Week 3 WDD330

lunes, 1 de mayo de 2023 20:50

RUTH CHADOMA!

10 MOBILE UX DESIGN PRINCIPLES YOU SHOULD KNOW

BASIC USABILITY PRINCIPLES:

- Learnability: How easily first-time users complete basic tasks.
- Efficiency: How quickly users perform basic tasks once they've learned the design.
- Memorability: The ability of users to remember how to use the system.
- Errors: The amount and severity of errors users make and how easily they can correct them.
- Satisfaction: How pleasant the experience of using the design was.

1. CONTENT PRIORITIZATION

- HUMANS HAVE 8 SECONDS OF ATTENTION
- LESS IS MORE
  - KEEP SIMPLE DESIGNS AND MINIMUN ELEMENTS
  - DISPLAY ONLY ESSENCIAL CONTENT & FUNCTIONALITIES THE USER NEEDS
  - SECONDARY CONTENT SHOULD BE AVAILABLE THROUGH A MENU
  - SIMPLE MENU LISTS TO NOT CONFUSE USERS
  - USE ICONS OR TEXT
  - PRIORITIZE CONTENT

2. MAKE NAVIGATION INTUITIVE

> *"If the user can't use it, it doesn't work."*– Susan Dray

- USERS SHOULD BE ABLE TO NAVIGATE INTUITIVELY WITHOUT ANY EXPLANATION

IF THE SITE IS TOO COMPLEX TO WORK THROUGH YOU WILL LOSE USERS

- USERS SHOULD BE ABLE TO NAVIGATE INTUITIVELY WITHOUT ANY EXPLANATION
- IF THE SITE IS TOO COMPLEX TO WORK THROUGH, YOU WILL LOSE USERS.
- SIMPLIFY PROCESS IN A SINGLE APP WITH ALL THE INFORMATION NEEDED AVAILABLE.
- THE USER SHOULD UNDERSTAND WHERE THEY ARE AND WHAT TO DO NEXT

3. TOUCHSCREEN TARGET SIZES

- MAKE INTERFACE ELEMENTS BIG ENOUGH
- MINIMUM TARGET SIZE: 44 PX WIDE x 44 PIXELS TALL (APPLE)
  SUGGESTED: BETWEEN 34PX (9mm) AND 26 PX (7mm) (WINDOWS)
- SPACING BETWEEN TARGETS IS IMPORTANT TOO
  ↳ DON'T BE AFRAID TO USE ENOUGH WHITE SPACES WITH RELATED CONTENT

4. PROVIDE USER CONTROL

- ALLOW USERS TO MAKE DECISIONS TO PERSONALIZE THEIR JOURNEY
  → CHANGING SETTINGS
  → CONTROLLING NOTIFICATIONS
  → CANCELING ACTIONS

- SUGGEST ACTIONS
- PROVIDE WARNINGS
- LET USER KNOW WHATS going on (STATUS, FEEDBACK)
- LET USER DEMO THE APP

5. LEGIBLE TEXT CONTENT

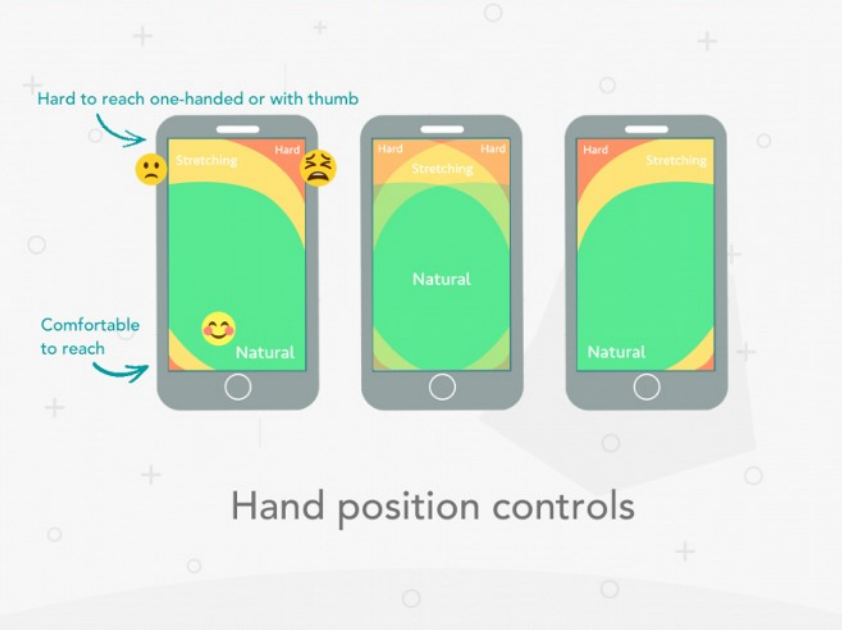
- READABILITY
- COMMUNICATE YOUR DESIGNS CLEARLY AND SIMPLE
- KEEP A BALANCE BETWEEN LEGIBILITY AND SPACE CONSERVATION
- USE TEXT >= 16 PX (11 POINTS)
- DON'T USE UNNECESSARY LARGE FONT SIZES
- USE [illegible]

- DON'T USE UNNECESSARY LARGE FONT SIZES
- USE 30–40 CHARACTER PER LINE FOR MOBILE
- SPACING AND LAYOUT

6. MAKE INTERFACE ELEMENTS CLEARLY VISIBLE

- HAVE SUFFICIENT CONTRAST BETWEEN CONTENT AND BACKGROUND SO ITS LEGIBLE IN ANY SETTING, EVEN OUTSIDE.
- USE CONTRAST RATIO BALANCE
- TEST WITH USERS

7. HAND POSITION CONTROLS



Hand position controls

- 49% OF PEOPLE RELY ON ONE THUMB
- COMMON FEATURES SHOULD BE PLACED IN EASILY ACCESSIBLE REGIONS
- ACTIONS SUCH AS DELETE BUTTONS SHOULD BE PLACES IN AREAS HARDER TO REACH TO AVOID ERRORS
- MADE ACCESSIBILITY BEHAVIOURAL FEATURES TO MAKE NAV. EASIER (ex: DOBLE TAPPING)

- Keep in mind RIGHT- AND LEFT- HANDEDNESS

8. Minimize Data Input

- Minimize THE NEED TO ENTER DATA
- Reduce TYPING BY SHORTENING FORMS
- Remove UNNECESSARY FIELDS.
- Use "REMEMBER ME" OPTIONS
- PROVIDE AUTOCOMPLETE, RECENT SEARCH, HISTORY & LOCATION DETECTION.
- DISPLAY KEYBOARD VARIATIONS (NUMERIC KEYPAD FOR NUMBERS, ETC)
- Keep USER ENGAGED AND SATISFIED WITH YOUR PRODUCT

9. Create a Seamless Experience

- Focus on KEY USER GOALS BY
  - REDUCING FRICTION
  - Minimizing STEPS
  - // PAGE LOADS
- MAKE CONTENT ACCESSIBLE EVEN WITHOUT INTERNET
- PROVIDE ALTERNATIVE PATHS
- USE MOBILE PHONE'S FEATURES (CAMERA, GPS, ETC)
- MAKE SMOOTH INTERACTIONS WITH DESIGN
- SYNCHRONIZATION ACROSS DEVICES TO NOT INTERRUPT WORKFLOW
- CREATE PLEASANT AND ADDICTIVE EXPERIENCES
- UNDERSTAND USER NEEDS BY COMMUNICATION

10. Test your Design

- TEST EARLY. TEST OFTEN
- CONTINUOUSLY TEST AND OPTIMIZE
- TEST DIFFERENT USABILITY FEATURES (DESIGNS, LAYOUT, ETC)

- BUILD YOUR PRODUCTS WITH A USER-CENTERED APPROACH
- AFTER EACH TESTING, UNCOVER NEW WAYS TO IMPROVE

INTRODUCTION TO WEB APIS

APIs = APPLICATION PROGRAMMING INTERFACES

↳ ALLOW DEVELOPERS TO CREATE COMPLEX FUNCTIONALITY MORE EASILY

APIs in CLIENT-SIDE JS:

↳ **BROWSER APIs**
- ARE BUILT INTO YOUR WEB BROWSER
- EXPOSE DATA FROM THE BROWSER AND
- SURROUNDING COMPUTER ENVIRONMENT
- DO USEFUL COMPLEX THINGS

↳ **THIRD-PARTY APIs**
- ARE NOT BUILT INTO BROWSER BY DEFAULT
- YOU HAVE TO RETRIVE THEIR CODE AND INFO FROM SOMEWHERE ON THE WEB.

JAVASCRIPT ⇒ A HIGH-LEVEL SCRIPTING LANGUAGE BUILT INTO BROWSERS
ALLOWS YOU TO IMPLEMENT FUNCTIONALITY ON WEB PAGES/APPS

JS LIBRARIES ⇒ CUSTOM FUNCTIONS THAT YOU CAN ATTACH TO YOUR WEB TO SPEED UP OR ENABLE WRITING COMMON FUNCTIONALITY

JS FRAMEWORKS ⇒ PACKAGES OF HTML, CSS, JS AND OTHER TECHNOLOGIES
THEY ARE USE TO WRITE AN ENTIRE WEB APP FROM SCRATCH
THE FRAMEWORK CALLS THE DEVELOPER'S CODE

WHAT CAN APIs DO?

Common Browser APIs:

- APIs for manipulating documents: loaded into the browser (example: DOM)
- APIs that fetch data from the server: to update small sections of a web page on their own. (ex: fetch (AJAX))
- APIs for drawing and manipulating graphics Allow you to programmatically update pixel data to create 2D and 3D scenes. (ex: Canvas, WebGL)
- Audio and Video APIs: Allow you to create and do things with multimedia (playing audio, video, subtitles, edition, effects, etc) (Audio API, WebRTC)
- Device APIs: Enable you to interact with device hardware (ex: Geolocation API)
- Client-side storage APIs: enable you to store data on the client-side (Web Storage API, IndexedDB API)

Common Third-party APIs:

- Twitter API: displaying things like your lastest tweets on your website
- Map API: Google Maps API, Mapquest
- Facebook suit of APIs: enables you to use various parts of the Facebook ecosystem to benefit your app
- Telegram APIs: to embed content from Telegram, and supports for bots
- Youtube API: to embed videos from youtube, playlist and more
- Pinterest API: provides tools to manage Pinterest boards

- PINTEREST API: PROVIDES TOOLS TO MANAGE PINTEREST BOARDS
- TWILIO API: PROVIDES A FRAMEWORK FOR BUILDING VOICE AND VIDEO CALL FUNCTIONALITY
- MASTODON API: ENABLES YOU TO MANIPULATE FEATURES OF THE MASTODON SOCIAL NETWORK PROGRAMMATICALLY

HOW DO APIs WORK?

1. THEY ARE BASED ON OBJECTS
2. THEY HAVE RECOGNIZABLE ENTRY POINTS HOW YOU ACCESS TO THE OBJECT, FOR EXAM: - DOM API → DOCUMENT()
   - WEB AUDIO API → AUDIOCONTEXT()
3. THEY OFTEN USE EVENTS TO HANDLE CHANGES IN STATE
4. THEY HAVE ADDITIONAL SECURITY MECHANISMS WHERE APPROPRIATE
   FOR EX = - SOME MODERN WEB APIs WILL ONLY WORK ON PAGES SERVED OVER HTTPS DUE TO TRANSMITTING POTENTIALLY SENSITIVE DATA
   - SOME WEB APIs REQUEST ENABLE PERMISSION (NOTIFICATION API)

DEPENDS ON HOW STRICT THE BROWSER IS.

SCHEDULING: setTimeout and setInterval

- SETTIMEOUT ⇒ ALLOWS TO RUN A FUCTION ONCE AFTER THE INTERVAL OF TIME
- SETINTERVAL ⇒ ALLOWS TO RUN A FUCTION REPEATEDLY, STARTING AFTER THE INTERVAL OF TIME, THEN REPEATING CONTINUOUSLY AT THAT INTERVAL

setTimeout

The syntax:

```
let timerId = setTimeout(func|code, [delay], [arg1], [arg2], ...)
```

Parameters:

`func|code`

Function or a string of code to execute. Usually, that's a function. For historical reasons, a string of code can be passed, but that's not recommended.

`delay`

The delay before run, in milliseconds (1000 ms = 1 second), by default 0.

`arg1`, `arg2` ...

Arguments for the function (OPTIONALS)

```
function sayHi(phrase, who) {
  alert( phrase + ', ' + who );
}
          Func   Delay   Arg1   Arg2
setTimeout(sayHi, 1000, "Hello", "John"); // Hello, John
```

FUNCTION REFERENCE DOES NOT NEED BRACKETS ()

CANCELLING WITH CLEARTIMEOUT

```
let timerId = setTimeout(...);   -> setInterval( )
clearTimeout(timerId);           -> clearInterval( )
```

WITH THE VALUE RETURNED BY SETTIMEOUT & SETINTERVAL

setInterval

The `setInterval` method has the same syntax as `setTimeout`:

```
let timerId = setInterval(func|code, [delay], [arg1], [arg2], ...)
```

All arguments have the same meaning. But unlike `setTimeout` it runs the function not only

All arguments have the same meaning. But unlike `setTimeout` it runs the function not only once, but regularly after the given interval of time.

To stop further calls, we should call `clearInterval(timerId)`.

NESTED SETTIMEOUT

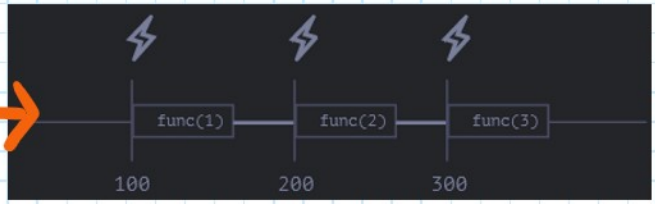
① Using TWO SETTIMEOUT

```
let timerId = setTimeout(function tick() {
  alert('tick');
  timerId = setTimeout(tick, 2000); // (*)
}, 2000);
```

- NESTED SETTIMEOUT IS A MORE FLEXIBLE METHOD THAN SETINTERVAL
- NESTED SETTIMEOUT ALLOWS TO SET THE DELAY BETWEEN THE EXECUTIONS MORE PRECISELY THAN SETINTERVAL

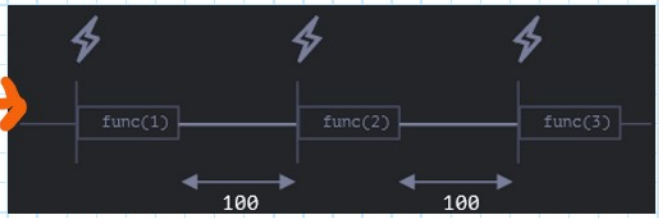
Let's compare two code fragments. The first one uses `setInterval`:

```
1  let i = 1;
2  setInterval(function() {
3    func(i++);
4  }, 100);
```



The second one uses nested `setTimeout`:

```
1  let i = 1;
2  setTimeout(function run() {
3    func(i++);
4    setTimeout(run, 100);
5  }, 100);
```



- THE REAL DELAY BETWEEN FUNC CALLS FOR SETINTERVAL IS LESS THAN IN THE CODE
- THE NESTED SETTIMEOUT GUARANTEES THE FIXED DELAY (HERE 100MS).

ZERO DELAY SETTIMEOUT:

SETTIMEOUT(FUNC, 0) OR SETTIMEOUT(FUNC)

↳ THIS SCHEDULE THE EXECUTION OF FUNC ASAP

↳ THE FUNCTION IS SCHEDULE TO RUN "RIGHT AFTER" THE CURRENT SCRIPT IS COMPLETE.

```
setTimeout(() => alert("World"));

alert("Hello");
```

- Zero delay is in fact not zero in a browser
  - The interval is forced to be at least 4 milliseconds.

All scheduling methods do not guarantee the exact delay.

For example, the in-browser timer may slow down for a lot of reasons:

- The CPU is overloaded.
- The browser tab is in the background mode.
- The laptop is on battery saving mode.