Building Production-Quality Apps

Where you are right now

• Building apps with complex uses
• For a limited use case
Overview of topics

1. Graceful crashes
2. Making remote requests informative
3. Accessibility
   1. Dynamic text
   2. Colors
   3. Voiceover
4. Localization
5. Different devices
6. Clear and consistent interfaces

Graceful crashes: the worst case scenario

- Worst case: app stops running, data is lost

Common reasons:
- Array index out of bounds
- Outlet/action has changed
- Force unwrapping nil
- Try!
How do we avoid it?

- Always use safe unwraps
  - if let and guard let
  - Nil coalescing operator: “??”
- Use safe try...catch
  - do...try...catch block
  - try?
- Avoid c-style for loops as much as possible

Graceful crashes: a meh scenario

The user is using the app, fills in a form, and taps the “submit” button. Nothing happens. They tap it again. Nothing happens. Finally they give up and close the app.

- The app has not hard crashed, but this is still not good
- The user knows it didn’t work, but has no idea why
  - Could be network issue
  - Could be some sort of guard let that stops the submit function from doing anything.
- We can do better
Graceful crashes: The great solution

The user is using the app, fills in a form, and taps the “submit” button. A loading indicator spins for a few seconds, until an alert pops up with a frown face and the message “You are not connected to the internet”. The user closes the app.

When they are back home on wifi, they open the app. An alert pops up: “A draft of your post was saved. Would you like to restore it?”

Graceful crashes: The great solution

• Why is this a good solution?
  • It makes the issue (and what they need to do) very clear to the user
  • It gives them a sense of confidence that a single crash won’t lose their data
**Informative Remote Requests**

- Anytime your app has to make a call to a remote source, there is a loading time involved
  - EX: Lab 4
- The more informative you can be during the call, the more in control your user will feel

**Remote Request: Worst Case**

The app opens with a white screen. It remains that way for about 20 seconds until movie posters suddenly appear. When you click on one, the app freezes for 10 seconds and then opens a screen about the movie.
Remote Request: Good Case

The app opens immediately with a blank tab bar and navigation bar on the loading screen. Then, the title and tabs appear, while an activity indicator spins where the movies will be. After 20 seconds, they spinner disappears and the movies fade in.

Tapping a movie immediately navigates to a new screen, where the already-fetched details of the movie are visible. There is a spinner under “Lead actors”, until the lead actors appear. After a second, the image becomes higher res.

Tips for remote requests

1. Show as much as possible
   1. Titles, tab bars, sections, etc
2. Make the loading state clear: loading, loaded, error
3. Don’t load everything at once, load on-demand
4. Don’t reload things unless necessary
Accessibility

- Millions of people use iOS devices who use some sort of accessibility features
- If you do not make your app accessible, you lose lots of potential customers
- Apple provides very easy to use accessibility tools

Accessibility – Dynamic Text

**Expectation:** When the system font size is increased or decreased, your app should scale its text as well.

**Implementation:** Choose a system font style (such as “Body”) and click “Automatically Adjusts Font” on any label to make it dynamic.

Also: use stacks and other dynamic layout tools so that larger text doesn’t run off the edge of your screen or overlap.
Accessibility – Dynamic Text

Accessibility

Accessibility

I am a label

I am a label

Accessibility – Colors

• Various related issues:
  – Dark Mode
  – High Contrast
  – Smart Invert

• We can easily solve them all by using primarily system colors!
  – Any of the colors on the right (except “Black Color” and down) will adapt to dark mode, high contrast, etc.
Voiceover

• If you’ve never used it, voiceover uses the following commands
  – Tap the screen to hear the current selected item’s name
  – Double tap to select
  – Swipe left or right to focus the next or previous item
  – And several more...

• By default, voiceover is enabled, although descriptions are likely lacking
  – For example, an imageview will give the spoken description “image”, which doesn’t tell the user much.
Implementing Voiceover

- To improve voiceover, you simply set accessibility attributes for each view.
  1. Label – A concise name for the view
  2. Traits – Give information about state, such as isSelected
  3. Hint – Describes the action that the control runs
  4. Frame – The element’s location on screen
  5. Value – The element’s current value, like the number in a UIStepper

- Also, you can group elements in accessibility groups, if they don’t need to be separate

- There’s a ton more resources for improving voiceover, it’s too big a subject to master with a few slides

Localization

- Right now, non-English speaking countries are a huge untapped markets
- Even popular apps and websites often overlook localization, so by implementing it you can reach many more people

- Implementation:
  - Strings files
  - Strings file for storyboard
  - Using locale defaults for dates, currencies
  - Supporting RTL text
Device Sizes

- Often, you test using mainly one simulator device or real device

- Even with good use of constraints, your app might not look great on a much bigger or smaller screen

Device Sizes

- How do we deal with it?

- Rely on simple layouts, lots of content-focused apps
- Use UICollectionView for dynamic content lists
- Avoid the “fixed-height, full-width” trap that I fell into in the previous slide
- Use “Vary for Traits” reasonably
- Use Horizontal and Vertical stacks
- Understand device paradigms, such as Split View on iPad
Clear and Consistent Interfaces

- Even if you build an amazing app, if the interface is confusing or inconsistent, it will be harder to appeal to users

- Generally, the easiest way to be consistent is to follow Apple design guidelines
  - Examples:
    - Minimal shadows
    - Simple backgrounds
    - Default navigation/tab bars

Examples

GitHub App – Good!
- Uses standard tab bar, navigation header
- Uses grouped UITableView
- Nice custom icons
- Shows most important content first
Examples

GET Food App – bad 😞

- Text gets cut off
- Uses hamburger menu (≡) when a tab bar would suffice
- Weird navigation

Examples


Pros:
- Strong visual style
- Good use of tab bar and nav bar
- Clear view hierarchy

Cons:
- Advertisements
- Busy
- Options such as location, movie type, and sort are scattered.
## Do’s and Don’ts

**DO:**
- Use the built-in navigation bar, tab bar, table view, etc to give the user familiarity
- Use custom colors, icons, fonts, and layouts to make things unique
- Build content-first interfaces
- Embrace iOS features like Haptic feedback and item previews

**DON’T:**
- Modify tab bars or navigation to being unrecognizable
- Try to include too much content on one page
- Use hamburger menus
- Build design-first interfaces
- Try to port a website or Android app as-is