

iOS alapú szoftverfejlesztés

Dr. Blázovics László

Blazovics.Laszlo@aut.bme.hu

2021. Október 05.



Automatizálási és
Alkalmazott
Informatikai Tanszék

Auto Layout

Alapok

iPhone képernyő 2007 - 2012

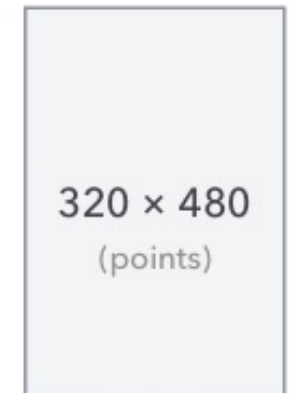
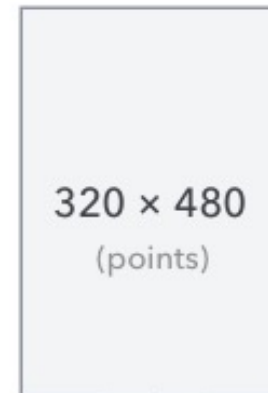
- Egy készülék méret
- Egyféle alkalmazás-felbontás
 - > Retina kijelző -> @2x
- Új platform
 - > „Alacsonyabb” igények
 - > Többnyelvűség még nem jellemző
 - > A fejlesztők és a designerek is „tanulják” még



4, 4s

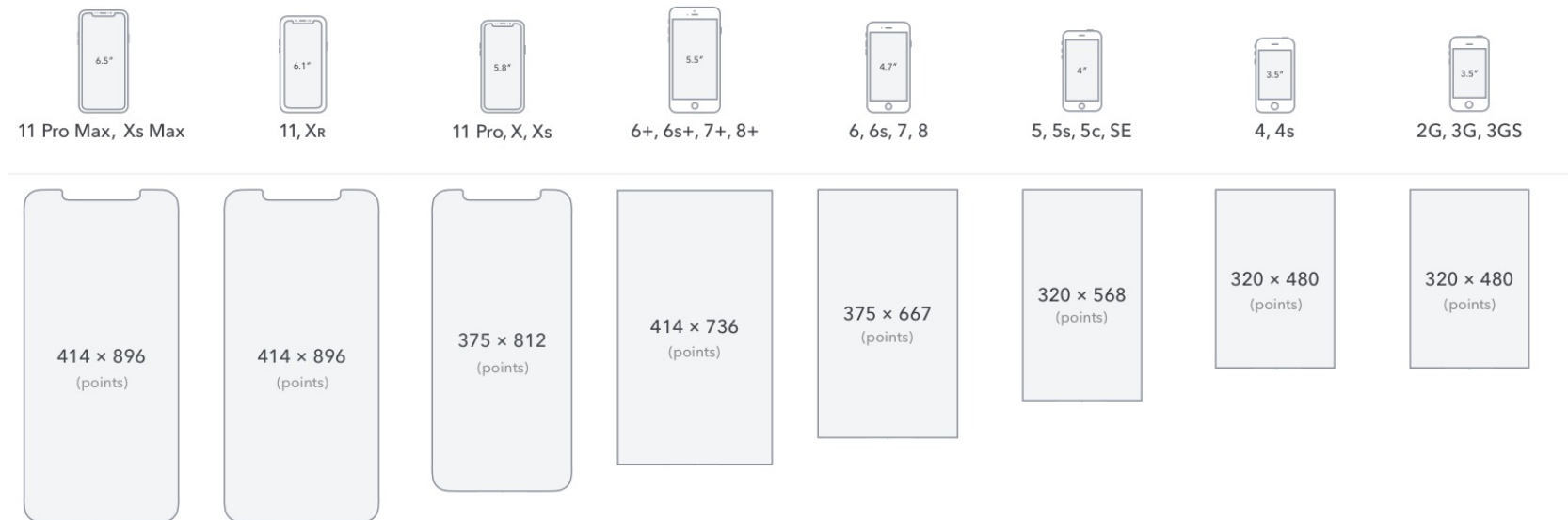


2G, 3G, 3GS



iPhone képernyők mostanában

- Nem annyira tagolt, mint az Android, de egyre több kijelzőméret
- Megjelenik az iPad is, mint célplatform



Felhasználói felület változások, amire érdemes reagálni

- Külső változások (external)
 - > A készüléket elforgatják
 - > Megjelenik/eltűnik az aktív hívást jelző sáv
 - > iPaden megnyitásra vagy bezárásra kerül a Split View
- Belső változások (internal)
 - > Más nyelvű lesz az alkalmazás (internationalization és localization)
 - > Dynamic Type használata (betűk méretének rendszer szintű megnövelése)

Hogyan lehet reagálni a változásokra

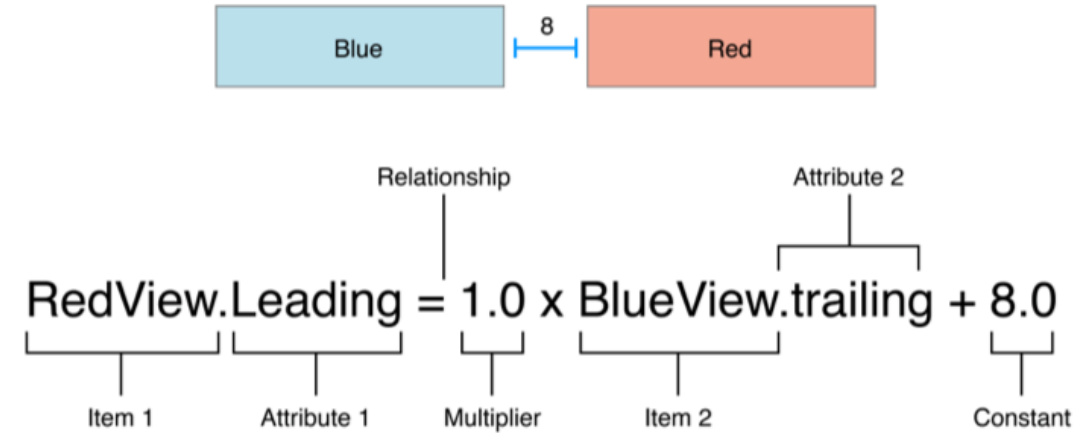
- Kézzel programozva
 - > A UI minden elemének kézzel kiszámolni a méretét és a pozícióját (*x, y, width, height*)
 - > Változások esetén újraszámolni...
 - > Lehetséges, de még egyszerű felületek esetén is sok munka lehet
- Autoresizing masks használata
 - > Könnyítés az előzőhöz képest, a superview frame-jének változására tud reagálni (pl. *flexibleWidth*)
 - > Még mindig sokat kell kódolni
- AutoLayout
- SwiftUI

Auto Layout

- **Layout:** nézetek méretének és pozíciójának meghatározása
- Az **Auto Layout** egy **deklaratív** módja a nézetek méretének és pozíciójának meghatározásának
 - > A felület kinézetére vonatkozó elvárásokat (kényszereket) fogalmazzuk meg
 - > A rendszer a kényszerek alapján futási időben kiszámítja a nézetek pozícióját és méretét
 - > Különböző felbontások/méretek esetén is a kényszereknek megfelelő elrendezést kapunk
 - > Reagál külső és belső változásokra is

Auto Layout - Példa

Kényszer - Constraint



- Reláció a nézetek attribútumai között
- A kényszerek összessége egy **lineáris egyenletrendszer** alkot
- Az Auto Layout feladata az egyenletrendszer megoldása
 - > Vagy legalábbis az "optimális" megoldás közelítése
 - > Lineáris programozást használ – nem kell ismerni, hogy használni tudjuk
 - > Az ismeretlenek a nézetek pozíció és méretei

Kényszer - Constraint

- Lehetséges **relációk** (Relationship)
 - > LessThanOrEqualTo (\leq)
 - > Equal ($=$)
 - > GreaterThanOrEqualTo (\geq)
 - > pl. `view1.width` *legalább akkora* legyen mint `view2.width`
- A kényszereknek **prioritása** is van
 - > 1-1000 közötti érték
 - > Ha ütközés van, a nagyobb prioritású kényszer jut érvényre
 - > Az 1000 prioritású kényszerek kötelezően kielégítendőők
 - > Amikor csak lehet használjuk az előre definiált értékeket!

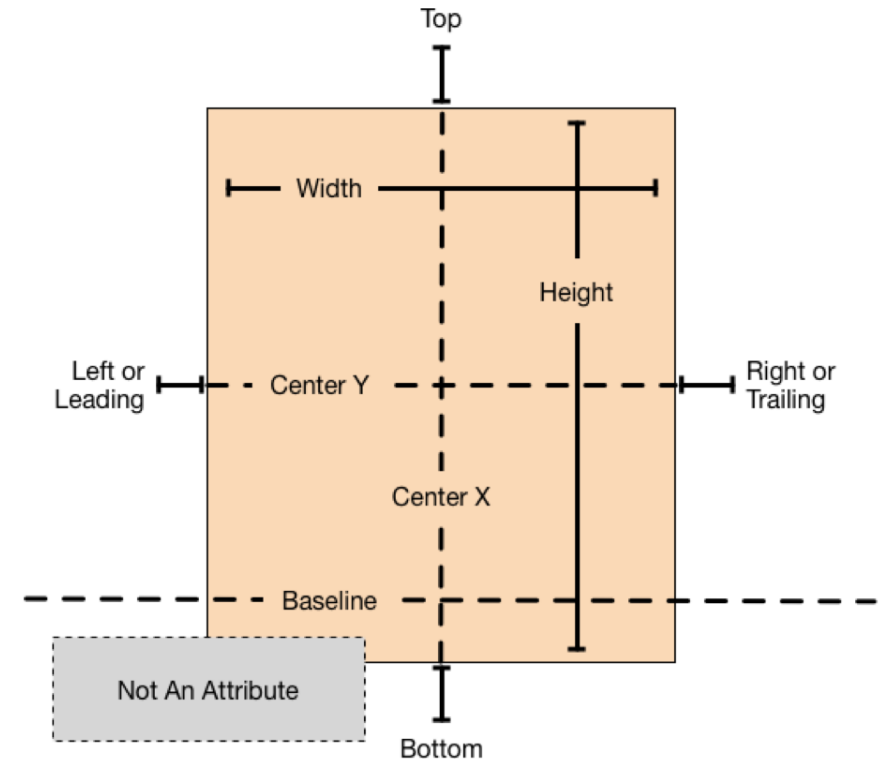
Kényszer - Constraint

- Problémák
 - > **Túl kevés kényszer:**
nincs egyértelmű megoldás (*ambiguous constraints*)
 - > **Egymásnak ellentmondó kényszerek:**
nincs olyan megoldás ami minden kényszert kielégítene (*unsatisfiable/conflicting constraints*)
 - > A rendszer ettől függetlenül megpróbál tenni valamit, csak nem biztos, hogy azt kapjuk, amit szettünk volna

Kényszerekben használható attributumok

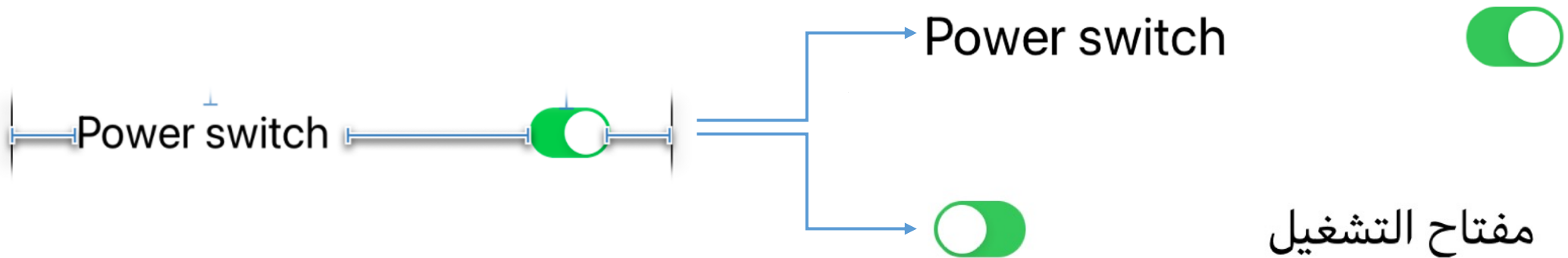
- **NSLayoutAttribute** **enum** konstansai

- > top, topMargin
- > bottom, bottomMargin
- > leading, leadingMargin
- > trailing, trailingMargin
- > width
- > height
- > centerX, centerXWithinMargins
- > centerY, centerYWithinMargins
- > ...
- > notAnAttribute – üres placeholder a konstansoknak

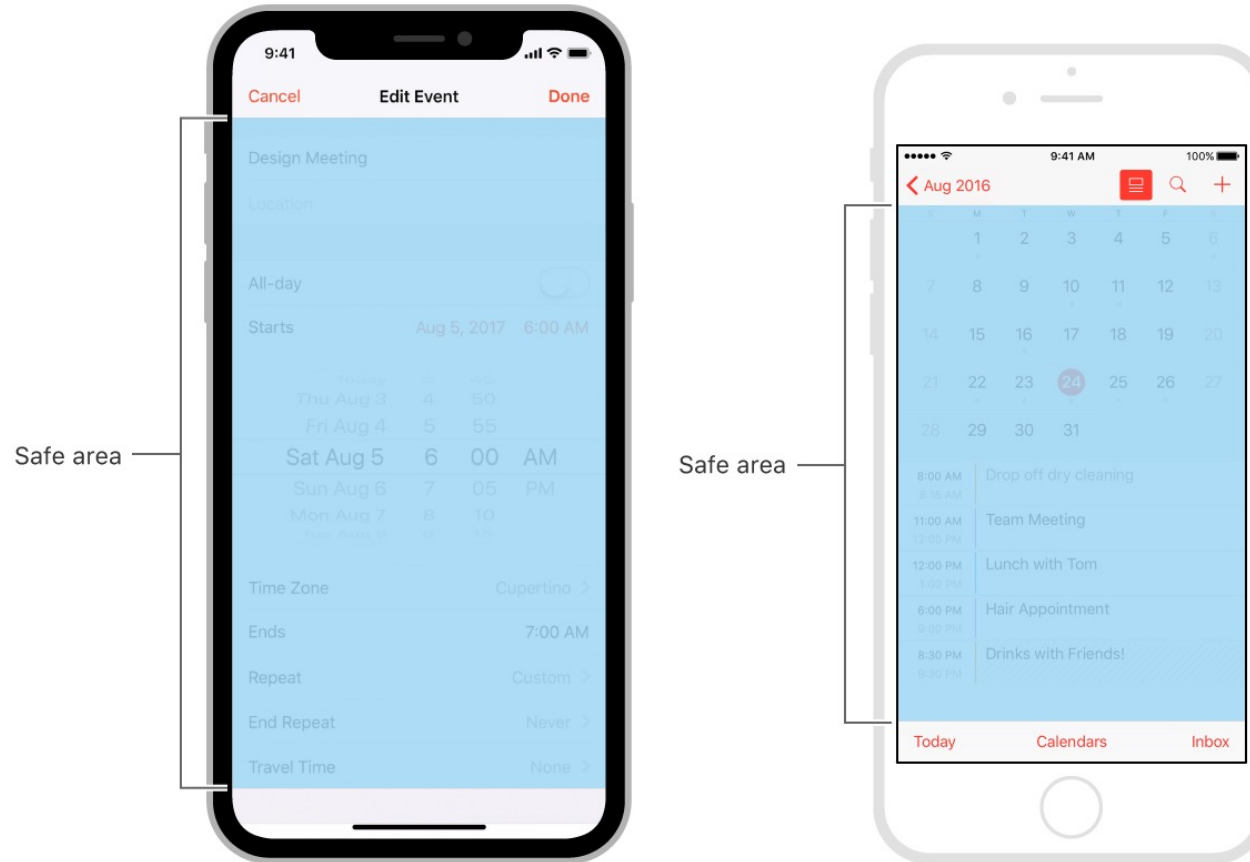


Leading – trailing vs. left – right

- Nem minden nyelven írnak balról jobbra
 - > Ha az angol szöveget a bal/jobbs oldalakhoz kötjük, akkor rosszul fog megjelenni pl.: arab nyelven
 - > A bal és jobb oldalak helyett kössük a nézetet az belépő és kilépő oldalhoz
 - A lokalizáció alapján az Auto Layout automatikusan átrendezi a nézeteket



Safe Area Layout Guide



Safe Area Layout Guide

- Kényszerekben használható láthatatlan téglalap alapú terület, mely mindig a nézet "hasznos területének" határait jelöli
 - > A rendszer automatikusan beállítja/létrehozza
- Az ezeken kívül eső terület vagy nem látszik (pl.: notch alatti terület) vagy csak áttetsző módon (pl.: Navigation Bar alatt)
- Elsősorban a ViewController gyökér nézetnél mérvadó
 - > A gyermek nézeteknél megegyezik a nézet széleivel. Kivéve, ha kilógnak a Safe Area-ból
 - > **Érdemes** ehhez igazítanunk és nem a tényleges szélekhez

Intrinsic Content Size

- ~ *Tartalom mérete*
- Bizonyos felhasználói felületi elemek tudják, hogy a bennük lévő tartalomnak mekkora helyre van szüksége: (**intrinsic content size**)
 - > pl. UILabel, UIButton, UIImageView (amennyiben nem üres), stb.
- Ezeknél a nézeteknél, ha nem adunk meg a méretükre vonatkozó kényszereket, akkor az intrinsic content size méretben fognak megjelenni
 - > A háttérben létrejön néhány alacsony prioritású implicit kényszer, melyek rögzítik a nézet méretét
- Felüldefiniálható rögzített értékekkel

Content Compression Resistance & Content Hugging

- Ha egy nézetnek van intrinsic content size-za, akkor tartozik hozzá 2 pár (összesen 4: 2 horizontal + 2 vertical) automatikusan létrehozott implicit kényszer, melyek a tartalom mérethez kötik a nézet méretét
- Ezeknek a kényszer pároknak a prioritását külön lehet állítani:
 - > **Content Hugging Priority:** annak a két kényszernek a prioritása, amik rögzítik, hogy a nézet nem lehet nagyobb mint az intrinsic content size
 - > **Content Compression Resistance Priority:** annak a két kényszernek a prioritása, amik rögzítik, hogy a nézet nem lehet kisebb mint az intrinsic content size
 - > Akkor van értelme, ha nem rögzítjük külön a nézet szélességét/magasságát!

Kényszerek létrehozása

- Interface Builder
 - > A leggyakrabban használt kényszer típusok felvehetők
 - > A kényszerekre Outletekkel hivatkozhatunk a kódból
 - Futás közben dinamikusan átírhatók a kényszerek egyes értékei, akár animálhatók is
- Kódból
 - > Olyan kényszerek is felvehetők, amik IB-ben nem
 - Tetszőleges nézet layout property-je összeköthető tetszőleges másik nézetével

IB-ben létrehozható kényszerek

- Nézeten „belüli” – konstans
 - > Width
 - > Height
 - > Aspect Ratio (width és height aránya)
- Nézet és szülő közötti
 - > Leading/Trailing/Top/Bottom Space to Superview
 - > Horizontal/Vertical Center in Container
- Két nézet közötti
 - > Width/Height Equality
 - > Aspect Ratio
 - > Align Leading/Trailing/Top/Bottom Edges
 - > Align Horizontal/Vertical Centers
 - > Align Baselines

Kényszer hibák az Interface Builderben

- Piros: hiba a kényszerek rendszerében
 - > Kényszerek hiányoznak (nem egyértelmű egy nézet mérete/pozíciója)
 - > Kényszerek egymásnak ellentmondanak
- Sárga: az Interface Builder által mutatott állapot nem egyezik meg a kényszerek által meghatározottal
 - > Nem feltétlenül hiba, csak épp mást fogunk kapni futás közben
 - > Ha magától nem frissülne, akkor az Update Frames gombbal lehet "kijavítani"

NSLayoutConstraint

- Minden kényszer egy NSLayoutConstraint típusú objektum
- A kényszerek paramétereit futási időben megváltoztathatjuk, pl.:
 - > A kényszer egyenletének konstans tényezőjét módosítjuk:
`buttonBottomConstraint.constant = 100`
 - > A kényszer prioritását módosítjuk:
`buttonBottomConstraint.priority = 500`
- Kényszerek módosítása után mindig értesítsük az Auto Layout motort a gyökérnézet egy metódusának meghívásával:
 - > `setNeedsLayout()` (ez a preferált metódus, mert "kötegelve" végzi el a felület frissítését, ha több kérés érkezik adott időn belül)
 - > `layoutIfNeeded()` (azonnal frissíti a felületet, animációkban használjuk)

Auto Layout - frame/bounds/center

- Auto Layout használata esetén, ne módosítsuk a nézetek frame/bounds/center property-jeit!
 - > Ezek értékének meghatározása már az Auto Layout motor feladata
 - > Csak a kényszereket módosítsuk!
- A kódból létrehozott nézetekhez tartozik néhány rejtett kényszer, melyek rögzítik a pozícióját/méretét a nézet frame/center/bounds property-jeihez
 - > Amíg nem adunk egyetlen kényszert sem a nézethez, használhatjuk a frame/center/bounds-ot!
 - > Ha egyedi kényszereket szeretnénk, ki kell kapcsolni az automatikusan létrejött kényszereket a `translatesAutoresizingMaskIntoConstraints` property `false`-ra állításával

Kényszerek létrehozása kódból

- Alapvetően három lehetőségünk van (ilyen sorrendben ajánlott a használatuk)

- > `NSLayoutAnchor` (iOS 9+) --> (van típusellenőrzés!)

```
NSLayoutConstraint.activate([pwrSwitch.leadingAnchor.constraint(equalTo:  
    self.view.leadingAnchor)])
```

- > `NSLayoutConstraint` (lehetőség van "hibás" kényszerek létrehozására is, amik csak runtime derülnek ki)

```
NSLayoutConstraint.activate([NSLayoutConstraint(item: pwrSwitch!,  
    attribute: .leading, relatedBy: .equal, toItem: self.view, attribute:  
    .leading, multiplier: 1, constant: 0)])
```

- > Visual Format Language (NE használjuk – rengeteg a hibalehetőség)

- "ASCII art" szerű vizuális nyelv, egyszerre több kényszer is létrejöhet ("[view1]-50.0-[view2]")
 - Az utóbbi években már nem fejlesztik

Scroll View és Auto Layout

- Auto Layout esetén a Scroll View a benne lévő alnézetekből és azok Auto Layout kényszereiből találja ki a tartalom méretét (`contentSize`)
- Alapszabály: a Scroll View-nak képesnek kell lennie kitalálnia a méretét a benne lévő alnézetek méretéből és pozíciójából
- Egy tipikus forgatókönyv
 - > A Scroll View minden oldalához "belülről" kötelezően hozzá kell kapcsolni legalább egy gyereknézetet (Auto Layout kényszerrel)
 - > A gyereknézetek méretét rögzítsük

Scroll View és Auto Layout II.

- Scroll View kényszereket általában két csoportba bontjuk:
 - > Scroll View frame-jének a Scroll View superview-jához való igazítása
 - > A Scroll View belsejében lévő content view(-k) Scroll View-hoz való igazítása
- A Scroll View-hoz két layout guide tartozik
 - > frameLayoutGuide
 - > contentLayoutGuide
- Ha superview-hoz igazítunk, akkor a Scroll View frame-jére vonatkoznak
- Ha a content view-khoz, akkor pedig a contenthez

Auto Layout



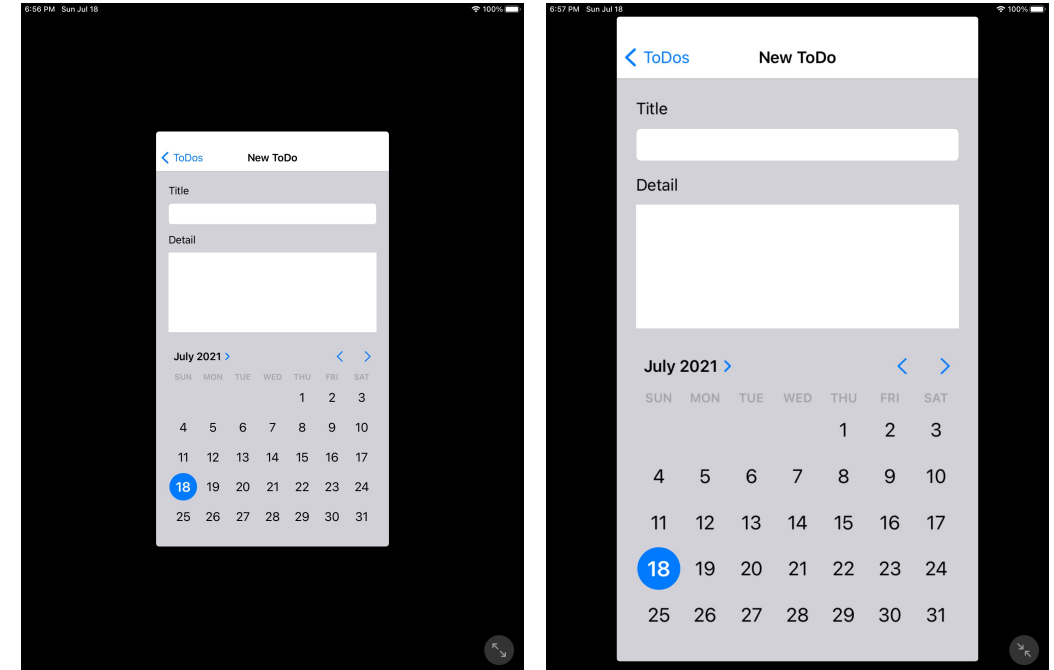
Adaptive Layout

Multiplatform támogatás

- Egy ~iOS alkalmazás nem csak iPhone-on futhat
- iOS alkalmazás típusok
 - > **iPhone only** (iPaden is elindul, de csak egy iPhone felbontású "ablakban")
 - > **iPad only**
 - > **Universal***: mind iPhone-on, mind iPaden a teljes képernyőt (felbontást) kihasználó alkalmazás
 - *Már nem külön kategória
 - Vagy külön UI az alkalmazás jeleneteihez iPhone-ra és iPadre (pl. külön Storyboard iPhone-ra és iPadre)
 - Vagy egységes Storyboard (**Unified Storyboards**), mely egy Storyboardban lefedi a különböző felbontású készülékeket
 - > **Mac**: iPad-et is kell támogatni, de lehet Mac specifikus UI

iPhone alkalmazások iPaden

- iOS 12-ig ha iPad-et nem támogató alkalmazást futtattunk iPaden, akkor egy iPhone 4-es felbontású ablakban jelent meg.
- iOS 12-től kezdve egy iPhone 6-os felbontású ablakban jelenik meg
- Van rá lehetőség, hogy belenagyítsunk!

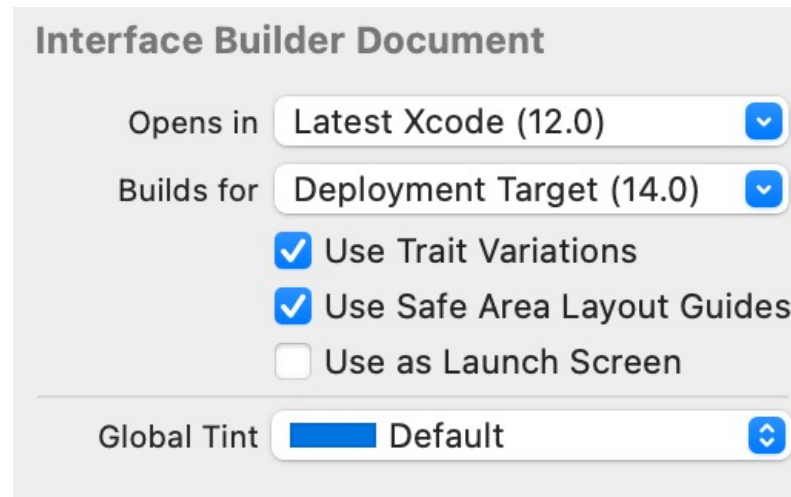


Adaptive Layout

- **Adaptive Layout:** adaptív az a felhasználói felület, amely a lehető legjobban kihasználja a rendelkezésre álló helyet, továbbá, ami tartalmát képes úgy igazítani, hogy az minden iOS eszközön megfelelően nézzen ki
- Ehhez a Trait Collectionöket kell használni
- Kijelzőmérettől, készüléktípustól és egyéb paraméterektől/jellemzőktől függően megadhatók:
 - > Auto Layout kényszerek (pl. más kényszerek iPaden, mint iPhone-on)
 - > Nézetek megjelenítése/eltűntetése (pl. néhány nézet csak iPaden látszik)
 - > Eltérő képek (pl. iPhone-on kisebb, iPad-en nagyobb)

Adaptivitás ki- és bekapcsolása

- Storyboardra lehet ki- és bekapcsolni
 - > File Inspector -> **Use Trait Variations**



Trait Collection

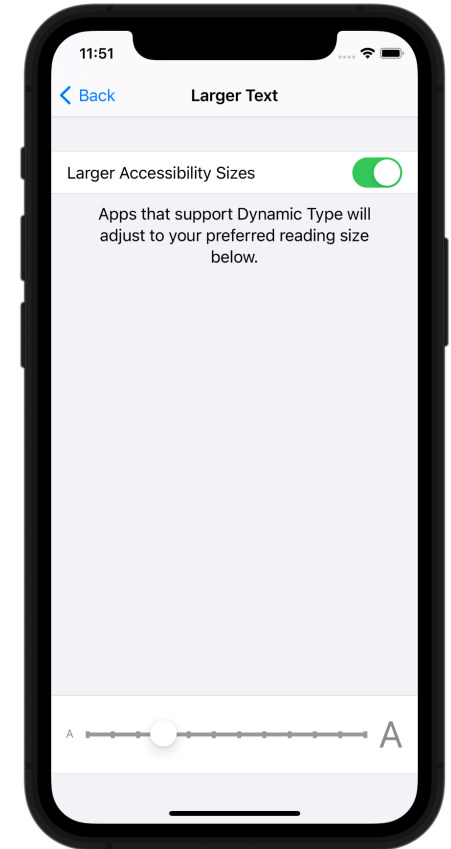
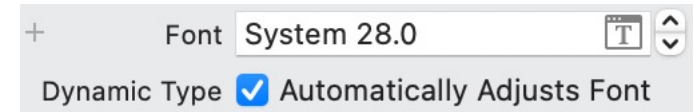
- Egy készülék összes méret-, tájolás-, megjelenítésspecifikus tulajdonságát egy **Trait Collection** objektum tartalmazza
- A UIScreen, UIWindow, UIView és a UIViewController implementálja a UITraitEnvironment prototolt:
 - > A **traitCollection** property-jükkel kérdezhető le
 - > Csak override-olni kell a **traitCollectionDidChange** metódust, hogy értesüljünk a változásról
- A Trait Collection szülőről gyerekre öröklődik
 - > Az értéke felüldefiniálható egy adott nézetben
- Megváltoztathatjuk a nézet viselkedését (pl. nézzen ki mindig úgy, mintha v:Compact, h:Compact *Size Class* értékek lennének aktívak)

Trait Collection elemei

- A Trait Collection a következő propertyket tartalmazza:
 - > `horizontal/verticalSizeClass`: {compact, regular} méretosztályok
 - > `userInterfaceStyle`: {light, dark} sötét/világos stílus
 - > `userInterfaceIdiom`: {phone, pad, tv, carplay, mac} milyen készüléken fut
 - > `displayScale`: {1.0, 2.0, 3.0} = pixel per point @1x, @2x, @3x
 - > `layoutDirection`: {leftToRight, rightToLeft} Szöveges tartalom iránya
 - > `preferredContentSizeCategory`: Szöveges tartalom dinamikus mérete
 - > stb.
- A legtöbb property értéke lehet `unspecified` is

Dinamikus szövegméret

- A Settingsben beállítható, hogy a „dinamikus szövegek” OS szinten mekkorák legyenek
- Alapból 7 méret, Accessibility még 5 nagyobbát engedélyez
- Alapelvek:
 - > Ha a szöveg dinamikus lehet, legyen is dinamikus
 - > A képernyő szélességének teljes kihasználása
 - > Ne legyen szöveg csonkítás
 - > Ne csak a szöveget, hanem a hozzá tartozó grafikus tartalmat is dinamikusan méretezzük
- A szövegméret futásidőben tud változni
- Saját fontokat is támogat
- Teszteléshez Interface Builder-ben az Environment Overrides ablakban szimulátorban futásidőben változtathatjuk a szöveg méretét

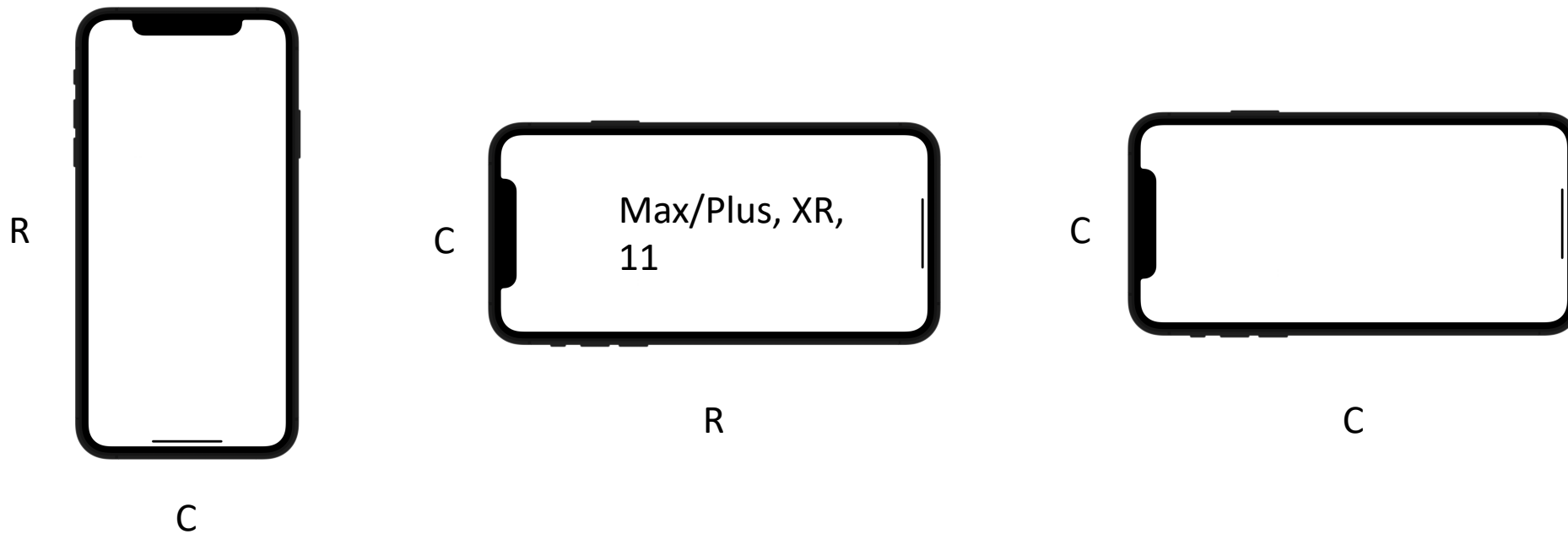


Méretosztályok (Size Classes)

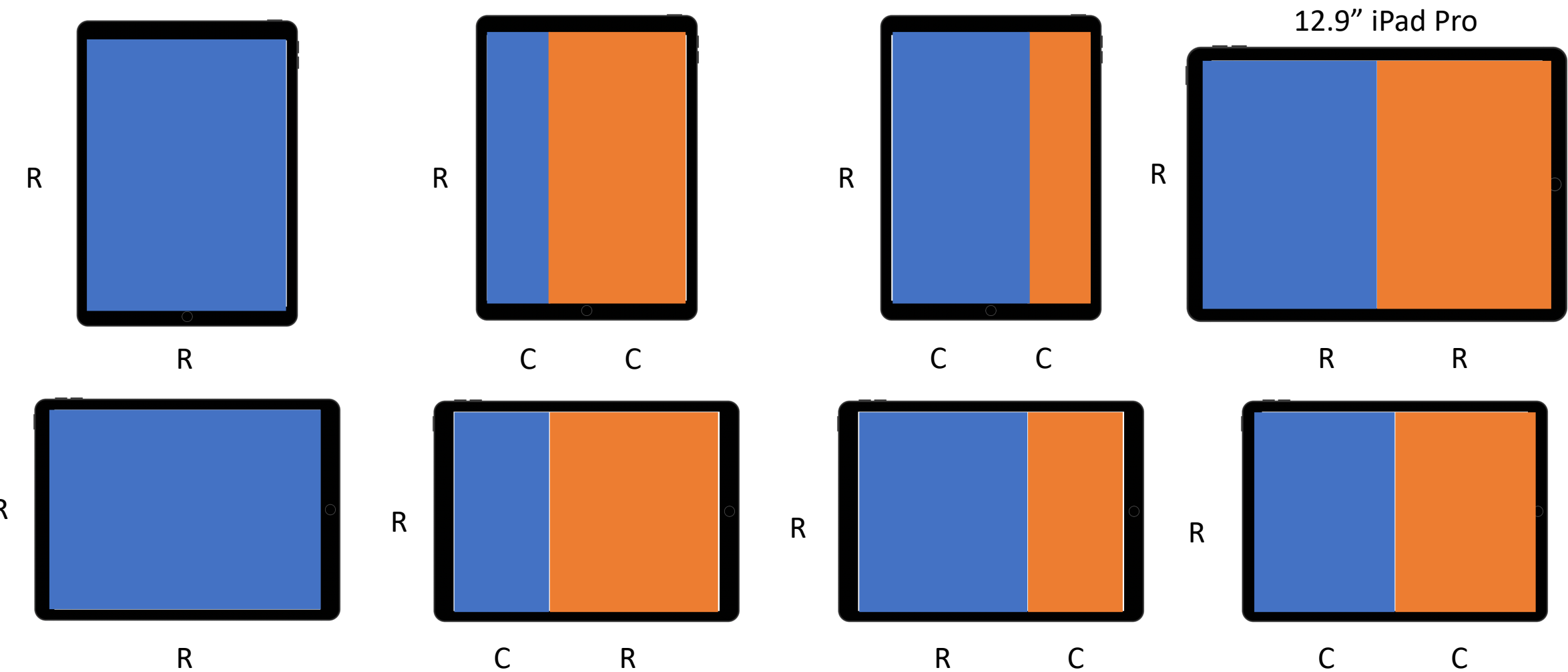
- A méretosztály meghatározza, hogy az adott nézet "mekkora tartalmat" tud megjeleníteni
- Két méretosztály property létezik:
 - > Horizontal (h)
 - > Vertical (w)
- A méretosztály property-knek két lehetséges értéke:
 - > Compact (C)
 - > Regular (R)

Eszközök és méretosztályok

- Minden eszköztípus (iPad/iPhone) és készülék tájolás (portrait/landscape) párhoz tartozik egy-egy vertikális és horizontális méretosztály érték:



Eszközök és méretosztályok II.

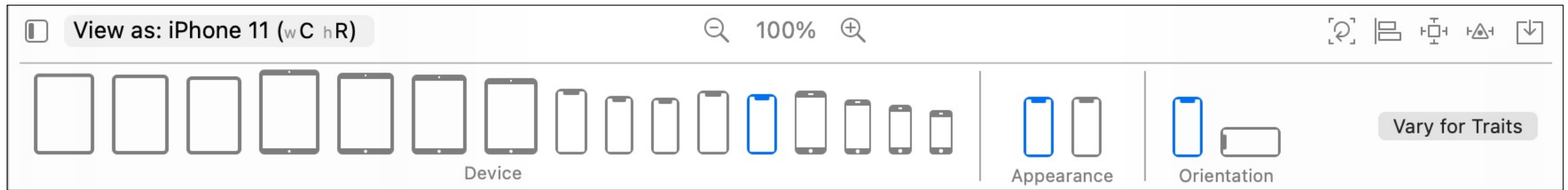


Adaptív tervezés

- Az **adaptív tervezés** lényege, hogy a felhasználói felületet méretosztályokat figyelembe véve tervezzük meg
- A méretosztályok elfedik az eszközspecifikus tulajdonságokat
 - > Méretosztályokhoz tervezzünk és NE készülék típusokhoz!
- Rugalmas és hatékony koncepció
 - > Az összes eszköz lefedhető néhány méretosztállyal
 - > Csak a méretosztályok közötti "különbségeket" kell leírni
 - > Még meg sem jelent eszközök automatikus támogatása
- Vannak esetek, amikor kevésnek bizonyulhat

Méretosztályok az Interface Builder-ben

- Méretosztályonként külön-külön megadható (Vary for Traits):
 - > Nézetek létrehozása/eltüntetése
 - > Kényszerek beállítása/eltávolítása
 - > Kényszerek paramétereinek megadása
 - > Eltérő betűtípus beállítások
 - > Eltérő színek beállítása



Egyedi nézetek

Core Graphics

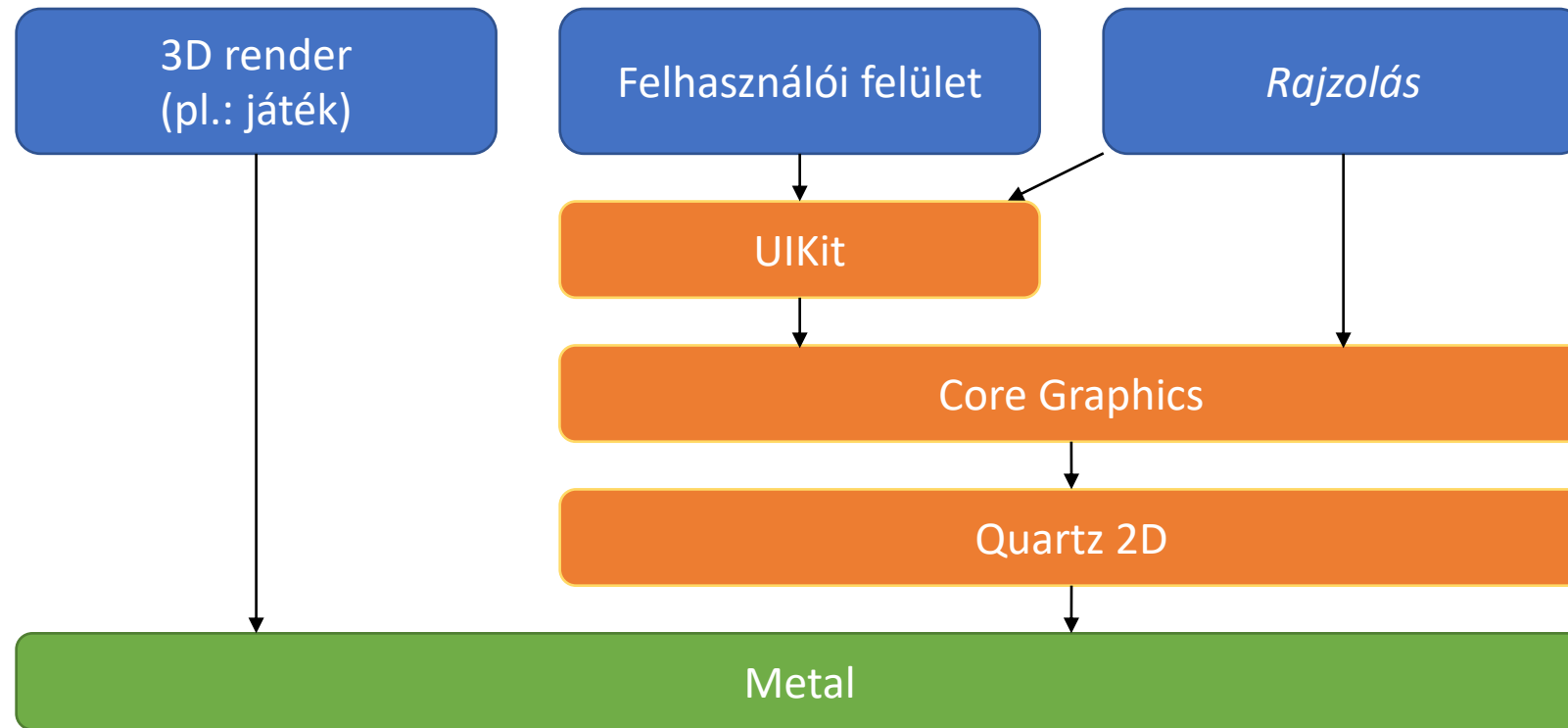


Automatizálási és
Alkalmazott
Informatikai Tanszék

UIView

- Mindennek (nagyjából), ami az alkalmazásunkban megjelenik, a UIView-ból kell leszármaznia
 - > Ha egyedi nézetet szeretnénk készíteni, nem lehet megkerülni
- Mitől lesz egyedi?
 - > Megjelenés/kinézet
 - > Érintések kezelése/viselkedés

Megjénítés iOS-en

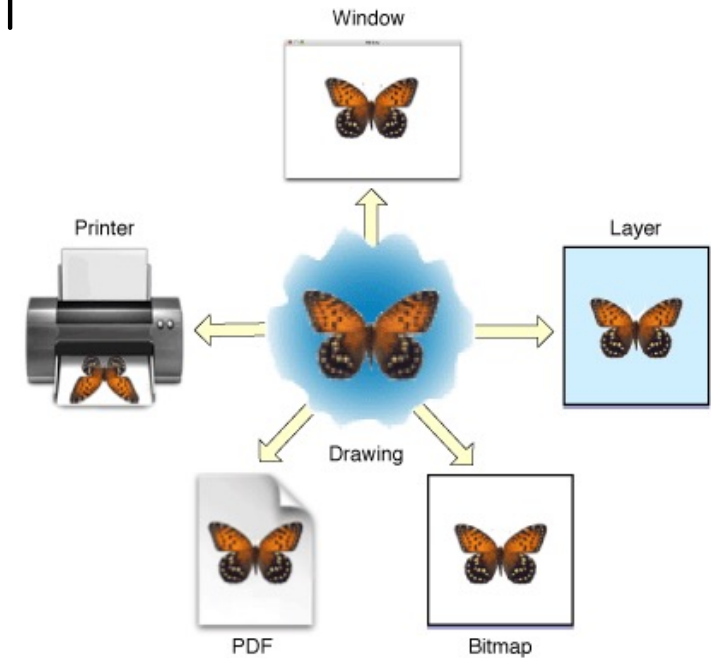


CG – Core Graphics

- Framework alacsony szintű 2D rendereléshez
 - > Saját CG típusok (korábban már foglalkoztunk velük), és speciális objektumok (CGContext, CGPath)
 - > Transzformációk, színátmenetek, maszkok stb.
- Rajta keresztül érjük el a Quartz 2D rajzoló „motort”
 - > A tartalmat már a Quartz rajzolja ki a kijelzőre, pontosabban az adott context-re.
 - > A context gyakorlatilag egy területet reprezentál, ahová a Quartz rajzolni fog
- A UIKit is a Core Graphics szolgáltatásait használja
 - > Minden UIView-hoz saját context tartozik

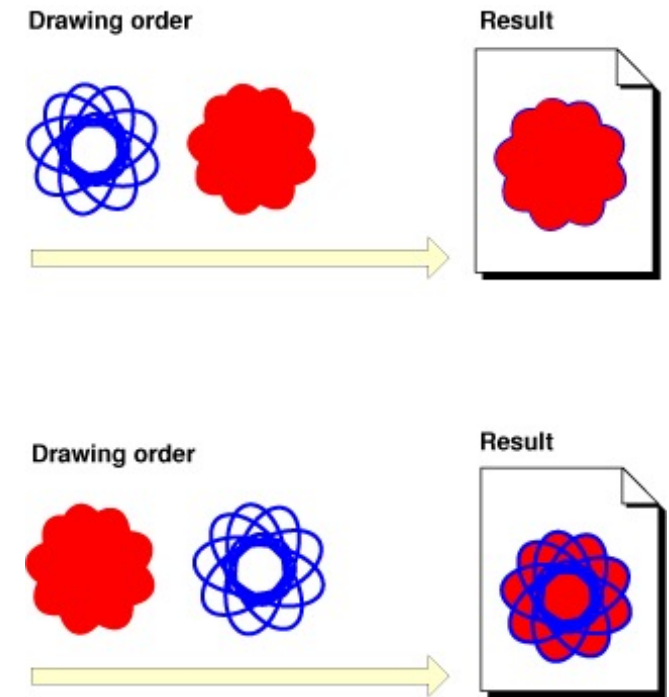
Quartz 2D

- 2D Rajzoló motor iOS, tvOS, iPadOS, illetve OS X platformokra
- Alacsony szintű interfész, „lightweight”
 - > Összetettebb megjelenítéshez relatíve sokat kell kódolni
- Elfedi az alatta lévő rétegeket (Metal)
- Nem csak képernyőre képes rajzolni:
 - > Kép
 - > Pdf
 - > Nyomtató
 - > Réteg



Quartz 2D - vászon

- A Quartz a „painter’s model”-t követi:
 - > Amint kiadunk egy festési (rajzolósi) utasítást, a festék egy új réteggként rákerül az adott vászonra (context-re).
 - > A már megrajzolt tartalom nem módosítható, legfeljebb felülírható. (Pont mint a valódi festés esetén)
 - > Hasonló a Windows WinAPI és egyéb, szintén alacsony szintű rajzoló rendszerekhez



Rajzolás Core Graphics-szal

- A rajzoláshoz mindig szükség van egy context-re
 - > Létre is lehet hozni, pl. bitkép, vagy PDF generálásához, de a UIView-ban elég lekérdezni az aktuálisat:
`let context = UIGraphicsGetCurrentContext()!`
- A context-en érdemes beállítani a kitöltőszínt és a körvonal színét, formáját és egyéb tulajdonságait:
 - > Kitöltőszín:
`context.setFillColor(UIColor.systemGreen.cgColor)`
 - > Körvonal színe:
`context.setStrokeColor(UIColor.systemRed.cgColor)`
 - > Körvonal vastagsága:
`context.setLineWidth(6)`

Rajzolás Core Graphics-szal II.

- Meg kell adni a rajzolási (kitöltés, körvonal, színátmenet, stb.) műveleteket, ami közvetlenül a contextre rajzolnak. Pl.:
 - > Téglalap - körvonal
`context.stroke(CGRect(x: 0, y: 0, width: 10, height: 20))`
 - > Ellipszis - körvonal
`context.strokeEllipse(in: CGRect(x: 0, y: 0, width: 10, height: 20))`
 - > Ellipszis - kitöltés
`context.fillEllipse(in: CGRect(x: 0, y: 0, width: 10, height: 20))`
 - > CGPath - körvonal
`context.strokePath()`
- A rajzolás mindig az parancs kiadásakor érvényes context beállításokkal történik: Ha pl. egyszer beállítjuk a körvonal színét kékre, onnantól minden utána kirajzolt objektum kék körvonalú lesz, amíg át nem állítjuk.

CGPath

- Speciális objektum, ami összefoghat több primitív elemet:
 - > Vonal
 - > Téglalap
 - > Ellipszis
 - > Körív
 - > Bézier-görbe
 - > CGPath
- Könnyebben lehet kezelni vele az egységesen megjelenő objektumokat
- A UIView context-nek alapból van egy path-e

UIView – egyedi megjelenés

- Ahhoz, hogy a megjelenés egyedi legyen felül kell definiálni a rajzolást végző metódust `draw()` a UIView leszármazottban:

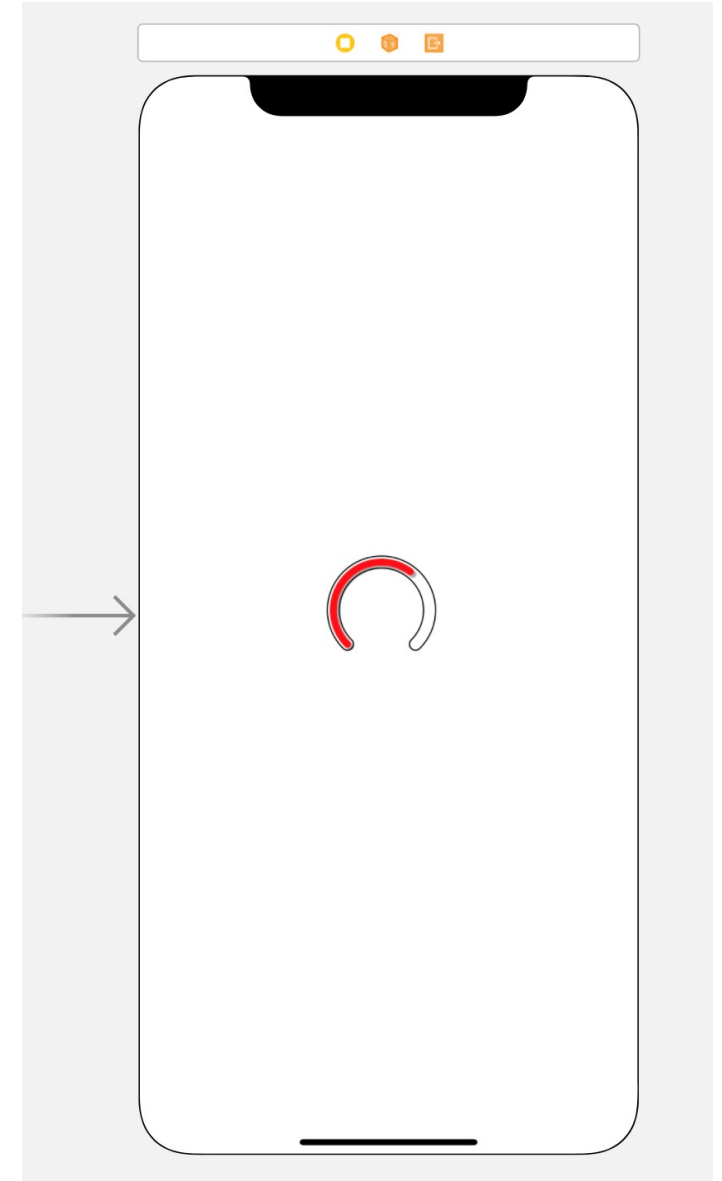
```
// Only override draw() if you perform custom drawing.  
// An empty implementation adversely affects performance during  
// animation.  
override func draw(_ rect: CGRect) {  
    // Drawing code  
}
```

- A `rect` mondja meg, hogy a nézetünk melyik részét kell újra kirajzolni
 - > Első rajzoláskor az egész nézetet ki kell rajzolni

@IBDesignable

- Az egyedi rajzolás tesztelése alapvetően nehézkes, hiszen el kell indítani az alkalmazást, hogy láthassuk az eredményt
- Ha a UIView leszármazott fejlécénél elhelyezzük az **@IBDesignable** kulcsszót, onnantól kezdve az Interface Builderben is látható lesz a nézet rajzolt tartalma

```
@IBDesignable  
class MyView: UIView {
```

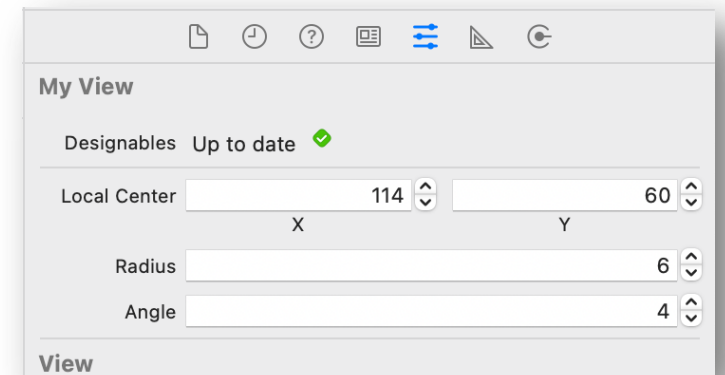


@IBInspectable

- Ha szeretnénk az testre szabni a megjelenését az egyedi nézetünknek, alapesetben csak a forráskódon keresztül van rá lehetőség, ami gyakran nem elég.
- Ha az Interface Builderben is szeretnénk a UIView leszármazottunk property-jeit elérni és módosítani, akkor az @IBInspectable kulcsszót kell előtte elhelyezni
 - > Fontos, hogy explicit meg kell adni a típust, a type inference nem elég!

```
class MyView: UIView {
```

```
    @IBInspectable var localCenter:CGPoint = CGPoint(x: 0, y: 0)  
    @IBInspectable var radius:CGFloat = CGFloat(40)  
    @IBInspectable var color:UIColor = UIColor.systemRed
```



UIView init

- A UIView-nak két inicializáló metódusa is van:
- initWithFrame – Akkor hívódik meg, amikor kódból hozzuk létre a nézetet

```
override init(frame: CGRect) {  
    super.init(frame: frame)  
    //Init code  
}
```

- initWithCoder – Akkor hívódik meg, amikor storyboardból vagy XIB-ből hozzuk létre a nézetet

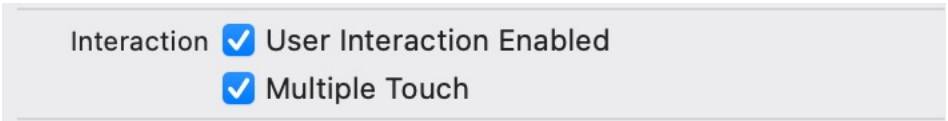
```
required init?(coder: NSCoder) {  
    super.init(coder: coder)  
    //Init code  
}
```

- Felüldefiniálás nélkül is használhatjuk a UIView-t, de ha az initWithFrame-t felüldefiniálnánk, az initWithCoder-t is felül kell.
 - > Ha csak az initWithCoder-t, akkor az initWithFrame-et nem kell.

Érintések iOS-en (és iPadOS-en)

- Minden UIControl vezérlő képes lekezelni az érintéseket
 - > Különben nem tudnánk használni a telefont...
- Az érintések lekezelését egy UIView leszármazottban felül is lehet definiálni
- Alapesetben a UIView egyszerre egy érintést kezel
 - > Ha szeretnénk többet is lekezelni, engedélyezni kell a multi-touch-ot
 - > Az iPhone szimultán 5 érintést képes egyszerre kezelni, az iPad 11-et

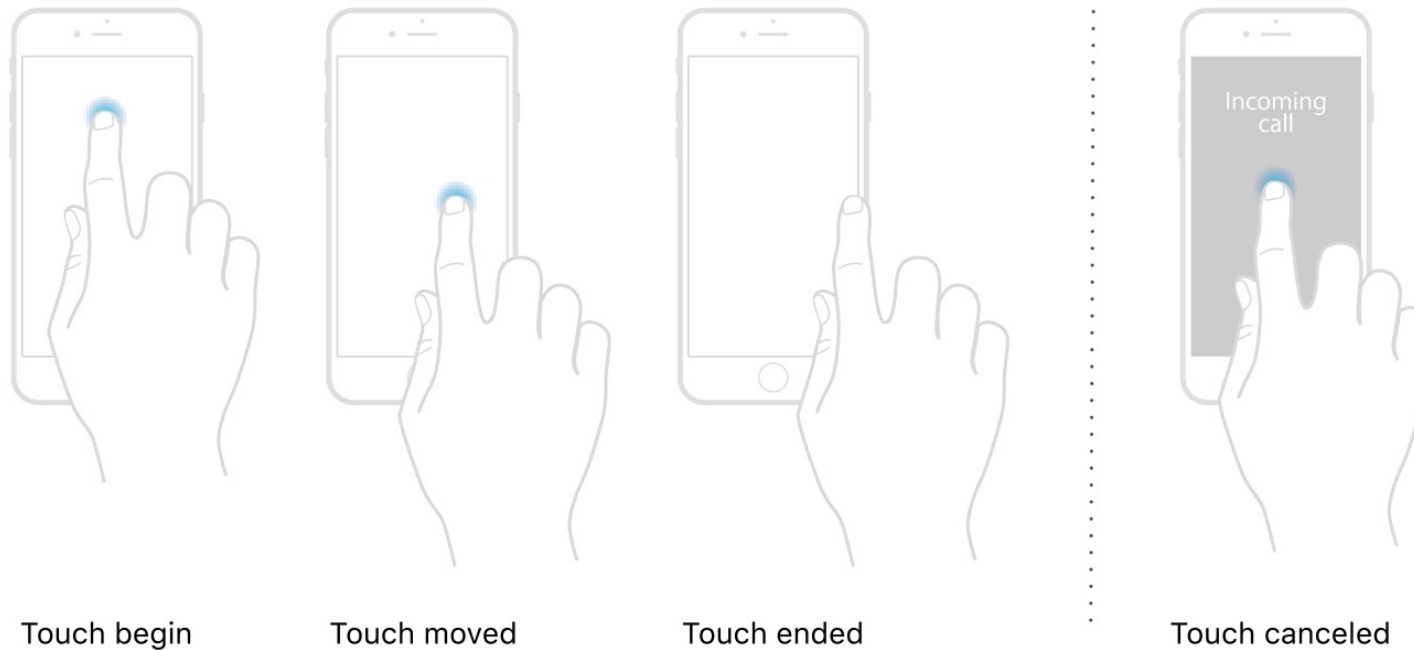
```
self.isMultipleTouchEnabled = true
```



Interaction ☒ User Interaction Enabled
☒ Multiple Touch

UIView érintések

- Amennyiben egy UIView-n érintés történt, azt helyben le is lehet kezelni:

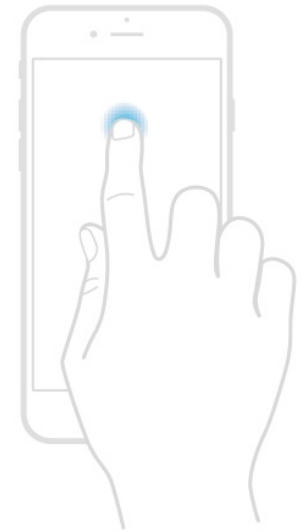


UIView érintések - touchesBegan

- Új érintés a nézeten

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?)
```

- Abban a pillanatban váltódik ki, ha
 - > Valaki megérinti a kijelzőt
 - > A nézet fölött megjelenik a kurzor (iPad)



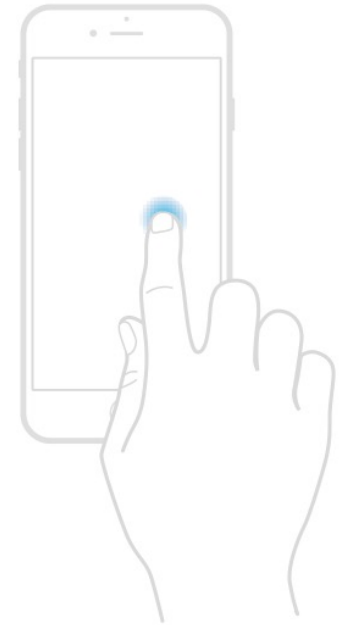
Touch begin

UIView érintések - touchesMoved

- Az érintés elmozdult a nézeten

```
override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?)
```

- Abban a pillanatban váltódik ki, ha
 - > Valaki mozgatja az ujját a kijelzőn, de nem engedi fel
 - > A nézet fölött mozog a kurzor (iPad)



Touch moved

UIView érintések - touchesEnded

- Az érintés megszűnt a nézeten

```
override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?)
```

- Abban a pillanatban váltódik ki, ha
 - > Valaki felengedi az ujját a kijelzőről
 - > A nézet fölül eltűnik a kurzor (iPad)



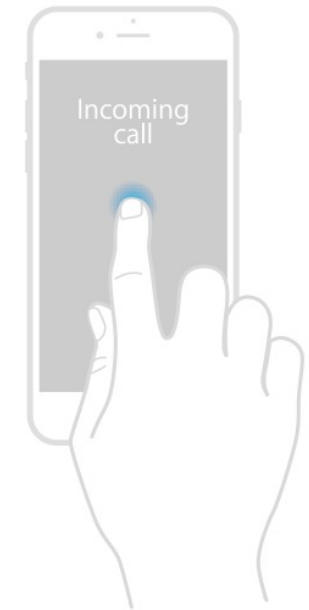
Touch ended

UIView érintések - touchesCanceled

- Az érintés követése megszakadt a nézeten

```
override func touchesCanceled(_ touches: Set<UITouch>, with event: UIEvent?)
```

- Abban a pillanatban váltódik ki, ha az operációs rendszer megszakítja a követést pl.:
 - > Túl sok szimultán érintés történik
 - > Az alkalmazás vagy a nézet már nem tud fogadni érintéseket. (pl.: bejövő hívás, UIAlertView megjelenítés.)



Touch canceled

UIView érintés üzenetek

- Az egyes érintések adatait egy kupacban – **UITouch**:
 - > Nézet – Melyik nézetet értesítette a rendszer
 - > Window – Melyik ablakban történt (iPad)
 - > Típus, időbélyeg és még pár egyéb paraméter.
 - > A érintés helyét, a **location** tagfüggvény adja vissza:

```
func location(in: UIView?)
```

UIView érintés üzenetek II.

- Az üzenetet kiváltó eseményt - UIEvent:
 - > Tartalmazza a érintések adatait
 - > A eseményhez köthető predikált és szorosan kötődő érintéseket
- „Nyers” adatok emiatt az összetettebb gesztúrák felismerése nehézkes
 - > Sokat kell hozzá kódolni