pause");  
  
    // VoiceOver will read this  
    sender.accessibilityLabel = newLabel;  
  
    // Update state  
    self.isPlaying = !self.isPlaying;  
}
```

If you set the accessibilityLabel on UIButton, that is what will be read out.

# Localizable.strings:

```
/*  
    Localizable.strings  
    Accessibility Playground  
  
    Created by John C Fox on 5/19/19.  
    Copyright © 2019 Netflix, Inc. All rights reserved.  
*/  
  
/* VoiceOver label for play state */  
"a11yPlayLabel" = "Играть";  
  
/* VoiceOver label for pause state */  
"a11yPauseLabel" = "пауза";
```

At work, we use the prefix “a11y” to denote that this key is for Accessibility purposes.

The number “11” is a way to shorten the word because there are 11 characters between the “a” and “y” in Accessibility. I like it as well, because it looks “ally”: we want to be a good ally to all the communities we serve.



This is what we now get in English.

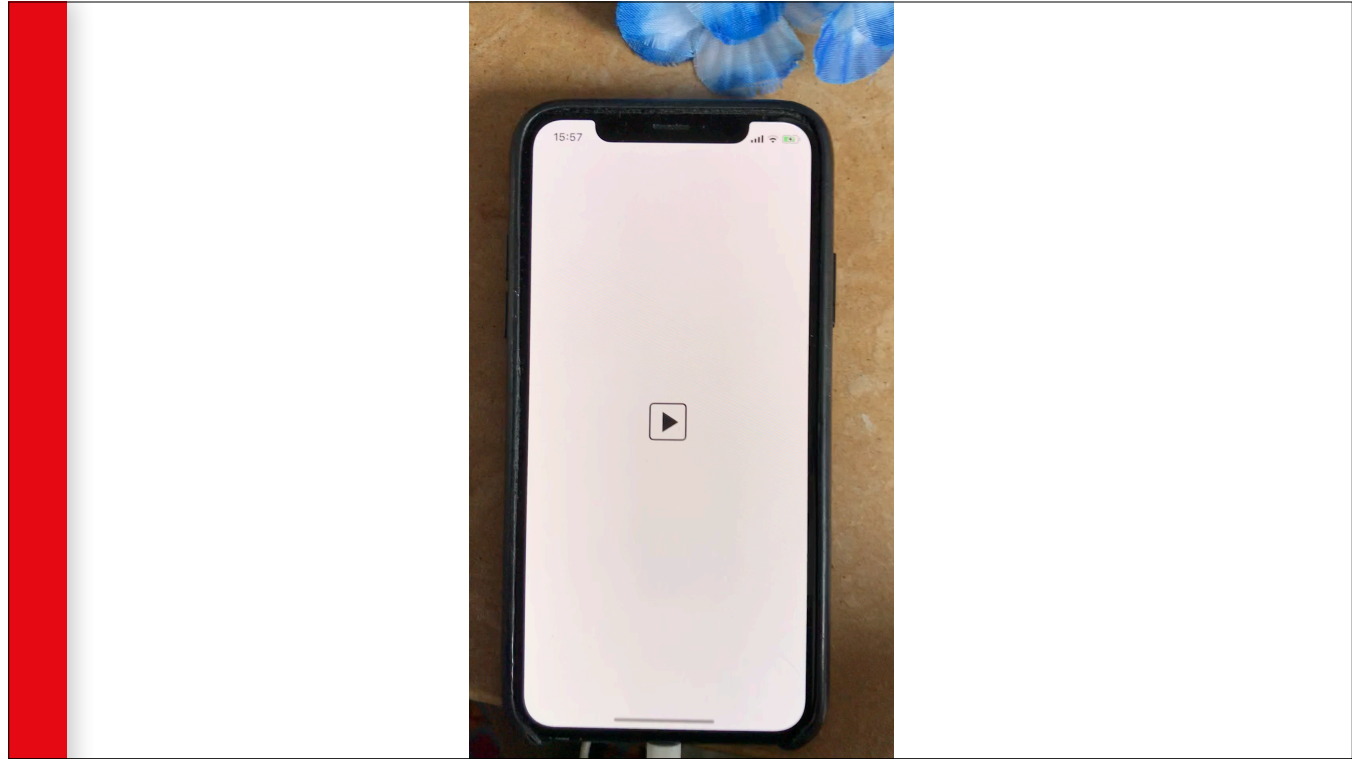


**Я люблю русский!**

Who here can read Russian?

Read it for us.

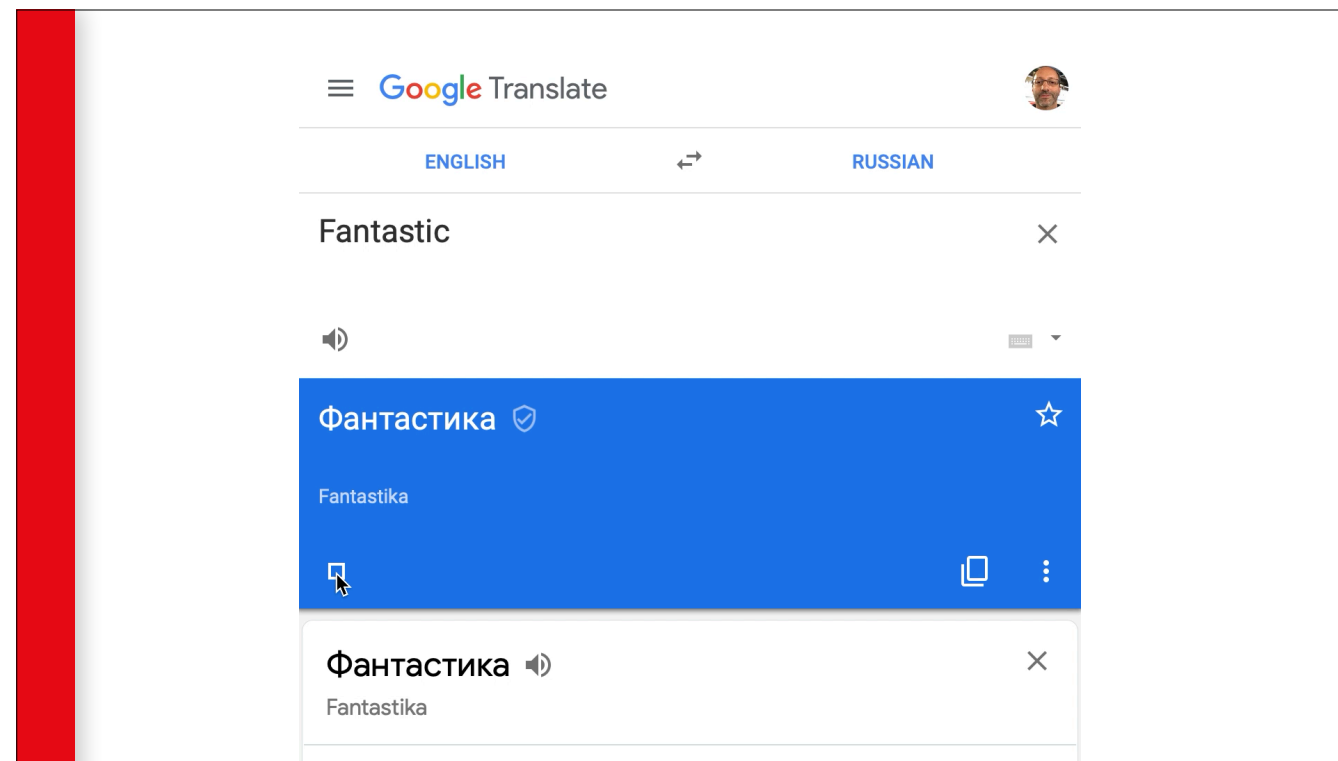
In English, this means I love Russian



Works great in Russian as well.



**How was that?**



I spared no expense in hiring the best translators on the Internet.

Remember this word, we'll be getting back to it later.

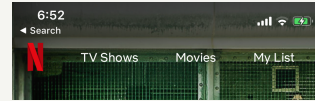


# What about custom UI?

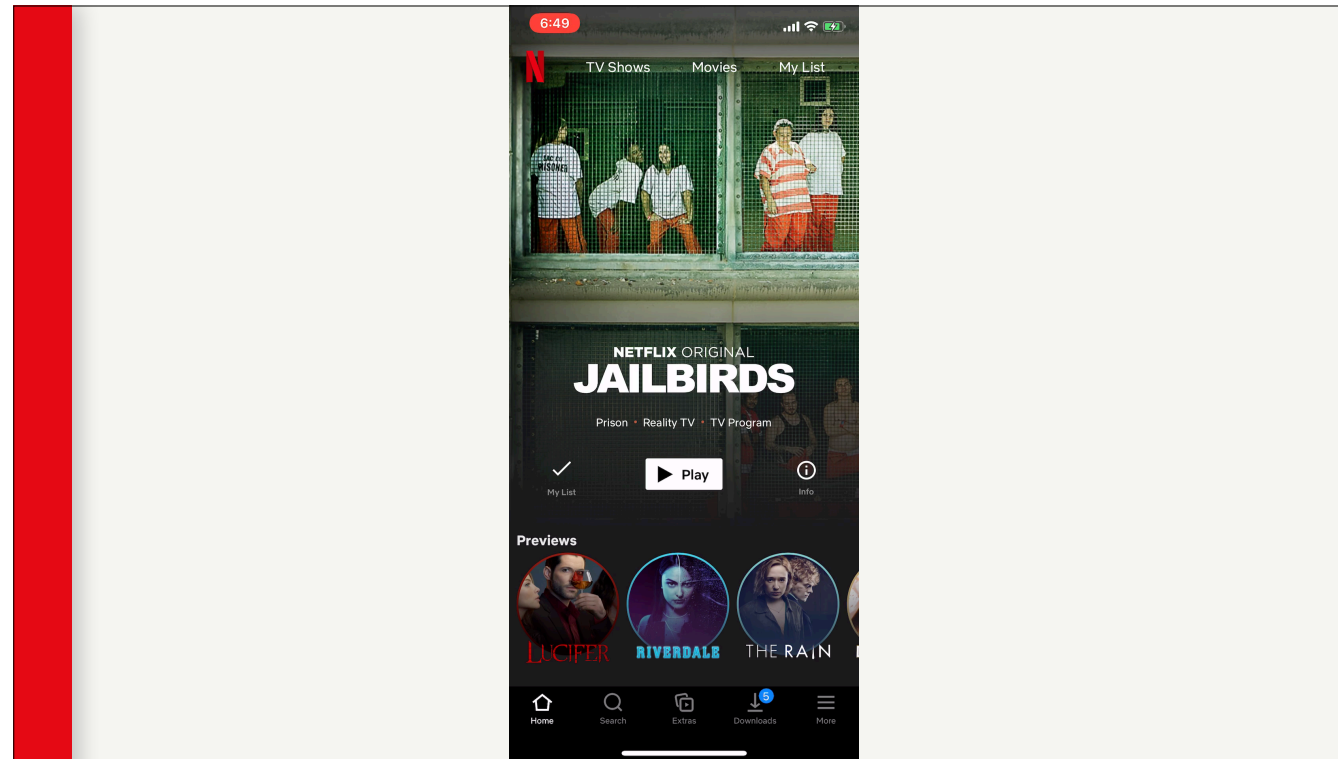
- You're responsible for setting traits, hints and labels
- It's very easy.
- So, make like a Nike ad, and Just do it!

Only the most basic apps use completely stock UI Elements. What happens when you create your own?

# Real-World Example



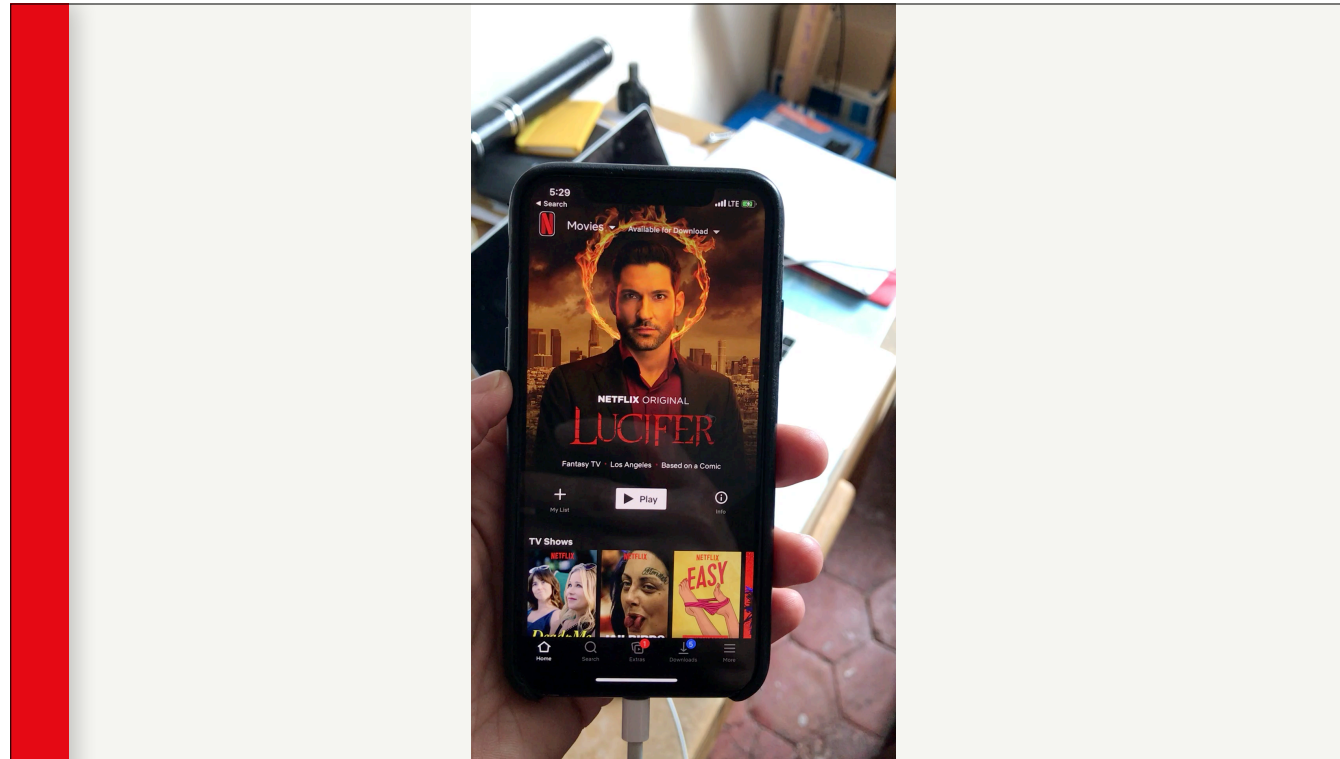
Not that long ago, we added a way to filter the main screen by type (movie/TV) and genre. We're going to watch a screen movie, but I want to you to focus on these UI elements at the top. This is a custom component.



(After video) Notice the subtle animation as the menu items are selected. Once the animation is complete, data is fetched from the server, and the underlying collection view is recalculated.

This is done with a custom UIView subclass which contains a UIStackView of button-like elements (the label and a downward pointing chevron) and a logo view for the Netflix “N”.

We had to do some accessibility work ourselves



Making this accessible was a pretty big challenge. We have to communicate to the user that there is a filter system, that it can be changed, and how to change it.

Let's watch this screen movie, then we'll step through the code needed.

# Code for hints and traits:

```
#pragma mark - Accessibility

- (NSString *)accessibilityHint {

    NSString *myHint = [NSString stringWithFormat:NFLXLocalizedString(@"a11yFilterActionFormat", @"Filter
    by TV Shows"), self.menuItem.name];

    if (!_shouldShowChevron) {
        myHint = NFLXLocalizedString(@"a11yFilterActionHint", @"Select a new filter.");
    }

    return myHint;
}

- (NSString *)accessibilityLabel {
    return self.menuItem.name.length ? self.menuItem.name : @"";
}

- (UIAccessibilityTraits)accessibilityTraits {
    return _shouldShowChevron ? UIAccessibilityTraitButton|UIAccessibilityTraitSelected :
    UIAccessibilityTraitButton;
}
```

Here's an example of implementing UIAccessibility methods in a custom UIView subclass

- \*accessibilityHint helps a user understand what will happen if they take the action
- \*Sometimes a label can be an empty string: a user can quickly skip
- \*When you have a list of items, and one can be selected, you can add the designation

# Coding for change:

```
// If we're animating a change, we need to ignore taps
- (void)setInteractionsEnabled:(BOOL)interactionsEnabled {

    // Dedupe
    if(!_interactionsEnabled == interactionsEnabled) {
        return;
    }

    _interactionsEnabled = interactionsEnabled;

    // Care about VoiceOver users
    [self updateAccessibilityLabels];

    UIAccessibilityPostNotification(UIAccessibilityLayoutChangedNotification, self);
}

- (void)updateAccessibilityLabels {
    // The "N" logoView should either be a button or nothing depending on filter state
    _logoView.accessibilityLabel = [self isUnfiltered] ? @" : NFLXLocalizedString(@"a11yFilterShowAllLabel",
        @"Remove all filters");
    _logoView.accessibilityTraits = [self isUnfiltered] ? UIAccessibilityTraitNone : UIAccessibilityTraitButton;
}
```

When the menu system is animating, it needs to ignore touches.

The “N” logo is a UIImageView which behaves like a button only when the collection view is filtered.

UIAccessibilityPostNotification let’s you tell the system that a layout change has taken place, so that VoiceOver can read again.

# Notifications:

```
// We care when a Lolomo has finished loading
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(handleLolomoResultsNotification:)
                                         name:@"NFUILolomoSuccessfulLoadNotification"
                                         object:nil];

// When a Lolomo filtering is complete, we can react for VoiceOver users
- (void)handleLolomoResultsNotification:(NSNotification *)aNotification {

    dispatch_async(dispatch_get_main_queue(), ^{

        // Tell the user what they're looking at
        [self a11yAnnounceCurrentStatus];

    });
}
```

For potentially long-running async process that are loosely coupled, a notification can be helpful when you need to react to its completion

# Coding what to say:

```
// Describe the current filter state
- (NSString *)a11yDescription {
    NSString *myVoiceOverString =
        NFLXLocalizedString(@"a11yFilterShowingAllLabel", @"Showing All");

    // We have some kind of filtering going on
    if (![self isUnfiltered]) {

        BOOL isFilteredBySubgenre = ![_subGenresView isHidden];

        myVoiceOverString = [NSString
            stringWithFormat:NFLXLocalizedString(@"a11yFilterStatusFormat",
                @"Showing TV Shows, Documentaries"),
                _topLevelView.menuItem.name, isFilteredBySubgenre ?
                _subGenresView.menuItem.name : @""];
    }

    return myVoiceOverString;
}
```

We have to describe the current filter state



# Code for talking:

```
- (void)a11yAnnounceCurrentStatus {  
    if (!UIAccessibilityIsVoiceOverRunning()) {  
        return;  
    }  
  
    NSString *myVoiceOverString = [self a11yDescription];  
  
    UIAccessibilityPostNotification(UIAccessibilityAnnouncementNotification,  
        myVoiceOverString);  
}
```

When you're ready for VoiceOver to say something, you post a notification which causes VoiceOver to read whatever string you pass it.

# Stringing you along:

```
/* Formatter to indicate what will happen when tapping a filter button (e.g. Filter by TV Shows) */
"a11yFilterActionFormat" = "Filter by %@",;

/* Accessibility hint for a filter button that is already selected (e.g. TV Shows) to indicate that tapping it
   will allow the user to change to a new filter (e.g. Movies) */
"a11yFilterActionHint" = "Select a new filter";

/* Accessibility Label for the Lolomo Filters. To reset all filters, thereby showing all items (as opposed to
   filtered by TV/Movie/Genre, etc.) */
"a11yFilterShowAllLabel" = "Remove all category filters";

/* Accessibility Label for the Lolomo Filters. Indicates that there is no filtering going on */
"a11yFilterShowingAllLabel" = "No active category filters";

/* Formatter to indicate the current state of Lololmo filters (e.g. Showing TV Shows, Documentaries) */
"a11yFilterStatusFormat" = "Currently showing %1$@, %2$@";
```

Here's an example of Localized strings in support of accessibility.

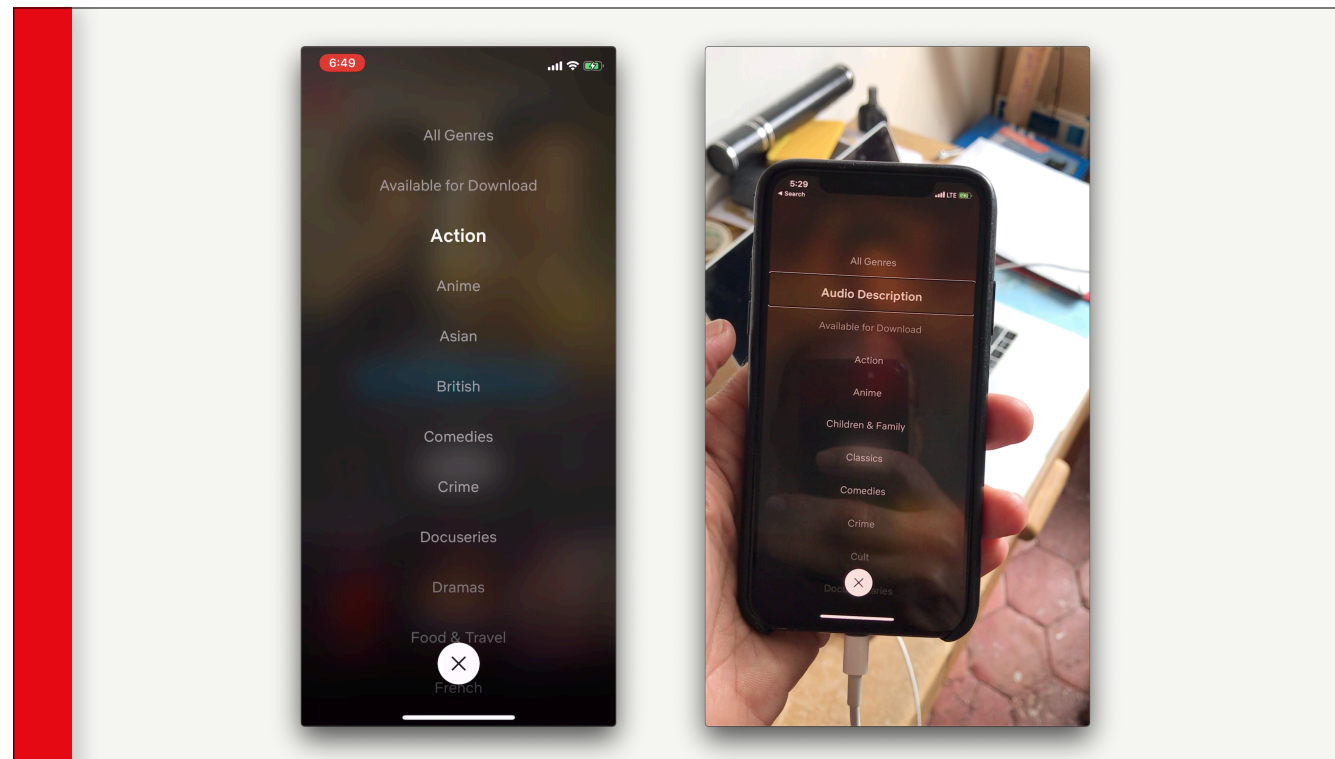
Pro Tip: help your friendly neighborhood translator by providing context in the comments.

The a11y prefix helps them know that this text will be read aloud, so they need to be concise (Think Pushkin/Eugene Onegin, not Tolstoy/War & Peace)



# Being kind

Sometimes, in order to provide a great user experience, you need to adapt it. If VoiceOver is running, that's a great signal that the user might want to quickly get to a list of titles with Audio Description



On the left is the regular Subgenres menu. Audio Description is all the way to the bottom.

On the right is the same list, but with the Audio Description genre inserted at the top



**UIAccessibilityVoiceOverStatusDidChangeNotification**

This notification lets you know when VoiceOver is turned on or off

# Code for handling:

```
// We care when VoiceOver is switched on or off, so that we can refresh our subgenre list
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(handleVoiceOverChanged)
                                         name:UIAccessibilityVoiceOverStatusDidChangeNotification
                                         object:nil];

// We need to reorder the Subgenres menu when VoiceOver is toggled
- (void)handleVoiceOverChanged {
    [self configureSubgenresAddingTopLevelItem:NO];
}
```

Subscribe to `UIAccessibilityVoiceOverStatusDidChangeNotification` to react to a user turning VoiceOver on or off

# Mother, may I?

Does anyone know the game Mother May I (or Captain, may I?) It's a children's game where the children or crew members try to reach the person by making various moves. They must always say Mother May I before asking to do something.

There's something like this for VoiceOver



**UIAccessibilityIsVoiceOverRunning()**

Like it says on the tin, this function lets you ask if VoiceOver is currently running so you can know if it's appropriate to change some behavior, like in this case...



# Code for rearranging:

```
// The Audio Description subgenre is really a synthetic genre
NSUInteger myIndex = [[arr valueForKey:@"aroCategoryId"] indexOfObject:@"assistiveAudio"];

if (myIndex != NSNotFound) {

    // Find the correct item
    NFUIAROGalleryMenuItem *myItem = [arr objectAtIndex:myIndex];

    [arr removeObjectAtIndex:myIndex];

    // Let's be kind to VoiceOver users by putting Audio Description at the top,
    // where it's easy to select
    if (UIAccessibilityIsVoiceOverRunning()) {
        [arr insertObject:myItem atIndex:1];
    } else {
        // Leave it at the bottom where it originally was
        [arr addObject:myItem];
    }
}
```

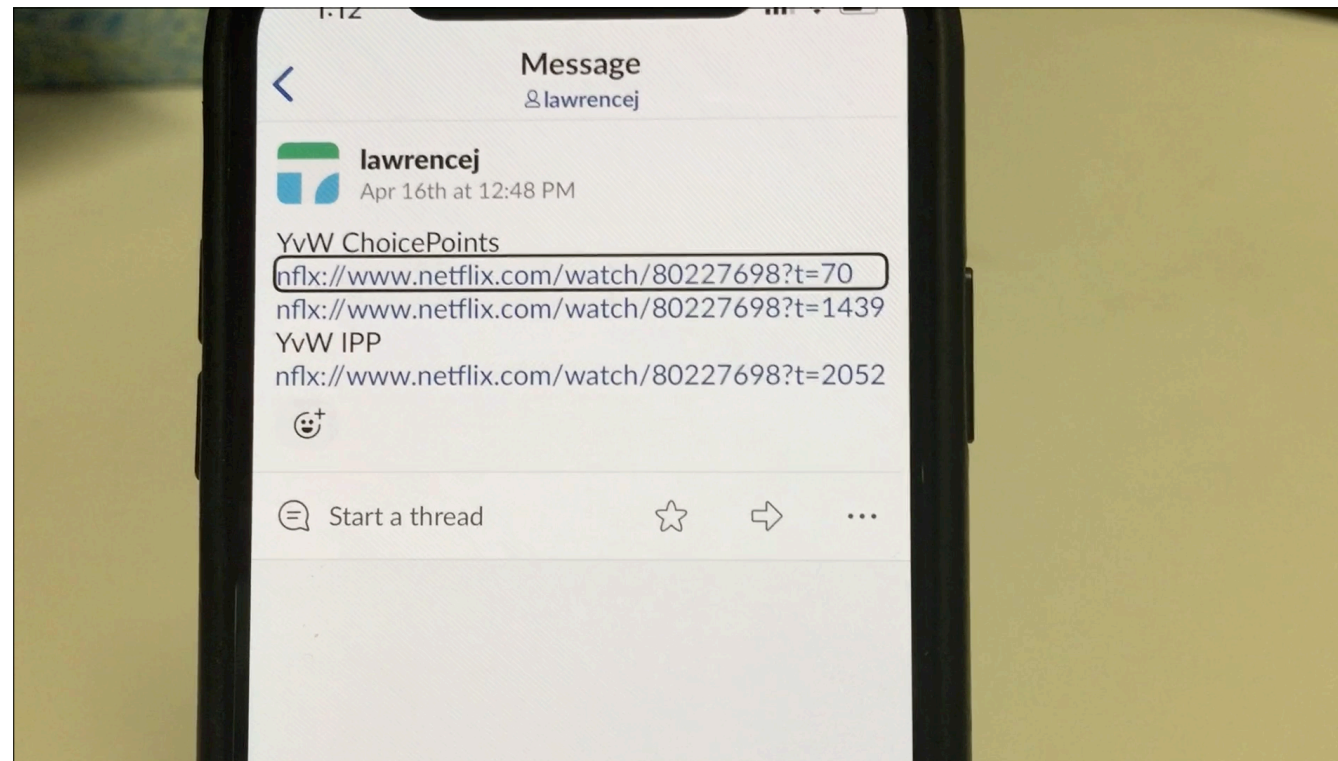


# Knowing when to be silent

While we're working on our apps, we have to keep in mind, is that despite our big egos as programmers, we're doing our work in service to the content, so in an ideal world, the User Interface should get out of the way.

Up until recently, when VoiceOver was running, and the video player was being used, we'd leave the playback controls permanently on the screen. As it turned out, that was a bad user experience for a number of reasons.

One is that it's visually disturbing for people with low-vision. Secondly, it can make it awkward for people who might be watching on a device (like an iPad) with a friend who has full vision.



An example of interactive titles with VoiceOver

(After video) A couple of things to notice. The first is that every view has an array of accessibilityElements. This is typically calculated for you. It's basically the array of all UI elements whose isAccessibilityElement value is YES.

The OS typically calculates the best order from top to bottom of the screen, but if you want to explicitly set an order, you can maintain and create your own array of elements. This can be helpful if some elements remain in the view hierarchy, but may be invisible (alpha 0) but not hidden.

# The code:

```
#pragma mark - Animations

- (void)animateInChoices:(NSArray <NFChoiceModel *> *)choices btns:(NSArray<UIButton *> *)btns
{
    NSUInteger count = btns.count;

    __weak typeof(self) weakSelf = self;
    [CATransaction begin];
    [CATransaction setCompletionBlock:^(
        if (weakSelf)
        {
            // Focus for VoiceOver users
            UIAccessibilityPostNotification(UIAccessibilityScreenChangedNotification, weakSelf);
        }
    )];
};
```

This is the code which presents the choices. We call them choice points. The important bit here is using `UIAccessibilityPostNotification` with the constant `UIAccessibilityScreenChangedNotification` to indicate to VoiceOver that there was a layout change, and to reset the focus on the first item in the `accessibilityElements` array.

The absence of controls during normal playback is what makes the arrival of choice points stand out, using a subtle audio cue which VoiceOver users are accustomed to hearing.

# A kind gesture

There are two shortcuts which are commonly used by VoiceOver users to help them navigate quickly.

One is the “magic tap” which toggles the most commonly used function on a screen, which for us is playback.

The accessibility escape lets users dismiss a modal view quickly with a two-finger z gesture



Magic tap and accessibility escape

# The code:

```
/*
 * The Magic Tap is a two finger tap used to implement the most common useful
 * action. Toggling Playback is the perfect choice here, especially now that controls
 * are hidden when VoiceOver is running.
 */
- (BOOL)accessibilityPerformMagicTap {

    // Bad things happen if we try to begin playback before playback is ready
    if (![self didFirstPlaybackBegin]) {
        return NO;
    }

    [self togglePlayback:nil];

    return YES;
}

// Let VoiceOver user dismiss using the standard two-finger Z gesture
- (BOOL)accessibilityPerformEscape {

    if ([self isFullScreen]) {
        [self toggleRotation:self.rotateButton];
    }

    return YES;
}

return NO;
}
```

The UIAccessibility protocol lets view controller implement methods which will react to certain gestures which are used. They follow the form accessibilityPerform\* Magic Tap, Escape, Scroll, etc.

Implementation of these methods is a matter of routing to existing methods you already would have in your code.

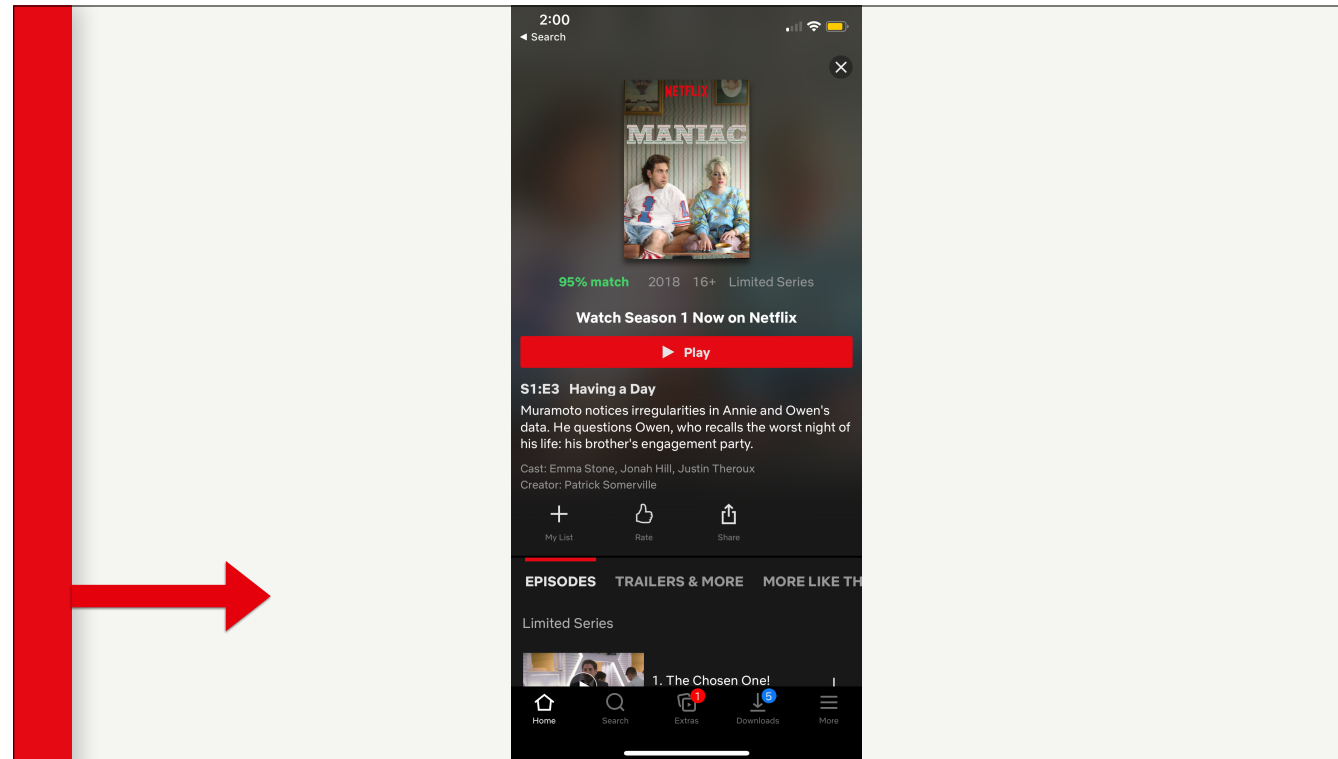


# **Accessibility lets you see problems**

A very useful side-effect of working on Accessibility is that it can often uncover UI challenges that effect all users.

I have an example of that.



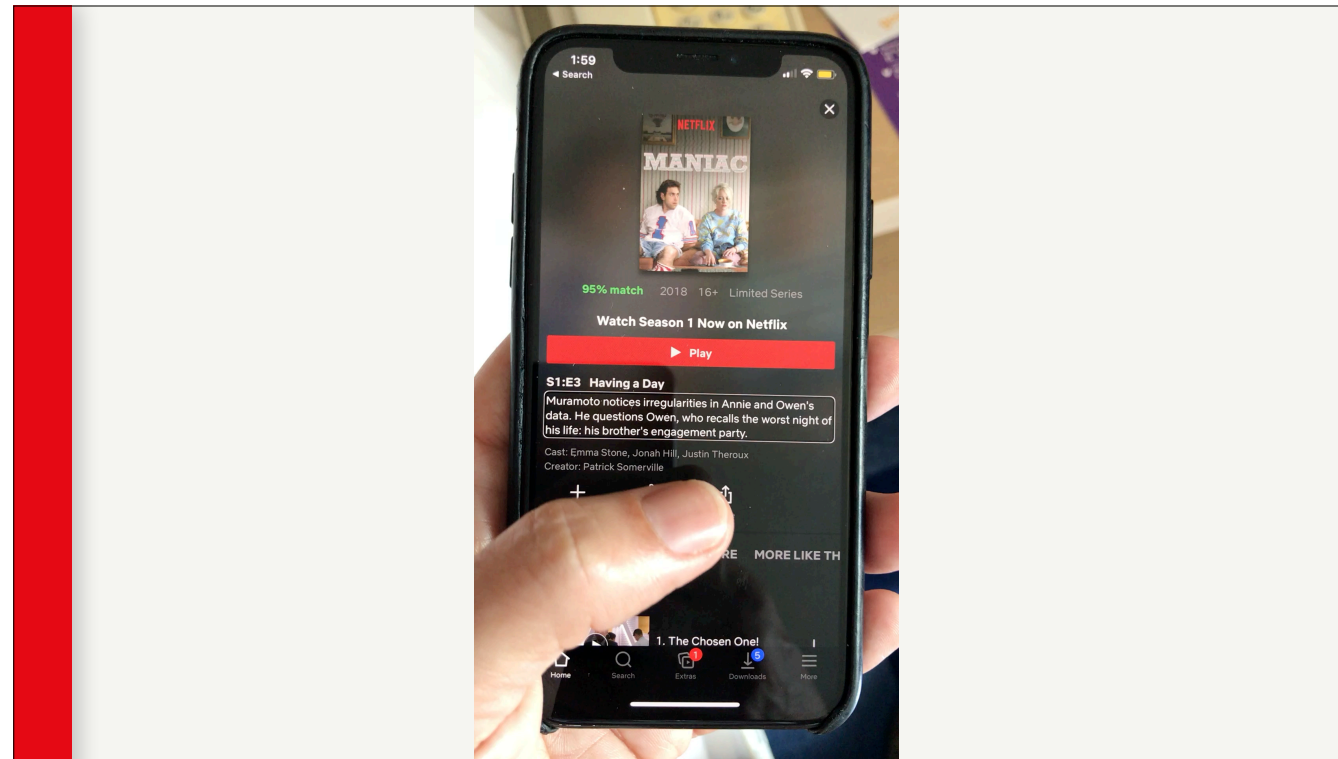


This is an example of what we call our Display Page. It gives more information about a particular title.

One of the items here is a list of Cast members.

Looking at it, would you know that it's tappable? Probably not.

When VoiceOver is running, it's actually more clear. Let's have a look.



As you could see, it's makes a couple of things really clear. One is that the first word "Cast" is highlighted as a "heading" so the user knows there's a list of items to follow.

Following that, each item has a hint of "Button" so that the user knows that you can take an action.

Let's look

# The code:

```
self.evidenceStarringLabel.text = [self.model.detailsEntity castString];  
self.evidenceStarringLabel.accessibilityLabel = NFLXLocalizedString(@"castAccessibilityLabel", @"");  
self.evidenceStarringLabel.accessibilityTraits = UIAccessibilityTraitHeader;  
  
[self addLinksFromPersonArray:self.model.detailsEntity.actors  
  toLinkLabel:self.evidenceStarringLabel  
  withDescription:[self.model.detailsEntity castString]];
```

Here, we indicate that the label itself has a trait of “header” which signals to the user that there are related items to follow.

# The code:

```
Argo > Argo > Common > View > Labels > NFUILinkLabel.m > M -accessibilityElements
accessibilityElements 11 matches + Aa Contain

NSString *accessibilityLabel = [sourceText substringWithRange:link.range];
NSString *accessibilityValue = link.accessibilityValue;

if (accessibilityLabel)
{
    UIAccessibilityElement *linkElement = [[UIAccessibilityElement alloc] initWithAccessibilityContainer:self];
    linkElement.accessibilityTraits = UIAccessibilityTraitButton;
    linkElement.accessibilityFrame = [self convertRect:[self boundingRectForCharacterRange:link.range]
                                                toView:self.window];
    linkElement.accessibilityLabel = accessibilityLabel;

    if (![accessibilityLabel isEqualToString:accessibilityValue])
    {
        linkElement.accessibilityValue = accessibilityValue;
    }
}
```

This is an implementation of the accessibilityElements of our UILabel subclass. The array of links and their handlers was configured in the superview.

What's important here is that we're creating "synthetic" UI elements, whose highlight frame is calculated in window coordinates, and setting their traits to button.

It's pretty complicated, but the irony is that VoiceOver users can "see" things better than those who don't!

This is a design issue that we've struggled with. The cheap solution would be to add an underline to make it look like a classic hyperlink. This offended the aesthetic sensibilities of our designers (and I agree), but it comes at a usability cost.

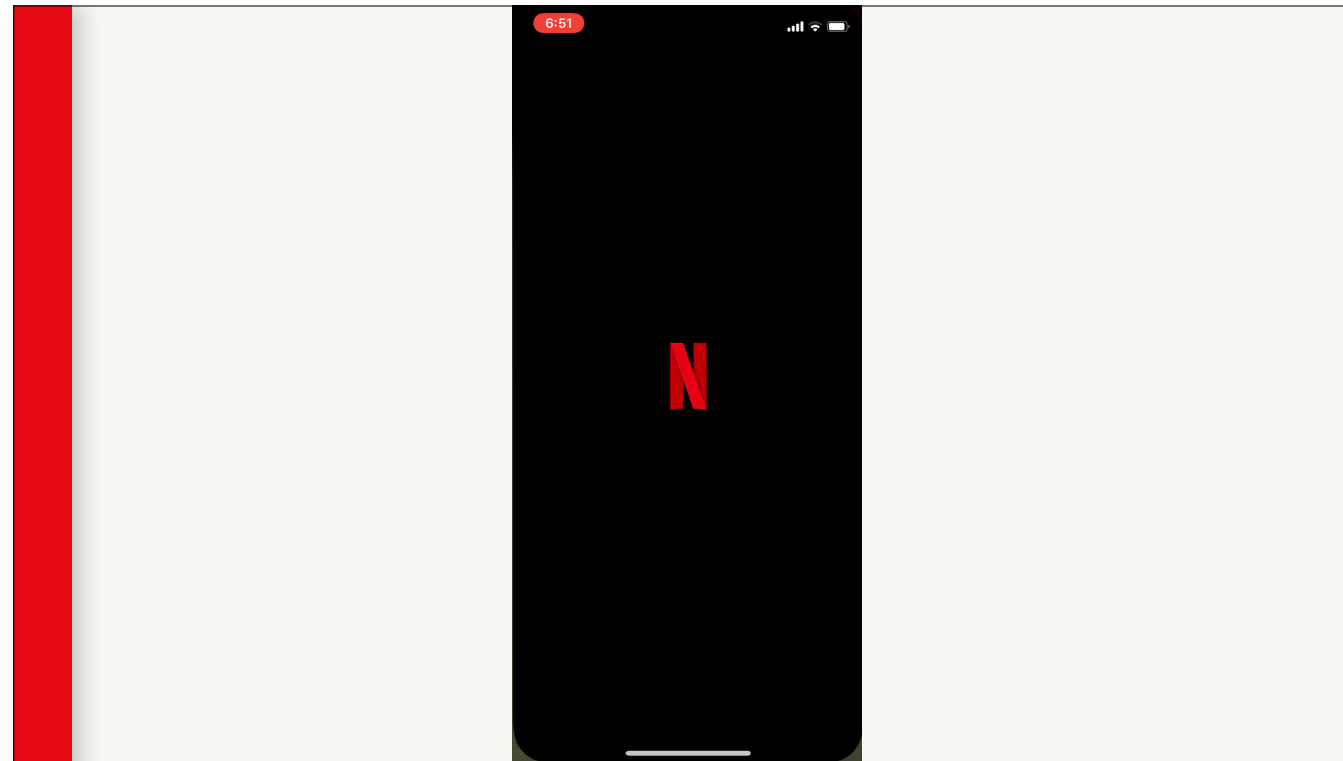
# Repurposing

What's good for one set of users, may be good other users. Here's an example

# Low Bandwidth Conditions

As you might imagine, we spend a lot of time thinking about how to make our mobile apps work well in low bandwidth conditions.

I have a screen movie which shows an example. Let's ask the projectionist to roll the film...



(after video) When there is a poor network conditions, we use a UILabel to indicate the title for an image. This is just like the alt tag for the HTML IMG entity. Y'all might not have ever had to browse the web with a dial up connection, but I did, and it was considered good design to have these tags to give some idea while the page was slowly loading.

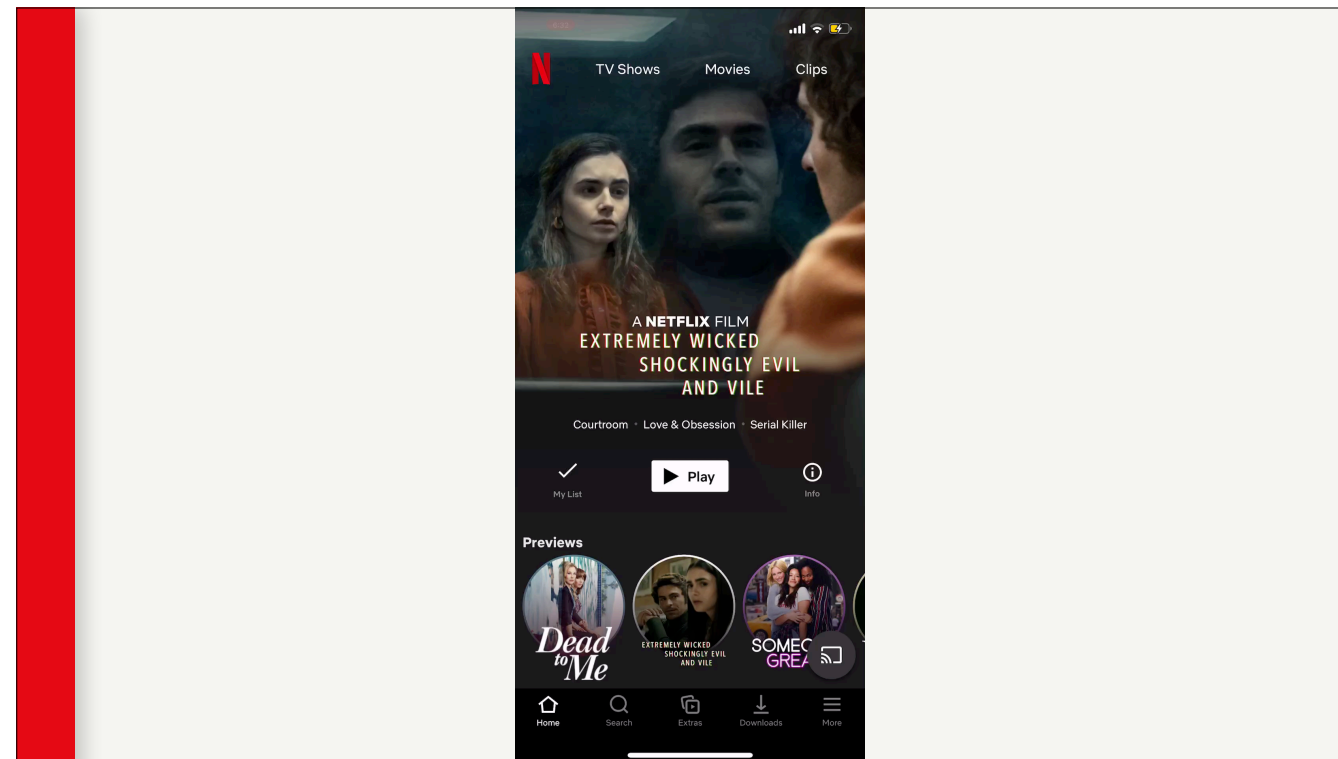


# Low Vision Conditions

We've talked about blind users, but there are actually far more people with low-vision. For these people, they can see some things, but having high contrast, big type is a big help. It's much easier than trying to decipher a tile from a colorful box art where each title is in a different style and color typeface.

I have another Oscar-worthy film that shows this. Would you like to see it?





Here's an experiment we're working on to enable a special mode where box art images aren't loaded, and instead large type labels are substituted.

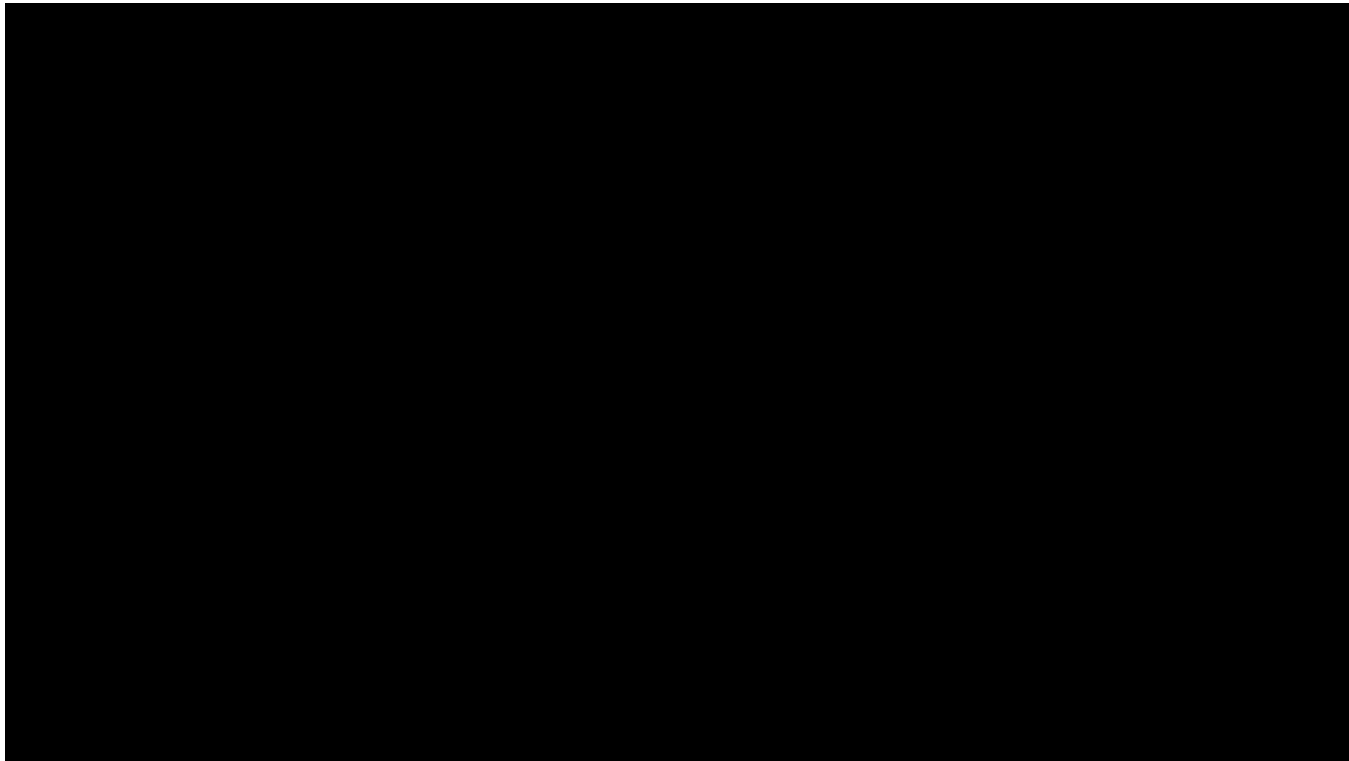


# **Accessibility is a *shared* practice**

Working on Accessibility is a practice. You have to evangelize it with your colleagues so that they care about it as a matter of course, just like with performance, or localization. Things work much better when everybody is at least a bit familiar.

Recently, we formed an Employee Resource Group for Accessibility at Netflix. I've got some stickers if you'd like.

We made a short video, I'd like to show you a little bit of it.



A video we shot to help explain why we care

# Testing, Testing: 1...2...3

So, maybe, there are people who remain unmoved. Maybe this guy in the back with his arms crossed is thinking Accessibility is a waste of time.

So, here's another angle: Automated testing.



# **accessibilityIdentifier**

At the beginning, I talked about the aspects of UIAccessibility protocol that are targeted towards human being, accessibility labels and hints which tell the user what something is and what it will do.

We talked accessibility traits, which tell the user how an element will behave.

But there's a great way of using UIAccessibility for uniquely identifying a UI Element

# **getElementById()**

Some of you have worked on web apps, so you're familiar with this. In the HTML DOM world, this is how you look up DOM elements so you can manipulate them.

# XCUIElement

- Defines a UIElement which can be queried, inspected and acted upon.
- You can query by accessibilityIdentifier

XCUIElement defines an UI Element in your app which can be queried, inspected (does it exist, is it tappable, etc.)

It uses a UI Element's accessibilityIdentifier so that you can find all items matching an NSPrecicate

# Code for identification:

```
-(void)setModel:(NFUIPipContinueWatchingCellViewModel*)model {
    __weak __typeof(self) weakSelf = self;

    self.contentView.accessibilityIdentifier = [NSString stringWithFormat:@"cwBarItem%@", @(model.indexPath.row)];

    if(model) {
        _model = model;

        _imageView.accessibilityIdentifier = [NSString stringWithFormat:@"cwBarItemImageView%@",
            @(model.indexPath.row)];
        _titleLabel.accessibilityIdentifier = [NSString stringWithFormat:@"cwBarTitleLabel%@",
            @(model.indexPath.row)];
        _openCloseImage.accessibilityIdentifier = [NSString stringWithFormat:@"cwBarOpenCloseButton%@",
            @(model.indexPath.row)];
        _timeRemainingLabel.accessibilityIdentifier = [NSString stringWithFormat:@"cwBarTimeRemainingLabel%@",
            @(model.indexPath.row)];
        _metalabel.accessibilityIdentifier = [NSString stringWithFormat:@"cwBarMetaLabel%@",
            @(model.indexPath.row)];
        _bookmarkContainer.accessibilityIdentifier = [NSString stringWithFormat:@"cwBarBookmarkContainer%@",
            @(model.indexPath.row)];
    }
}
```

This is an example of using a common prefix to name the various elements in a Collection View Cell.

The identifier also contains the indexPath's row which is helpful



# Code for finding:

```
- (void)tapOnCWBarItemImage {
    [[TestCase getTestRun] startStep:@"Tap on CW Bar item image"];
    NSPredicate* cwBarPredicate = [NSPredicate predicateWithFormat:@"identifier CONTAINS %@",
        cwBarItemImageIdentifier];
    XCUIElementQuery* cwBarQueryQuery = [self.app.images matchingPredicate:cwBarPredicate];
    NSArray<XCUIElement*>* allCwBarItems = cwBarQueryQuery.allElementsBoundByIndex;
    XCUIElement* cwBarItem = allCwBarItems[0];
    [self waitForUIElementToExist:cwBarItem timeout:5];
    if (!cwBarItem.exists) {
        @throw [self exception:@"CW Bar item image Not Found" reason:@"The image we are looking for didn't exist
            after 5 seconds."];
    }
    else {
        [cwBarItem tap];
    }
}
```

Notice here the use of NSPredicate which you might be using for things like auto-completing text fields.

Predicates can be used to find a subset of XCUIElements (like Continue Watching items)

From there, your test does what it needs to: check for existence, execute a tap, etc.

UIAccessibility makes automated testing possible. So even if you have no love in your heart for helping people with disabilities, the fact that engineers at Apple did, allowed other engineers to make a testing framework.

It's the exact same concept as mandated curb ramps helping everyone.



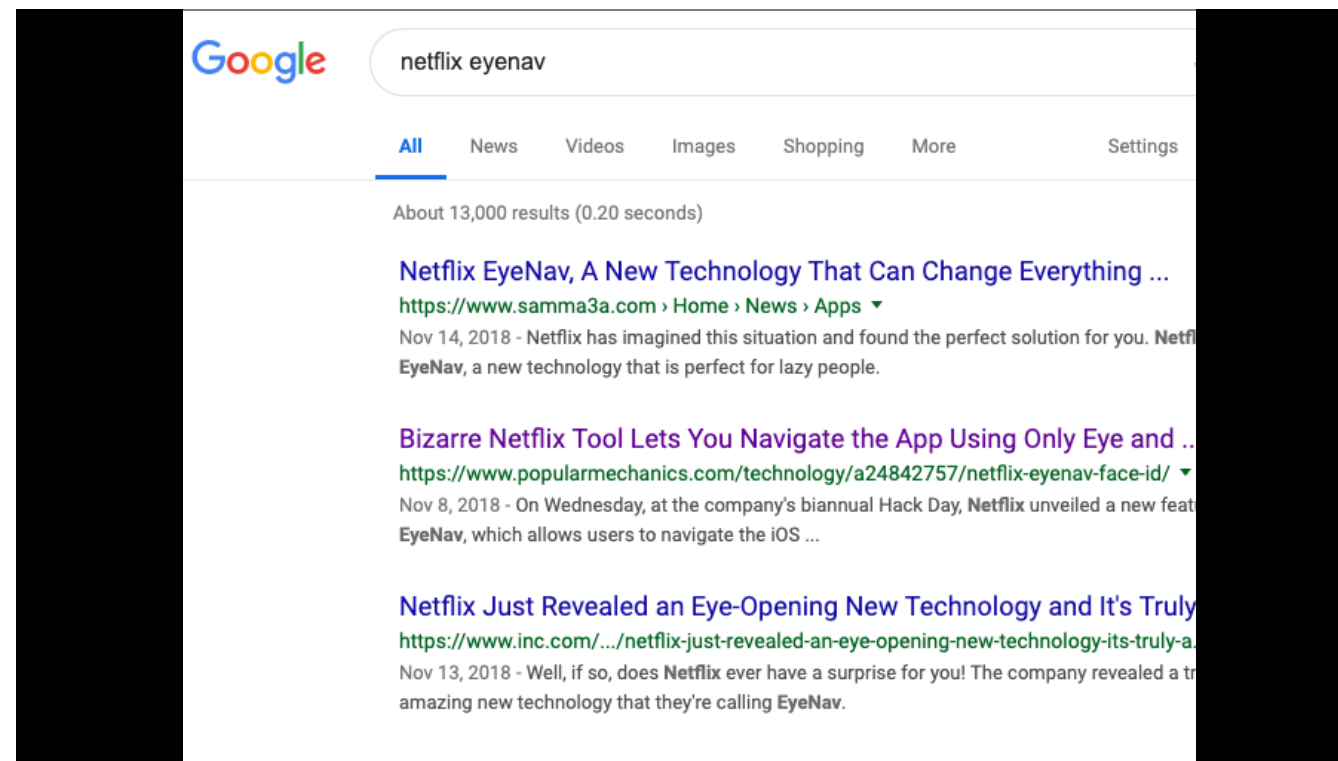
# Having fun

Up until now, we've only covered VoiceOver, but there are many other accessibility techniques, like help from people with limited motor skills.

Here's an example from a Hack Day we had last October.



This is a video of our HackDay project using AR Kit to allow navigation of the Netflix app using just your eyes.



After releasing the video in a blog post, we got tons of press,

POPULAR MECHANICS


TECHNOLOGY CARS TOOLS DEFENSE STAY WARM!

SUBSCRIBE NEWSLETTER

# Kooky Netflix Hack Lets You Navigate the App Using Only Eye and Facial Movements

Netflix and chill takes on a whole new dimension.


By Sam Blum Nov 8, 2018



NETFLIX/YOUTUBE

The future of Netflix and chill might be more like Netflix and stare.

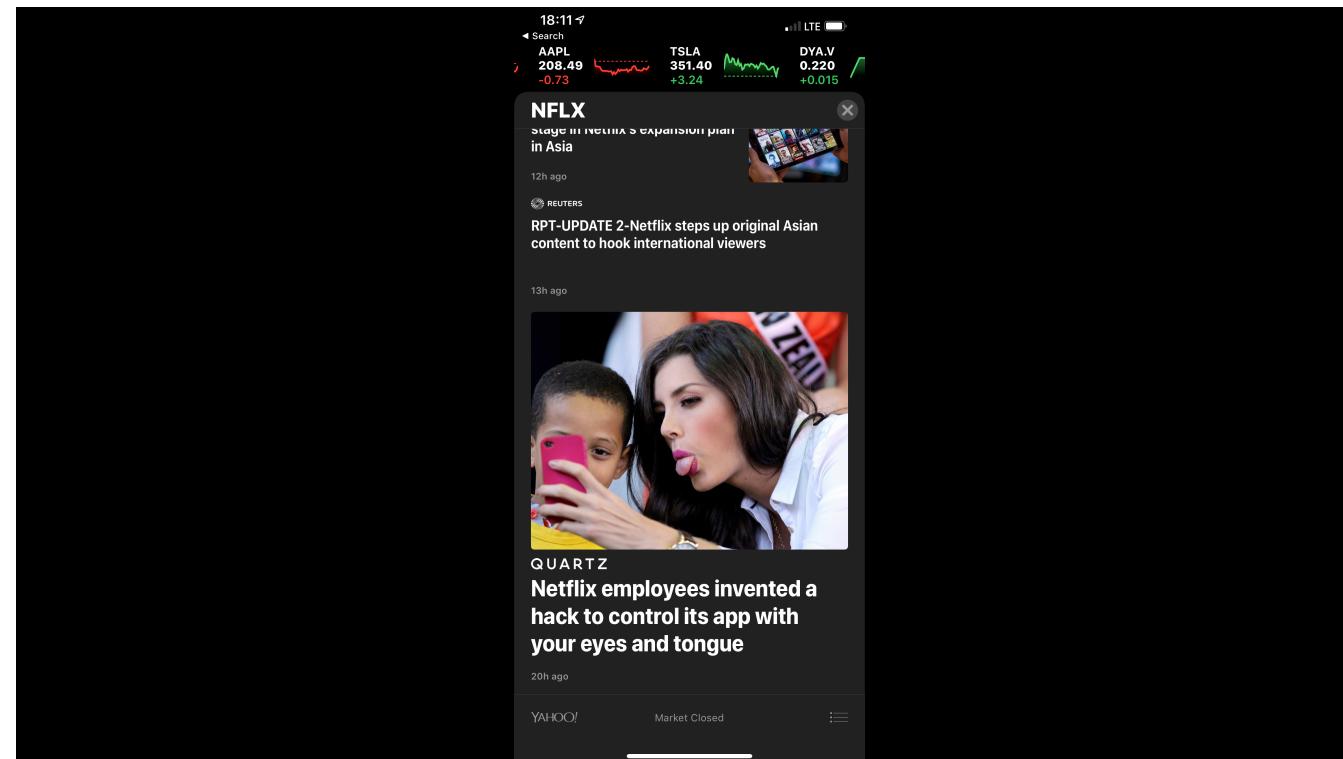
On Wednesday, at the company's biannual Hack Day, Netflix unveiled a new feature called EyeNav, which allows users to navigate the iOS version of the app using only eye movements and facial expressions, including sticking out your tongue. Really. Enabled by Apple's FaceID and its [augmented-reality platform Arkit](#), EyeNav is probably the closest thing



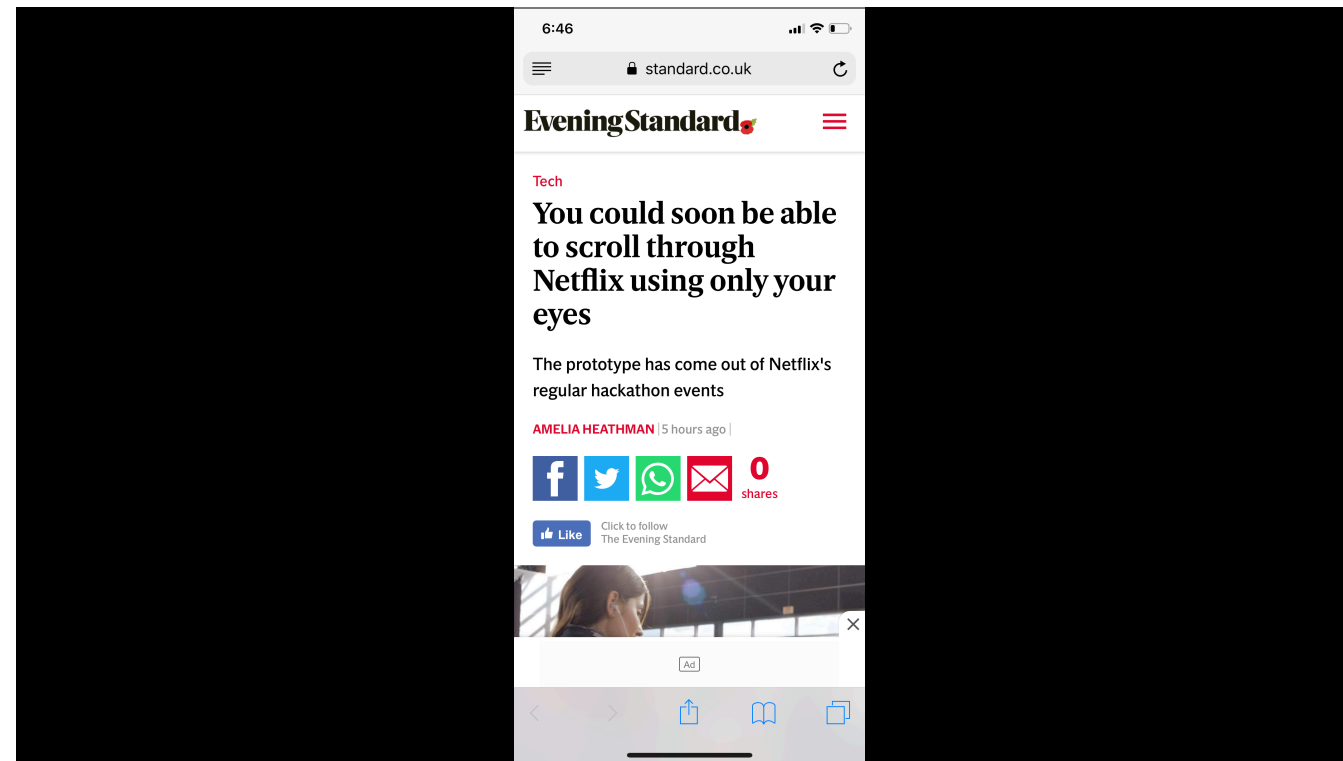
CEFCU  
Not a bank. Better.

Better rates.

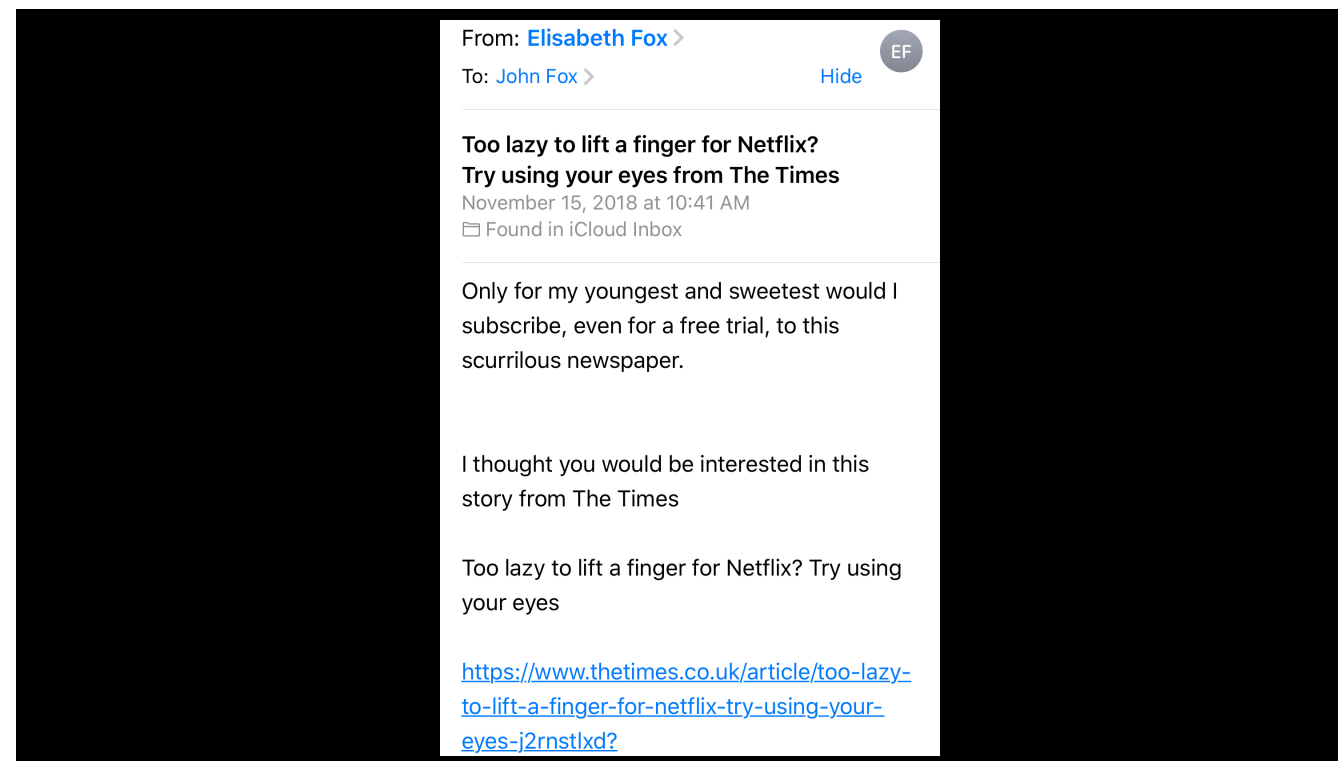
We were covered in both the industry press like Variety, which is a Hollywood weekly magazine, along with tech and business press (Popular Mechanics, Fortune).



People really locked on to the idea of sticking your tongue out at the screen as a way to dismiss a modal window. My colleagues and I took credit for Netflix stock going up on the day when all the press hit.



We even made it into general purpose press like the Evening Standard and London Sunday Times.



When we made it into The London Sunday Times, my mom, who's British, finally started taking my calls again.

Who says working on Accessibility isn't rewarding?!





# Just say...

So, I hope you've enjoyed my talk and have found some inspiration. If you're still on the fence about it, still wondering if it's worth taking the plunge, allow me to be Californian, and invite you to

Just...say...



**да!**

Da!!!!

A thick red vertical bar is positioned on the left side of the slide, extending from the top to the bottom of the main content area.

# Спасибо

I want to thank you for listening to my talk, and I want to take a moment to thank my colleagues at work.



I'd also like to give a special thanks to the Lighthouse for the Blind in San Francisco. We've learned so much through our collaboration with them. They have a great website, which I encourage you to visit and learn about what they do.



**Stay in touch:**

**@djembe on Twitter**

I invite you to keep in touch.

## Questions?



So now it's time for questions. (can tell the story of Canadian Steve's famous beard which made him look like a religious idol that people wanted to photograph. in addition to being jealous of all the attention people paid him, I had to suffer the indignity of his beard showing up in my face while we were riding around in a tuktuk)