

# iOS alapú szoftverfejlesztés

Dr. Blázovics László

[Blazovics.Laszlo@aut.bme.hu](mailto:Blazovics.Laszlo@aut.bme.hu)

2021. Szeptember 28.



Automatizálási és  
Alkalmazott  
Informatikai Tanszék

# Többnézetes alkalmazások



Automatizálási és  
Alkalmazott  
Informatikai Tanszék

# Eddig egynézetes alkalmazások

- Egy View Controller, melyhez egy nézethierarchia tartozott, minden ebben jelenítettünk meg
- Hogyan lehet az alkalmazáshoz több "képernyőnyi" nézetet rendelni és ezek között váltani, navigálni?

# Többnélzetes app példa I.

- Clock



World Clock View Controller

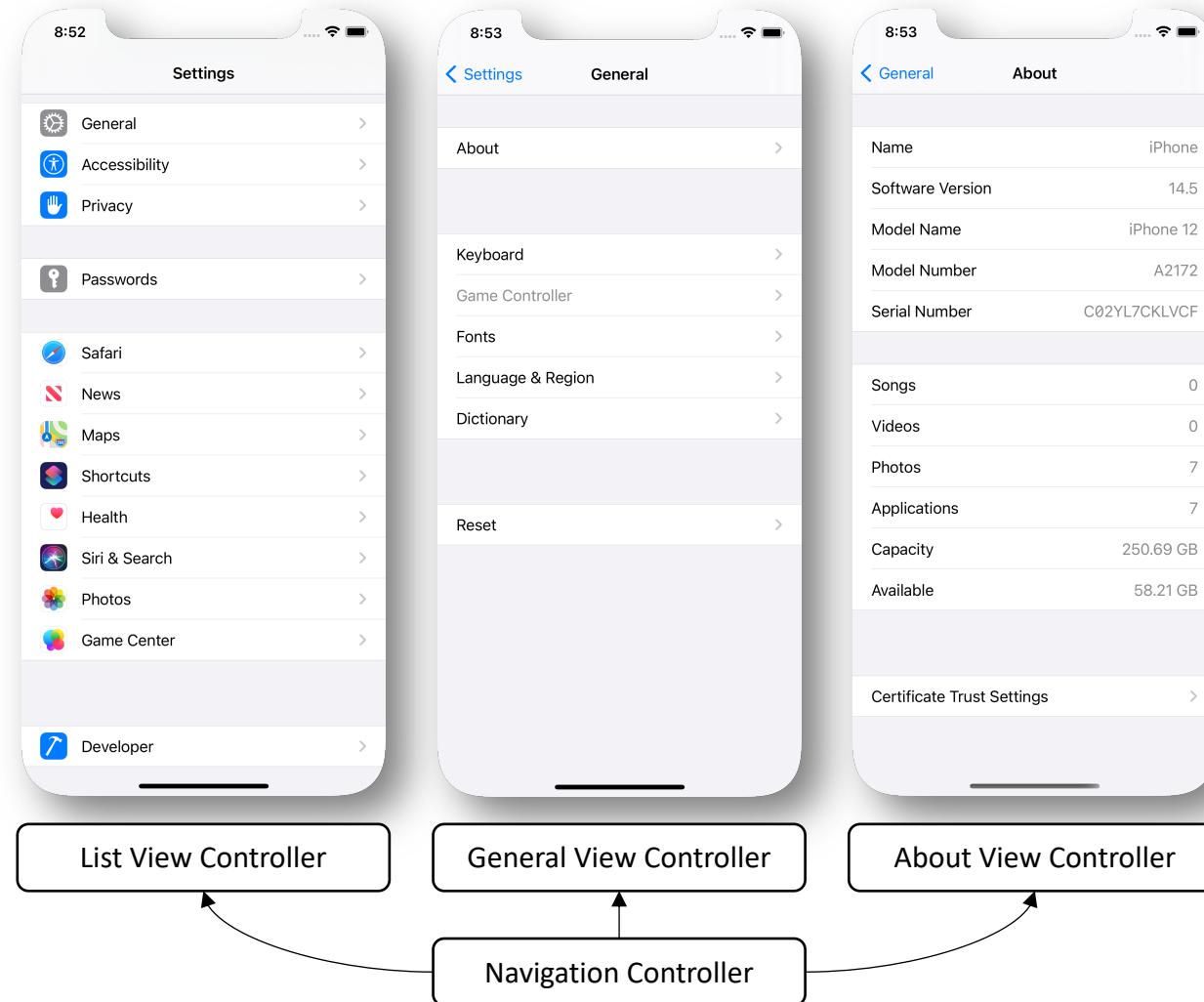
Alarm View Controller

Stopwatch View Controller

Tab Bar Controller

# Többnélzetes app példa II.

- Settings

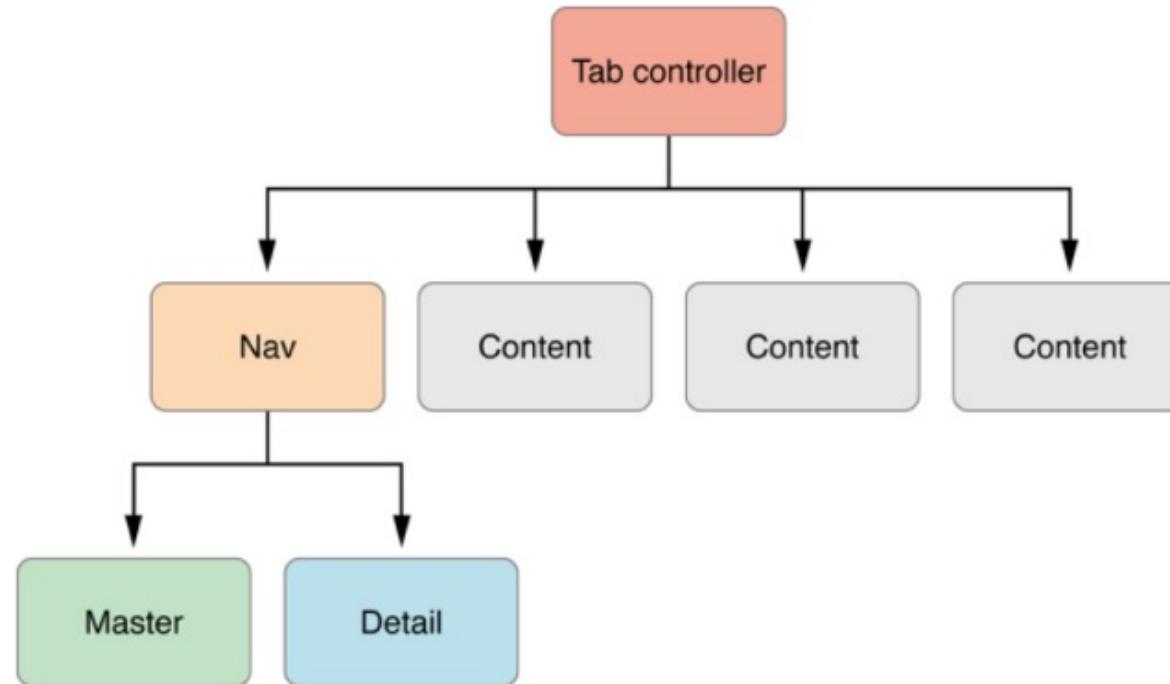


# ViewController típusok

- **Content View Controller:**
  - > Feladata tipikusan egy "képernyőnyi" összetartozó nézetek egy csoportjának felügyelete, a programozó valósítja meg
- **Container View Controller:**
  - > Más ViewController-eket menedzsel, tipikusan navigációhoz vagy komplex, "többképernyős" nézethierarchiákhoz használjuk:
    - UITabBarController
    - UINavigationController
    - UISplitViewController
    - UIPageViewController
  - > A ViewControllerek-nek is lehetnek gyerek ViewController-eik
    - Pl. egy Tab Bar Controller minden különálló tabjához egy külön gyerek View Controller tartozik

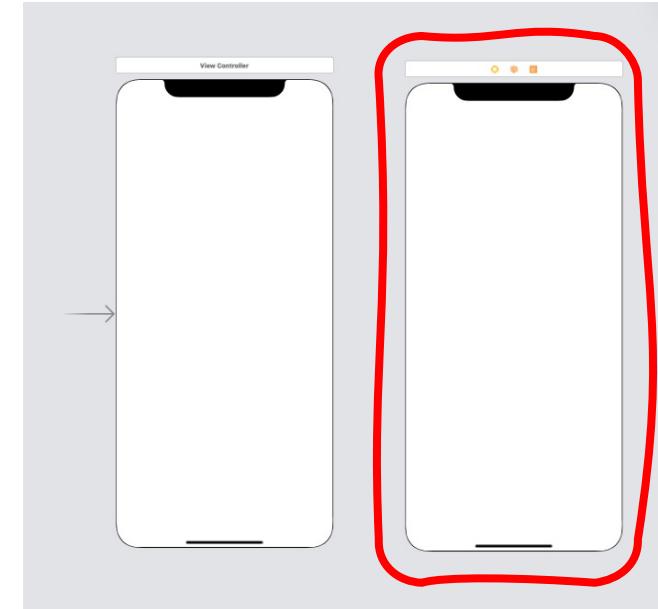
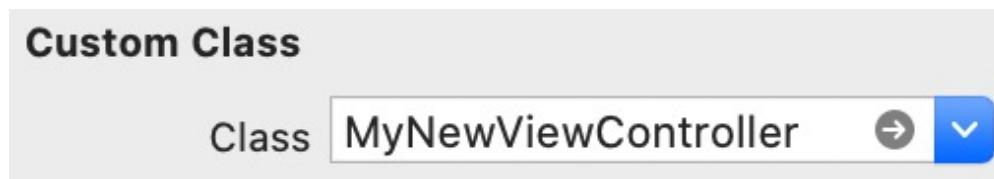
# ViewController hierarchia

- A ViewController-rek is szülő-gyerek hierarchiába rendezhetők
  - > A szülő ViewController gyökér nézetének alnézeteként jelenik meg a gyerek ViewController gyökérnézete



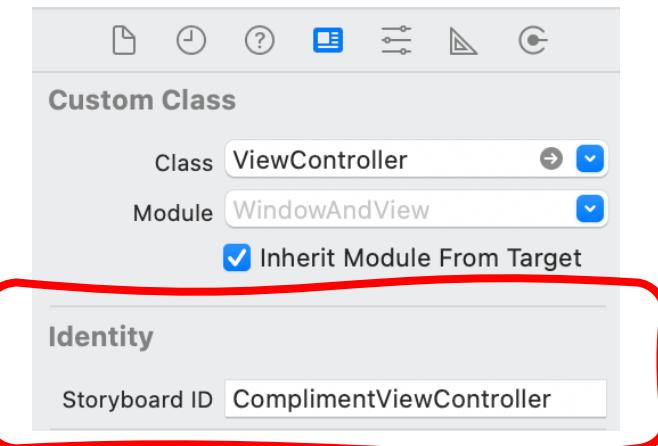
# Új ViewController a storyboardban

1. Létrehozzuk az új ViewController-t (új fájl)
2. A Storyboard-ban hozzáadunk egy új ViewController-t
3. Az Identity inspector-ban beállítjuk a frissen létrehozott ViewController Class-t



# ViewController példányosítása

- A nem Storyboard-ban definiált ViewController-ek (pl. rendszer vagy saját egyedi ViewController-ek, amelyek nézetei kódból épülnek fel) hasonlóan példányosíthatók, mint bármilyen más Swift objektum
- Storyboard-ban definiált ViewController-ek esetében:
  - > A Storyboard Identity inspector-ában hozzá kell rendelni egy Storyboard ID-t, a kódból pedig ezzel az ID-val azonosítjuk a létrehozandó ViewController-t



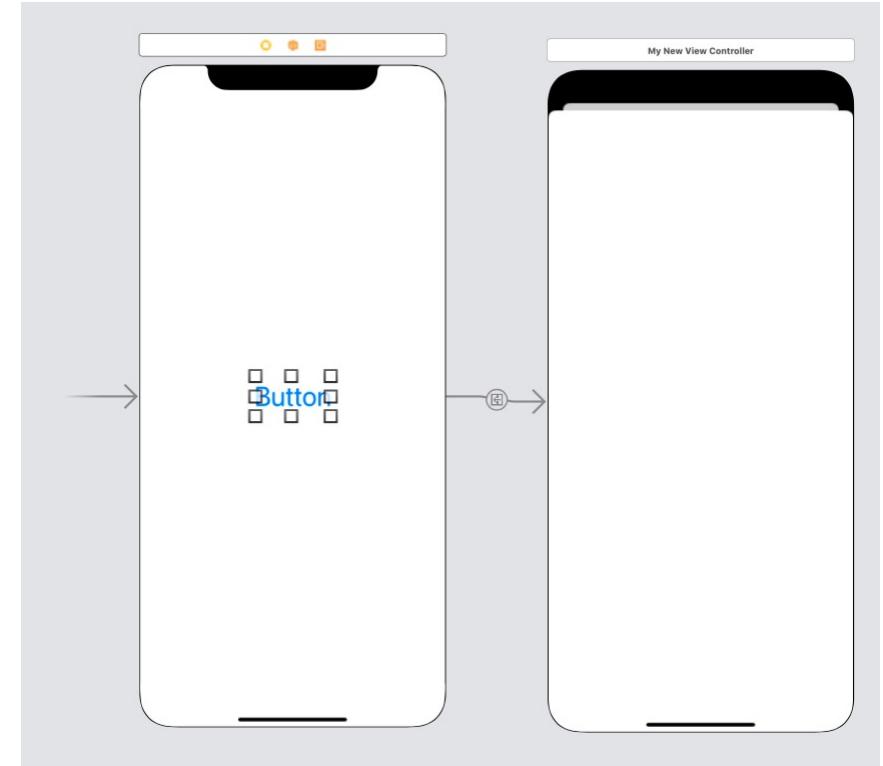
```
let mainStoryboard = UIStoryboard(name: "Main", bundle: .main)
let complimentVC = mainStoryboard.instantiateViewController(
    identifier: "ComplimentViewController") as! ComplimentViewController
present(complimentVC, animated: true, completion: nil)
```

# A Storyboard felépítése

- Egy Storyboard "jelenetekből" áll
- Jelenet ~ View Controller + nézetei
- A jelenetek között Segue-ek (ejtsd: szegvéj) létesíthetők:
  - **Tartalmazás (Relationship):** Container View Controller-ek és gyerekeik közötti tartalmazás, pl. Tab Bar Controller-ben a tabokhoz tartozó ViewController-ek
  - **Show:** "navigálás egy másik jelenetre", a fogadó ViewController-től függ, hogy pontosan mi történik
  - **Show Detail:** Detail nézet cseréje SplitViewController-ben
  - **Present Modally:** jelenet modális megjelenítése (kitakarja az aktuális jelenetet)
  - **Present as Popover:** iPaden\* ViewController megjelenítése Popoverben
  - **Unwind:** visszanavigálás, egy korábbi Segue "visszacsinálása", ritkán használjuk

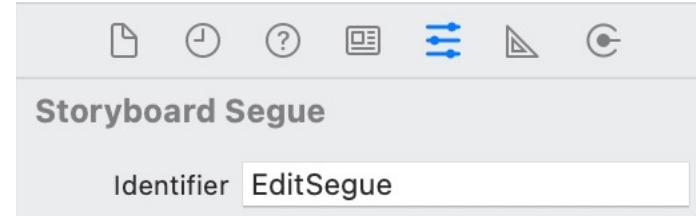
# Segue végrehajtása

- A segue végrehajtása történhet:
  - > Kódóból
  - > Egy adott elemhez (pl. UIButtonhoz) kötve közvetlenül a storyboardon belül



# Segue végrehajtása kódóból

- A Storyboard-ban felvett Segue-ekhez azonosítót rendelhetünk (**Identifier** attribútum)



- A kiinduló ViewController kódjából Segue a `performSegue` (`(withIdentifier:sender:)`) metódussal végrehajtható  
`performSegue(withIdentifier: "EditSegue", sender: self)`

# „Felkészülés” egy segue-re

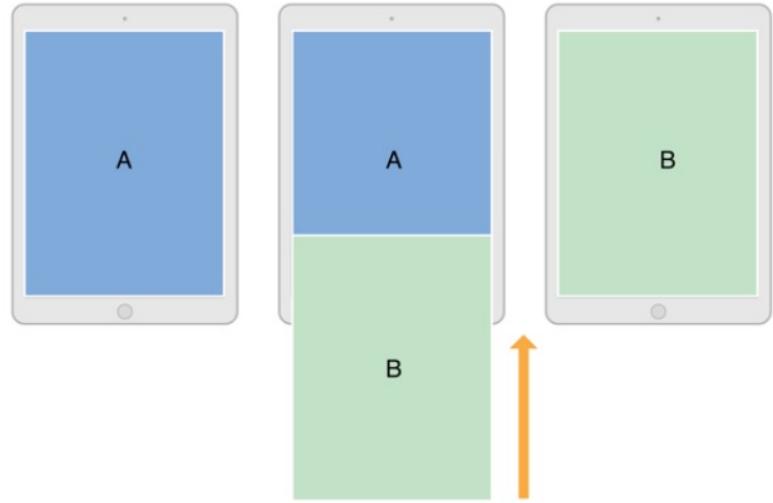
- Segue bekövetkeztekor mindenig egy „forrás” ViewController-ből váltunk át egy másik, „cél” ViewController-be
- A Segue végrehajtása előtt a „forrás” ViewController értesítést kap
- A kapott **segue** nevű paraméter **destination** property-jével tudjuk lekérdezni a cél ViewController-t (amire épp átváltunk)

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    //Prepare for the segue  
    let destinationVC = segue.destination as? RepeaterViewController  
    destinationVC?.number0fRepeats = 5  
}
```



# Modális megjelenítés

- ViewController-ek ideiglenes megjelenítésére
  - >iPhone-on általában kitakarja a képernyő egyéb részeit (vagy felül hagy egy pici helyet)
  - >iPaden jellemzően nem, „Popoverként” jelenik meg
- Blokkolja az alkalmazás egyéb nézeteinek és a navigációnak a használatát, amíg be nem zárják
- Bármilyen\* ViewController megjeleníthető modálisan bármilyen más ViewController felett



# Modális megjelenítés storyboardból

- Modális megjelenítéshez szükség van egy *prezentáló* ViewController-re, ami *felett* megjelenik a modális ViewController
  - > A prezentálónak nem kell Container View Controller-nek lennie
- A szülőből egy **Present Modally (Modal) Segue**-t indítunk a modálisan megjelenítendő ViewController-re
- Visszalépés tipikusan kódóból

# Modális megjelenítés kódból

- A UIViewController `present(_:animated:completion:)` metódusaival:

```
present(viewControllerToPresent, animated: true, completion: nil)
```

- A modálisan megjelenített ViewController eltüntetése kódból a `dismiss(animated:completion:)` hívással
  - > Akár a prezentált, akár a prezentáló hívhatja

```
dismiss(animated: true, completion: nil)
```

# Container View Controller-ek

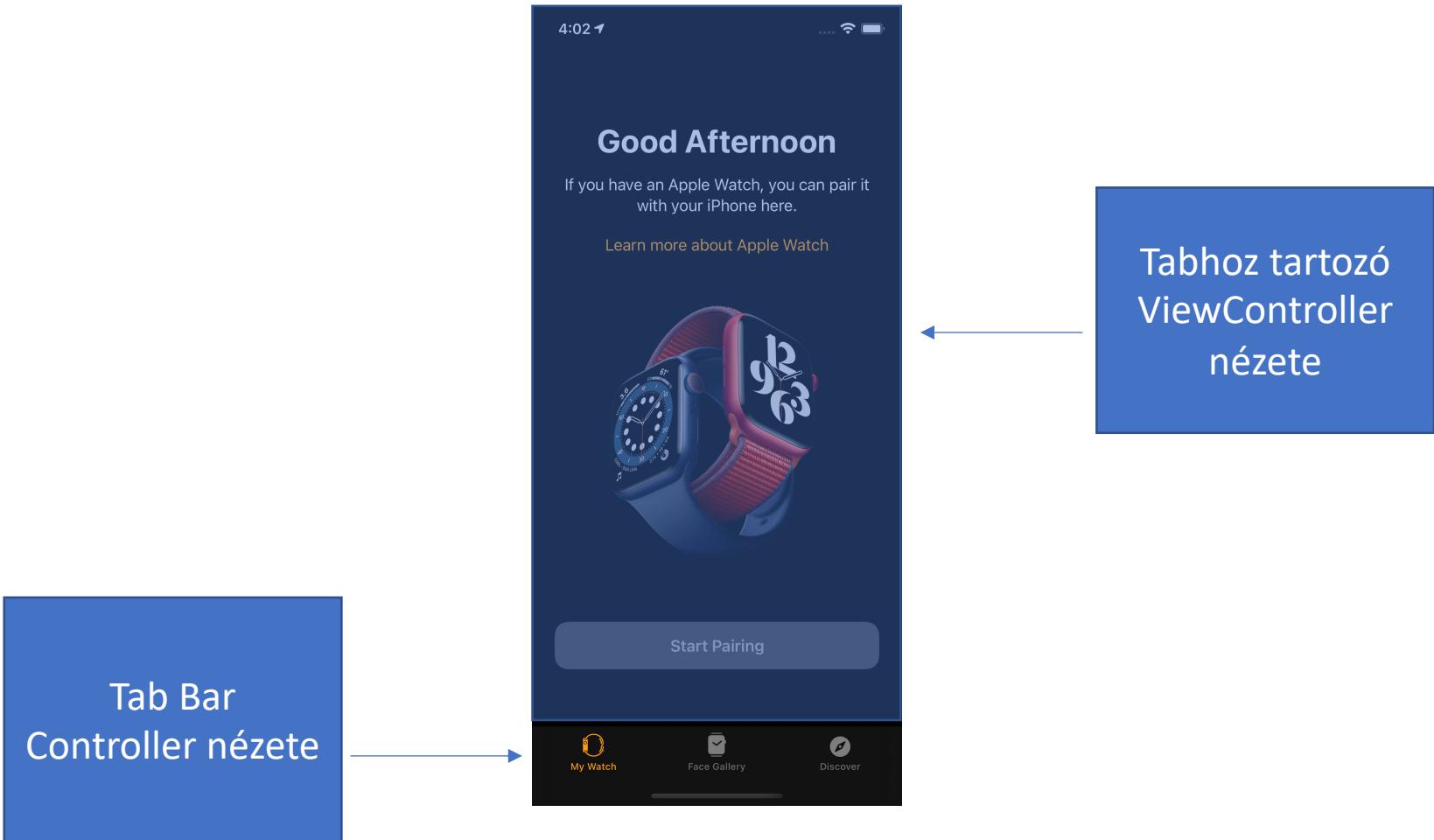
Tab Bar Controller



# Tab Bar Controller

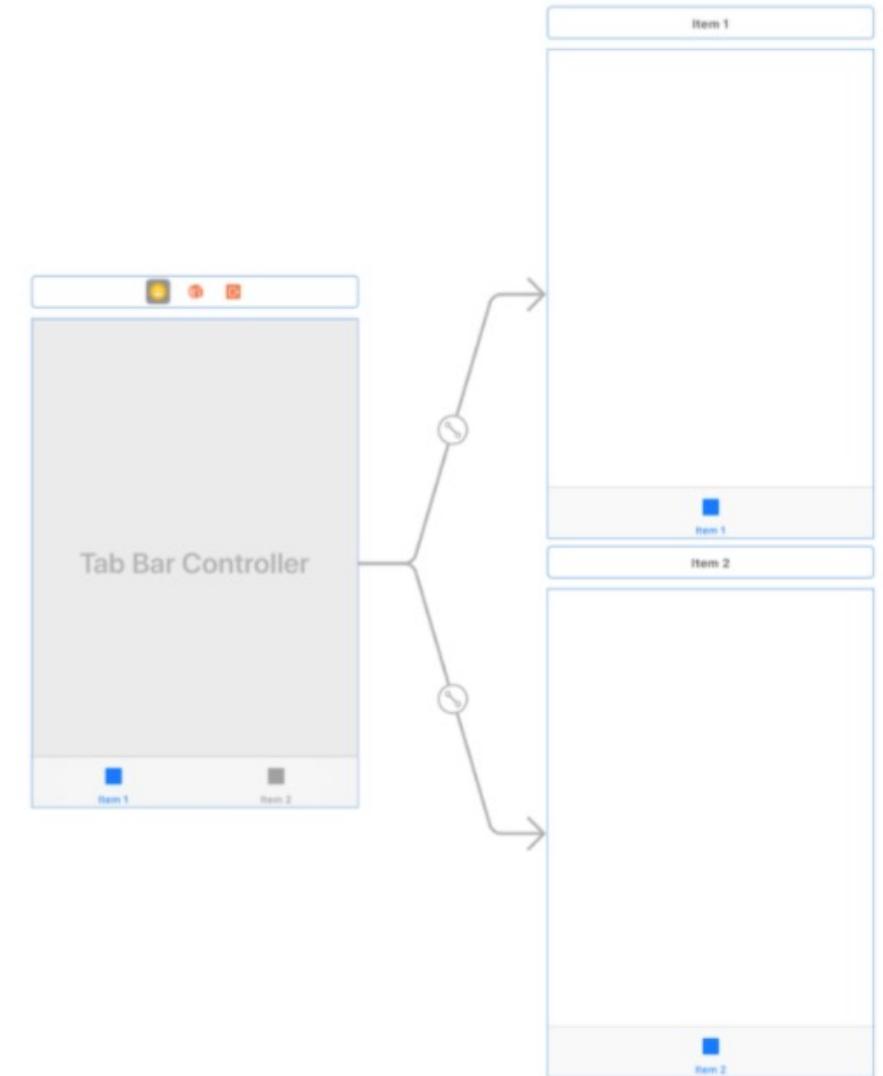
- Tabokra bontott felhasználói felületet valósít meg
- minden tabhoz tartozik egy külön ViewController (hierarchia)
- UITabBarController valósítja meg
  - > Nem vagy csak nagyon ritkán szoktunk egyedi osztályt leszármaztatni belőle, legtöbbször csak példányosítjuk
  - > Az opcionális eseménykezelést a UITabBarControllerDelegate megvalósításával végezhetjük

# Példa Watch App



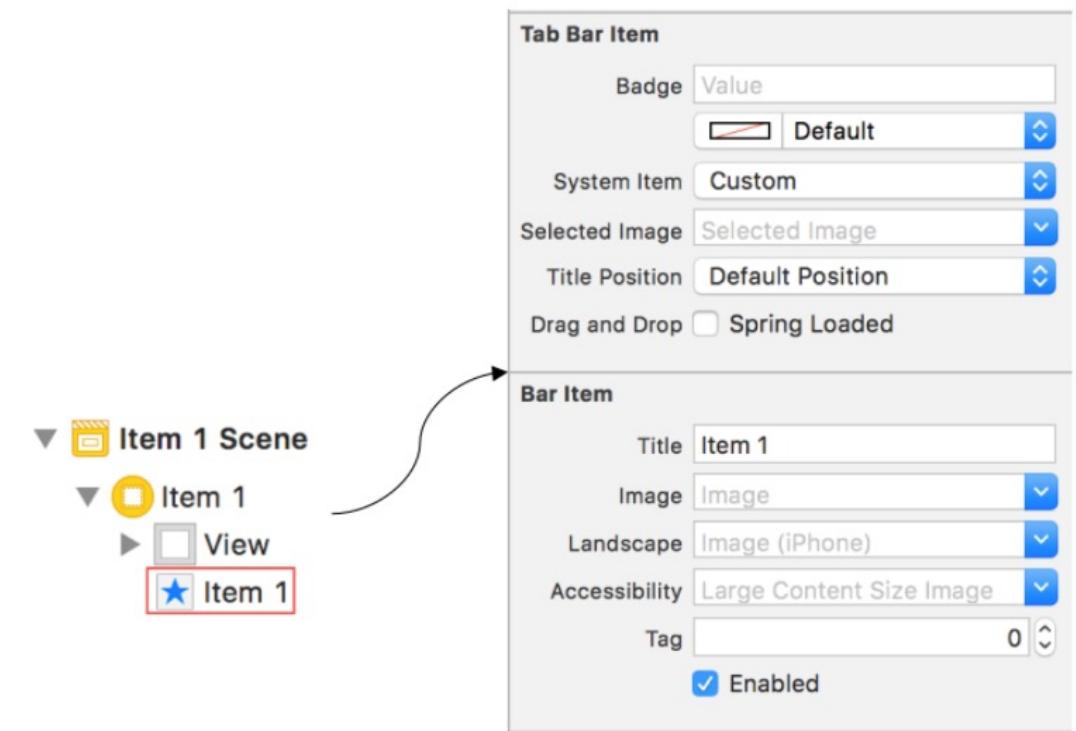
# Tab Bar Controller a Storyboard-ban

- A Tab Bar Controller és a taboknak megfelelő ViewController-ek között relationship kapcsolat
  - >A tabok a Tab Bar Controller gyerek ViewController-ei



# Tabok testreszabása I.

- A tabok szövegét és ikonját nem a Tab Bar Controller, hanem a tabokhoz tartozó ViewController-ek tartalmazzák
- minden ViewControllerhez tartozik egy **Tab Bar Item**
- Kódból és Interface Builderben is szerkeszthető



# Tabok testreszabása II.

- UITabBarItem

- > Beépített stílusú tabok létrehozása:

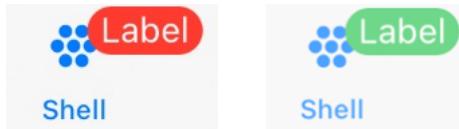
- `init(tabBarSystemItem systemItem: UITabBarItem.SystemItem, tag: Int)`

- > Egyedi tab létrehozása saját szöveggel és képpel:

- `init(title: String?, image: UIImage?, tag: Int)`

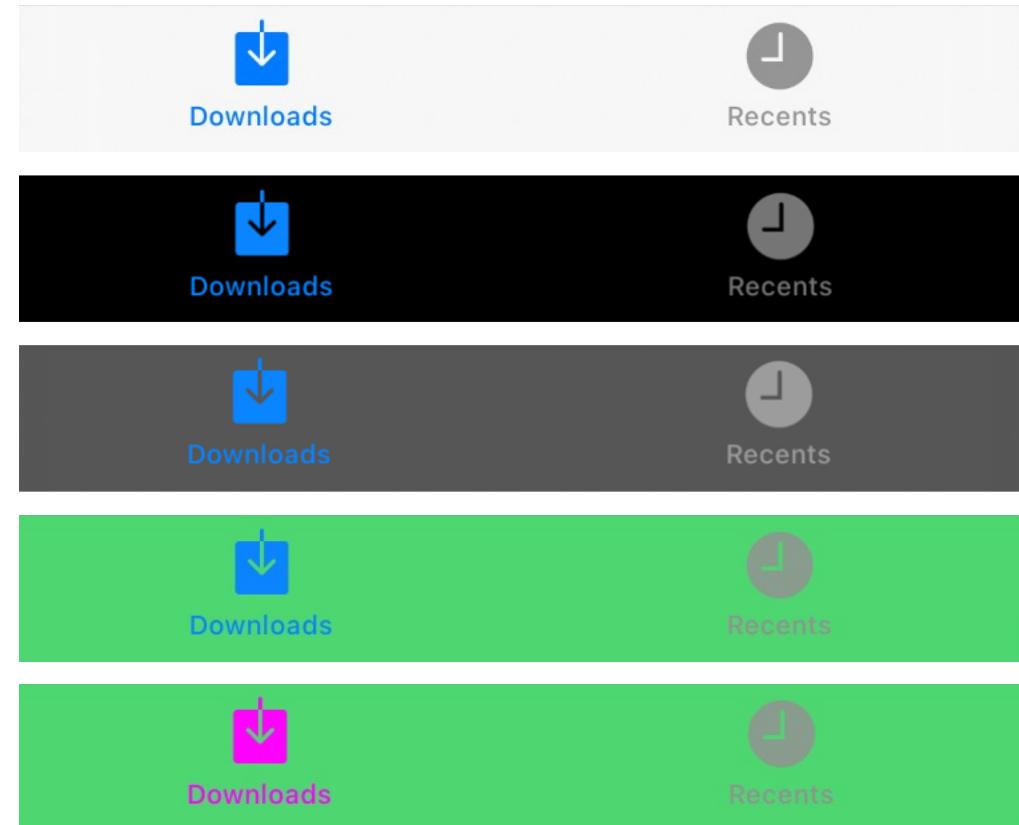
- > Badge (piros címke):

- Felirat módosítása: `badgeValue` property
    - Szín módosítása: `badgeColor` property



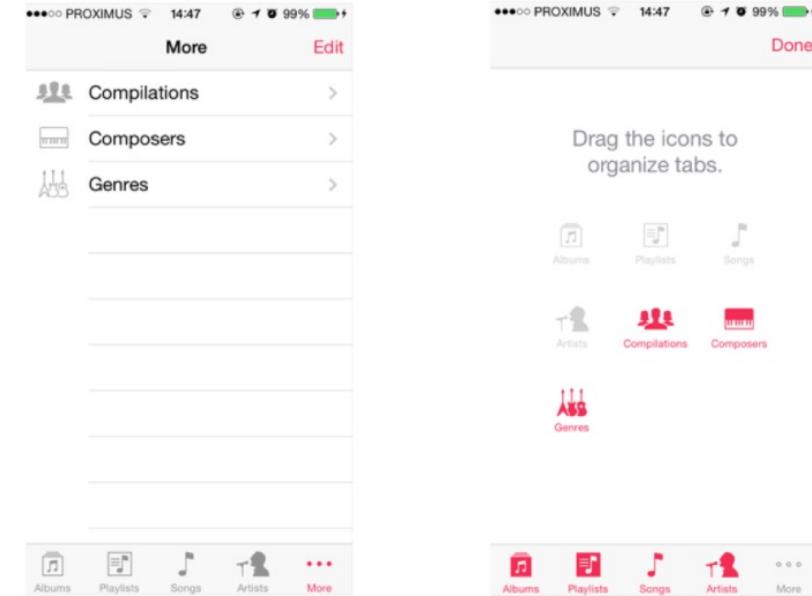
# A bar testreszabása

- Háttérszín:
  - > Beépített stílusok: **barStyle** property:
    - Default
    - Black
    - Áttetszőség: **isTranslucent** property
  - > Egyedi szín: **barTintColor** property
  - > Kijelölt szín: **tintColor** property



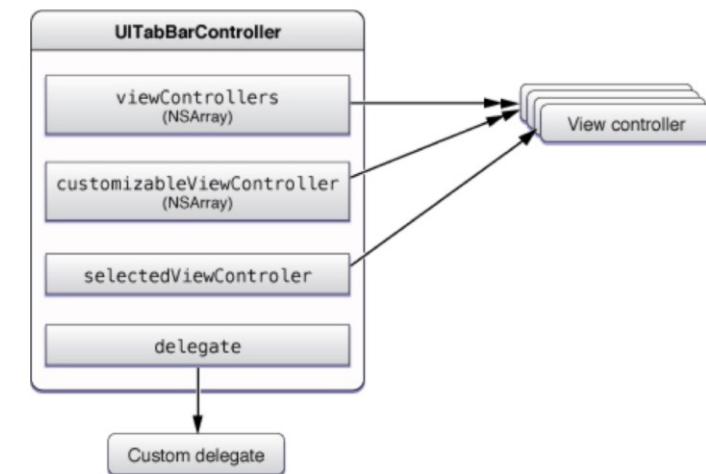
# Tabok száma

- A Tab Bar egyszerre maximum 5 tabot tud megjeleníteni
- Ha 5-nél több tab tartozik hozzá, akkor megjelenik egy *More* tab, ami egy lista nézetbe visz
- A felhasználó futási időben testreszabhatja, melyik tabok jelenjenek meg valódi tabként vagy a *More* listában
- Ezt a funkciót a rendszer automatikusan generálja, nem kell lekódolni!
- TIPP: a Tab Bar Controller `customizableViewControllers` property-jében adható meg, hogy mely tabokat tudja a felhasználó átrendezni. Alapból az összes tabot tartalmazza.



# Tab Bar Controller propertyk

- `viewControllers`: az egyes tabokhoz tartozó ViewController-ek listája
- `customizableViewControllers`: az átrendezhető tabok View  
Controlerei
- `selectedViewController`: az éppen aktív ViewController
- `delegate`: a Tab Bar Controller delegáltja



# Tab Bar Controller delegate

- `UITabBarControllerDelegate` protocol

`tabBarController?.delegate = self`

- Szerepe:

> Üzenetet kap tabváltásokról

```
func tabBarController(_ tabBarController: UITabBarController, didSelect  
viewController: UIViewController)
```

> Letilthatók a tabváltások

> Üzenetet kap tabok átrendezésekor

> Letilthatók a tab átrendezések

# Container View Controller-ek

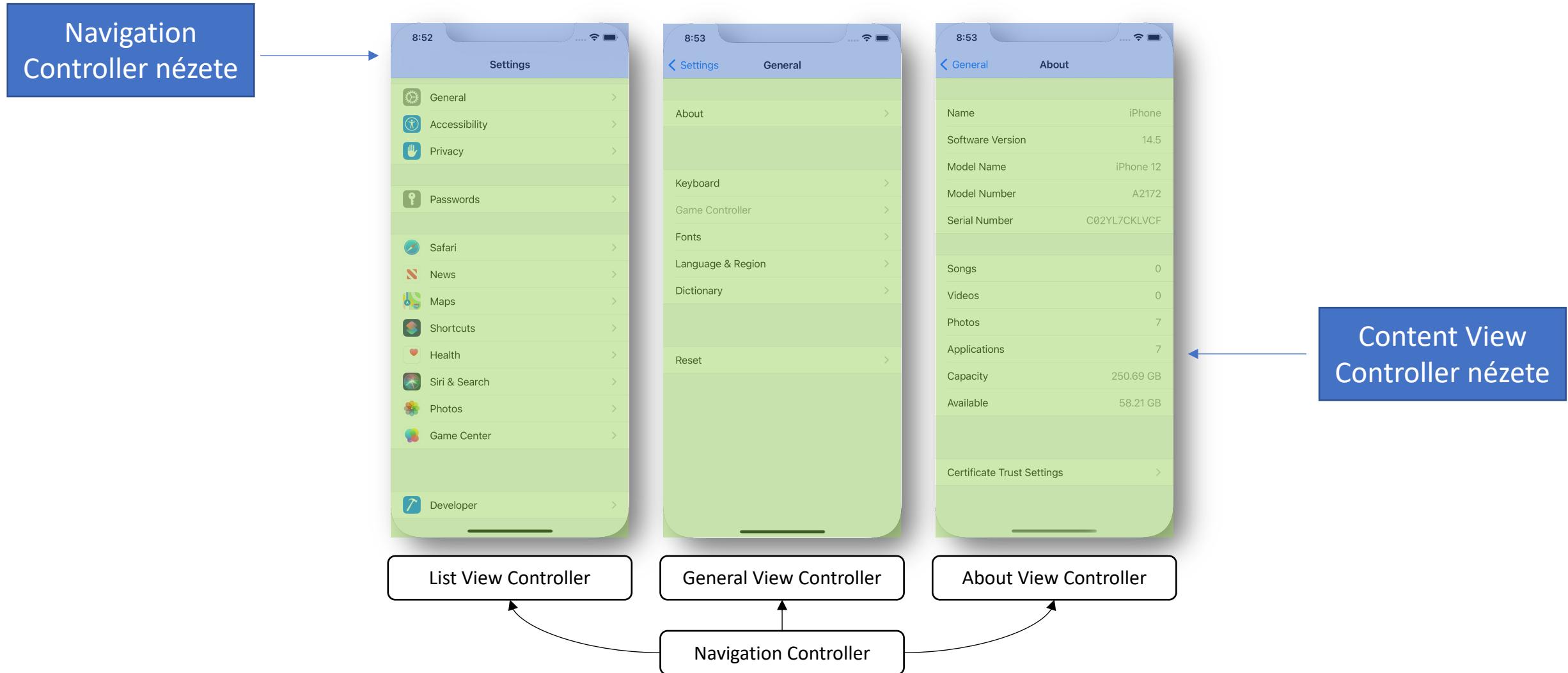
Navigation Controller



# Navigation Controller

- Többszintű nézethierarchia megjelenítésére
  - > A hozzárendelt ViewController-ek mint „egymásra rakott lapok”
  - > Visszatérés az előző lapra a Navigation Bar „vissza” gombjával
- A lapokat verem szerkezetű (LIFO) hierarchiába rendezi:
  - > **navigációs verem**
  - > Mindig a verem tetején lévő lap aktív
  - > A verem tetején lévő lapot eltávolítva visszatérünk az előző laphoz
- A megjelenített ViewController nézete felett található rész a Navigation Bar
- A "vissza" gombon minden az előző View Controller címe olvasható

# Példa: Settings app



# Navigation Controller Storyboard-ban

- A Navigation Controller és az általa megjelenített első lap (Root View Controller) között **relationship** kapcsolat
- A Navigation Controller `rootViewController` property-je kerül beállításra
- A navigálásnál további "lapokat" a Show (Push) Segue-jel tehetünk a navigációs verembe



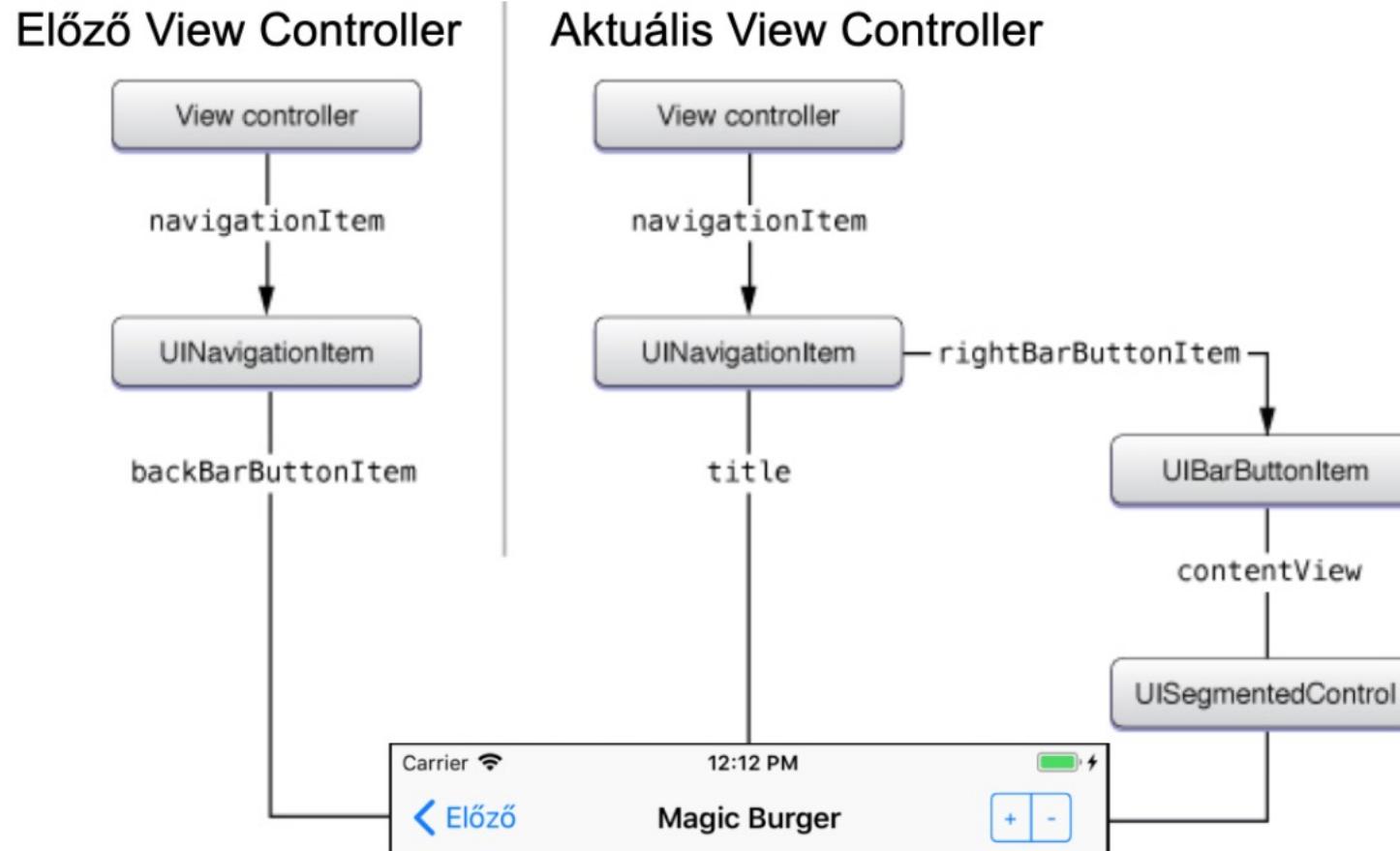
# Nézetváltás - navigáció

- Storyboard-ban: Show (Push) Segue
- minden ViewController-ből elérhető a hozzá tartozó Navigation Controller: `self.navigationController`
- Kódóból: ViewController-t felhelyezzük a Navigation Controller vermére: `pushViewController(_:animated:)`  
`navigationController?.pushViewController(someViewController, animated: true)`
- Visszalépés az előző nézetre
  - > Felhasználó megnyomja a vissza gombot a Navigation Bar-on
  - > Unwind Segue
  - > Programkódból, a Navigation Controller metódusával:  
`navigationController?.popViewController(animated: true)`

# Navigation Bar

- A Navigation Controller-ben lévő gyerek ViewControllerek-hez tartozik egy Navigation Bar
  - > **UINavigationBar**: UIView-ból származik, fel lehet használni másol is a UI-ban
- A Navigation Bar-on jelenik meg:
  - > Gomb az előző nézethez való visszatéréshez
  - > Az éppen aktív ViewController neve
  - > Opcionálisan további gombok
- A Navigation Bar elemeit a Navigation Controller állítja be a hozzáadott ViewController-ek **navigationItem** property-je alapján

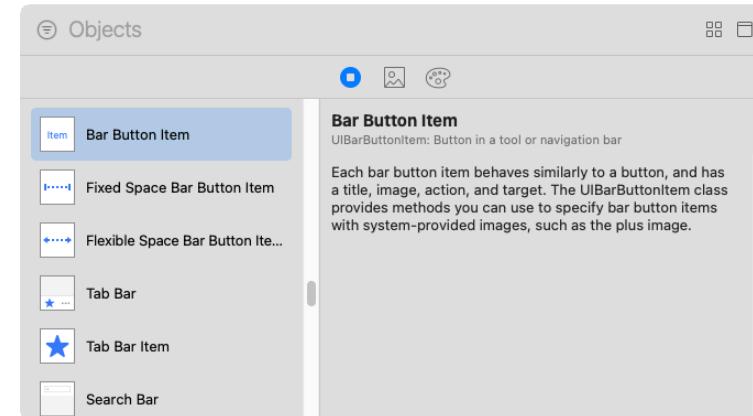
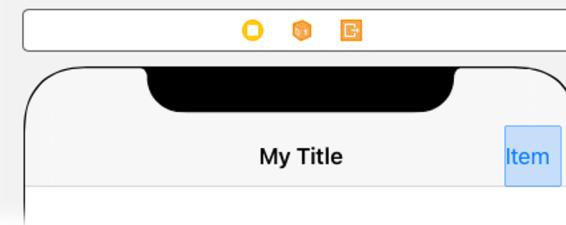
# Navigation Bar beállításai



# Gomb hozzáadása a barhoz

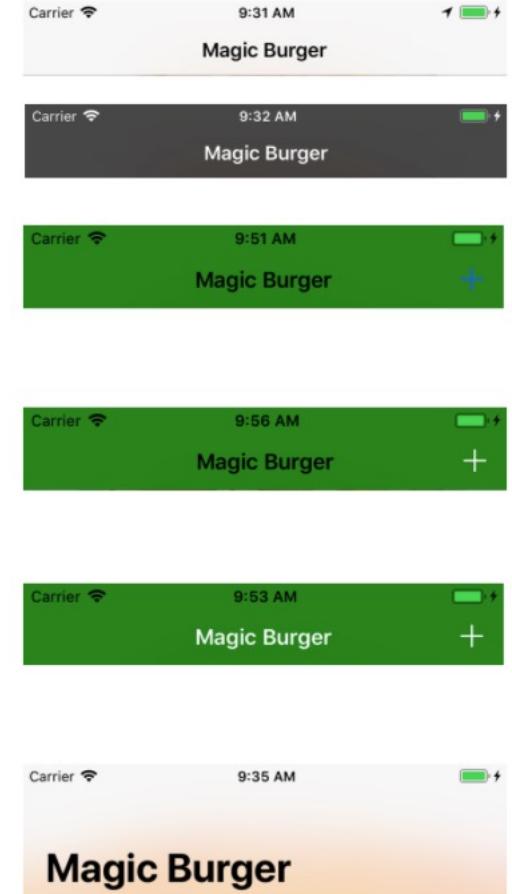
- Storyboard: "drag & drop" Bar Button Item
- Kódóból a View Controller `navigationItem`jéhez lehet hozzárendelni
  - > `right/leftBarButtonItem(s)` property-k

```
navigationItem.rightBarButtonItem =  
UIBarButtonItem(barButtonSystemItem: .edit, target: self, action: #selector(editButtonTap))
```



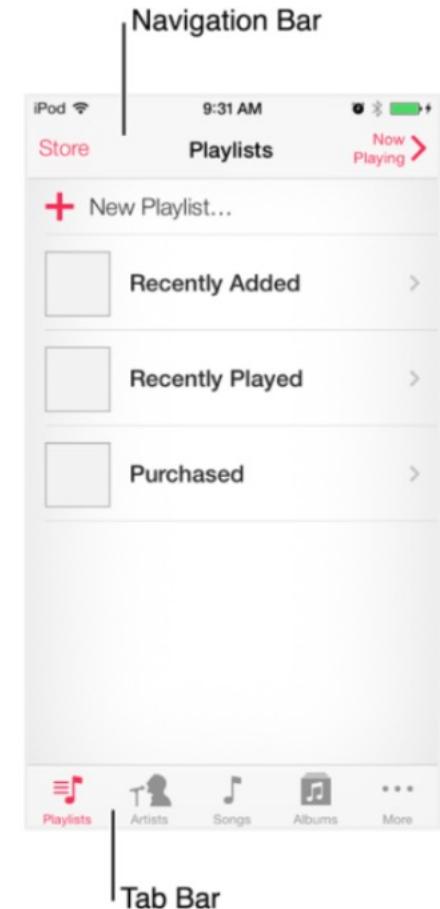
# Navigation Bar testreszabása

- Háttérszín:
  - > Beépített stílusok (`barStyle` property):
    - Default
    - Black
    - Áttetszőség: `isTranslucent` property
  - > Egyedi szín: `barTintColor` property
- Vissza gomb, jobb oldali nézet(ek) színe:
  - > `tintColor` property
- Felirat színe:
  - > `titleTextAttributes` property
- Large Title
  - > `prefersLargeTitles` property
  - > iOS 11-től, csak a legelső hierarchiaszintre ajánlott!



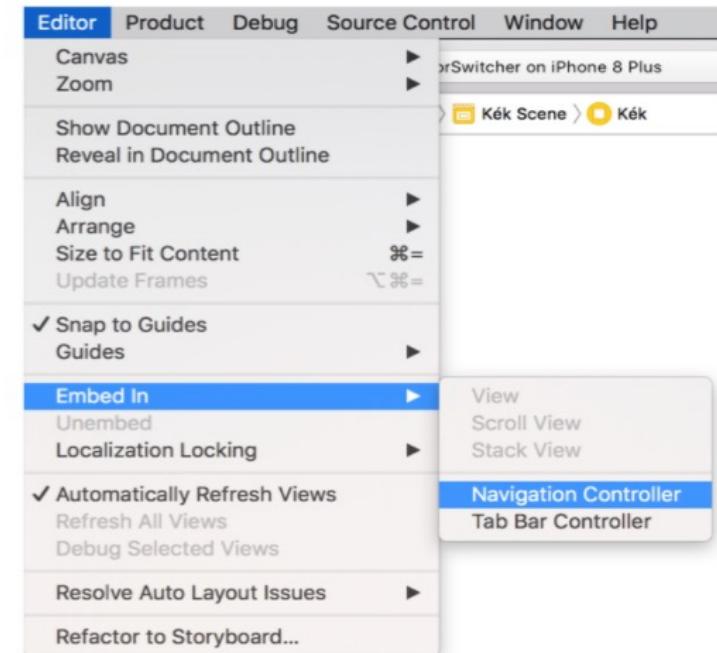
# Tab Bar és Navigation Bar együtt

- Tab Bar Controllerhez hozzáadott ViewController lehet Navigation Controller is
  - > Ilyenkor egy tabhoz tartozik a navigációs hierarchia
- Fordítva nem szokás! Navigation Controllerhez nem adunk hozzá Tab Bar Controllert!



# Interface builder: „Embed In”

- Az Interface Builderben az Editor menüben Embed In opcióval a kiválasztott View Controller beágazható:
  - > Tab Bar Controllerbe
  - > Navigation Controllerbe
- Kényelmi funkció



# Container View Controller elérése

- Mind a Navigation, mind a Tab Bar Controller elérhető bármelyik gyerek ViewController-ből azok property-jein keresztül:

```
var navigationController: UINavigationController? { get }
var tabBarController: UITabBarController? { get }
```

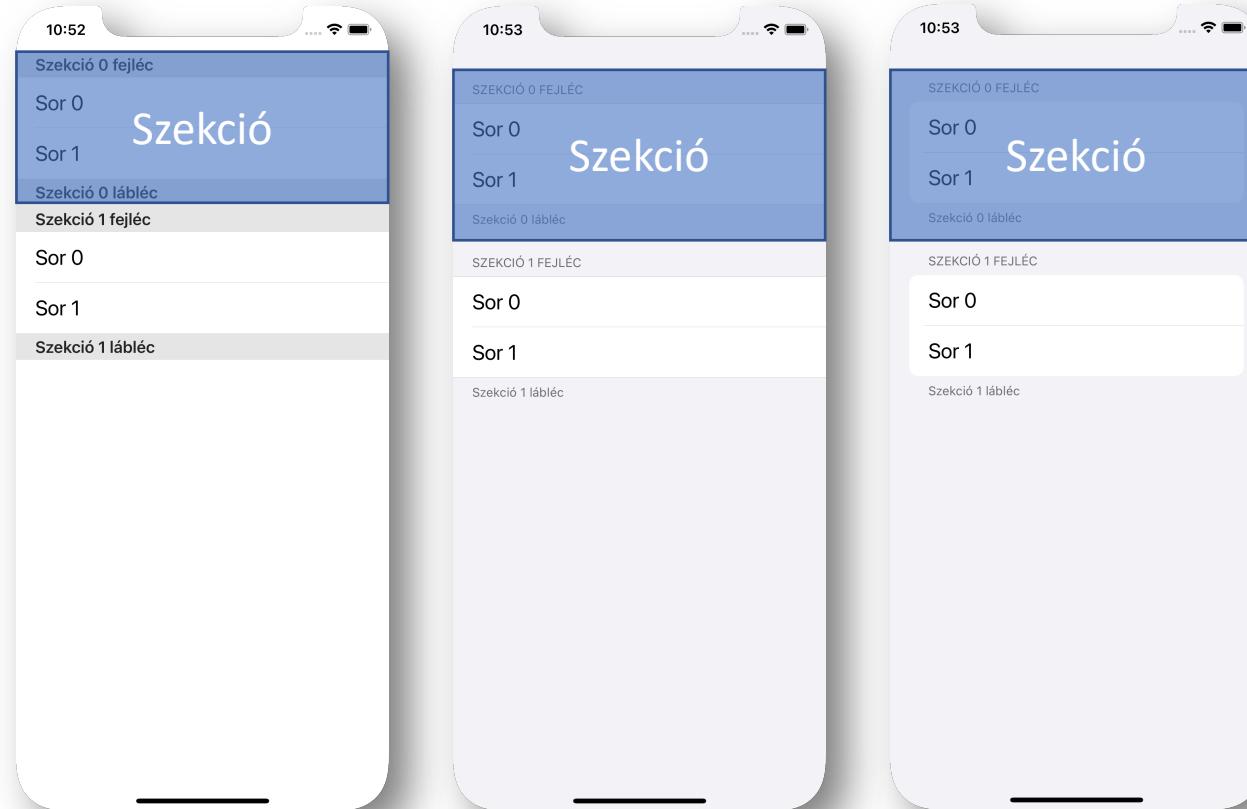
# Table View

# TableView

- A UITableView az iOS lista nézete
- 2 szintű hierarchia
  - > A TableView szekciókból (section) áll
  - > A szekciók cellákból (cell) állnak
- minden cella egy UITableViewCell példány
- Ha nem szeretnénk szekciókat használni, akkor az összes cellát az első (0.) szekcióba rakjuk
- minden szekcióhoz hozzárendelhető egy szöveges fejléc (header) és lábléc (footer)
  - > Vagy bármilyen egyedi nézet is
- A TableView-hoz rendelhető egy fejléc és lábléc nézet is (UIView)

# TableView stílusok

- plain, grouped, inset grouped



# TableView tartalmának megadása

- Dinamikus
  - > Kódból egy adatforrás protokoll (interfész) megvalósításával adjuk meg futás közben dinamikusan a cellákat
  - > Delegálás minta
  - > Cella prototípusokat használhatunk az egyes cellák kinézetének megadásához
- Statikus
  - > Fix számú és tartalmú cellát tartalmaznak
  - > A cellák tartalmát előre megadjuk Interface Builderben
  - > Egyfajta "layout" elemként szokás (ritkán) használni

# Dinamikus table view adata

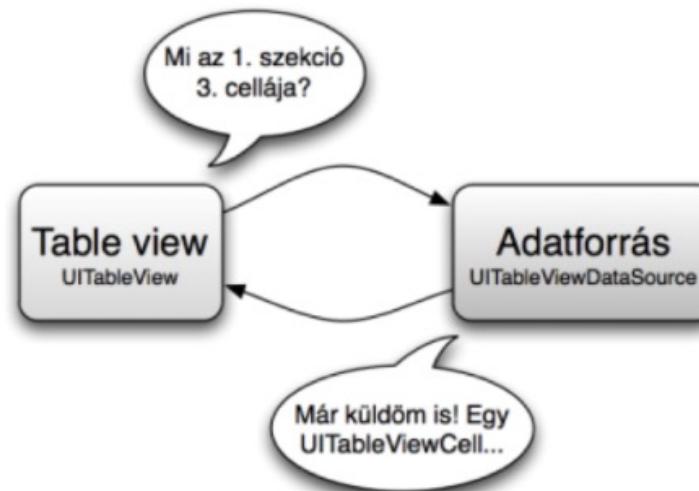
- Hogyan tudjuk elemekkel feltölteni a listát? Egy delegálton keresztül! minden TableView-hoz tartozik egy adatforrás (data source), melyet egy property tárol:

```
//Mutató egy UITableViewDataSource protocol-t megvalósító osztály  
(delegált) példányára  
var dataSource: UITableViewDataSource?
```

- Adatforrás lehet például a TableView-hoz tartozó ViewController
  - > Bármilyen más osztály is lehet, ami megvalósítja a UITableViewDataSource protokollt
- Az adatforrás feladatai
  - > A TableView szekcióinak és celláinak megadása
  - > Az „adatok” módosítása, ha a felhasználó töröl, módosít, beszűr vagy áthelyez egy cellát a felhasználói felületen

# Adatforrás működése

- A TableView üzeneteket küld az adatforrásnak, hogy adja meg a cellákat
  - > Optimalizálás! Csak azokat a cellákat kérdezi le, amik épp láthatóak vagy meg fognak jelenni.
- A cellák és szekciók azonosítása sorszámokkal történik (pl. 0. szekció 12. cella)
  - > IndexPath egy cellát azonosít, összefogja a cella szekcióját és egy cella sorszámát
    - indexPath.section // a szekció sorszáma
    - indexPath.row // a cella sorszáma a szekción belül



# UITableViewDataSource

- A szekciók számának megadása
  - > Alapesetben 1, nem kötelező felüldefiniálni

```
optional func numberOfSections(in tableView: UITableView) -> Int // Default is 1 if not implemented
```

- A szekciók fejlécének megadása (ha szeretnénk fejlécet)
  - > Hasonlóan adható meg a lábléc felirata is

```
optional func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String?
```

- Egy szekción belül a cellák számának megadása

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
```

- Cella megadása – Ezt a metódust hívja a TableView minden alkalommal, amikor egy cellát készül megjeleníteni

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell
```

# UITableViewDataSource példa

- Az adatok String példányok egy tömbben eltárolva:

```
var names: [String]
```

- Cellák számának megadása:

> Csak 1 TableView "szekciót" fogunk használni

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  
    return names.count  
}
```

- Cella megadása:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    let cell = tableView.dequeueReusableCell(withIdentifier: "NameCell", for: indexPath)  
    cell.textLabel?.text = names[indexPath.row]  
    return cell  
}
```

# TableView létrehozása

- 2 lehetőség:
  - > UITableView + különálló adatforrás osztály
    1. TableView (UITableView) létrehozása valahol a nézethierarchiában
    2. Az adatforrás protokoll (UITableViewDataSource) adoptálása egy ViewController-rel
    3. Adatforrás hozzárendelése a TableView-hoz
  - > UITableViewController
    - Kevesebb kód, további extra funkciók
    - Használjuk amikor csak lehet



# UITableViewController

- "Kényelmi osztály", ha egy TableView-t megjelenítő nézethez kell ViewController
  - > UIViewController-ből származik
  - > Megvalósítja a UITableViewDataSource és UITableViewDelegate protokolt
- További extra funkciók
  - > Beépített Edit gomb kezelése
  - > TableView adatainak újrátöltése megjelenítés előtt
  - > Scroll indikátor villogtatás a nézet megjelenítését követően

# Cella prototípus

- A dinamikus TableView celláinak felépítését a Storyboard-on belül is definiálhatjuk, prototípus cellák megadásával
  - > Prototípus: tartalmazza a cellához tartozó alnézeteket és ezek paramétereit



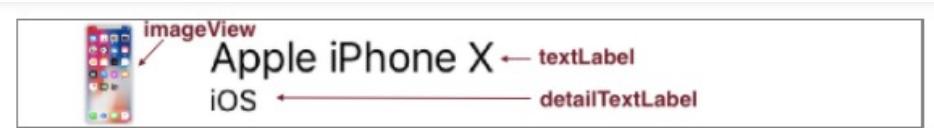
- Egy TableView-hoz több cella prototípus is tartozhat
  - > minden prototípushoz tartozik egy azonosító, pl. "MyCellId"
- Kódban a TableView metódusa hozza létre a cellát a prototípus azonosító alapján `dequeueReusableCellReusableCell(withIdentifier:for:)`

```
let cell = tableView.dequeueReusableCell(withIdentifier: "MyCellId", for: indexPath)
```

# UITableViewCell

- minden cella egy **UITableViewCell** típusú nézet objektum
  - > **UIView**-ból származik
- A cellák alapesetben 3 alnézetet tartalmaznak az adatok megjelenítéséhez
  - > Az alnézetek automatikusan jönnek létre, mikor futás során először hivatkoznak az alnézet property-jére

```
var imageView: UIImageView?  
var.textLabel: UILabel?  
var detailTextLabel: UILabel?
```



- > Az alapfelépítésen kívül teljesen egyedi cellák is definiálhatók, tetszőleges alnézetekkel

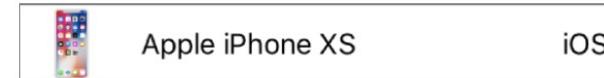
# Beépített cellatípusok

- Az alap cellafelépítéshez tartozó stílusok

> Basic



> Right Detail



> Left Detail



> Subtitle



- Storyboard-ban a cell prototype-nál adható meg

# Cellák újrahasznosítása

- Cella újrahasznosítás
  - > Egy TableView több (száz)ezer elemet is tartalmazhat, így nem megengedhető, hogy minden egyikhez egy külön UITableViewCell tartozzon
  - > Megoldás: csak néhány képernyőnyi cellát hozunk létre, és az új elemek megjelenítéséhez azokat használjuk fel, amik éppen nem látszanak a képernyőn
  - > A cella példányosításakor meg kell adni egy szöveges azonosítót, ez alapján lehet később újrahasznosítani az adott cellát

# Egyedi cella használata

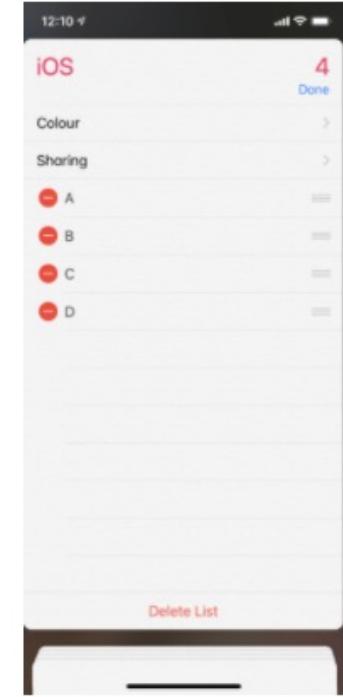
- Ha egyedi cellát szeretnénk használni:
  1. Hozzunk létre egy új UITableViewCell leszármazott osztályt (új fájlt)
  2. Adjunk hozzá a Storyboard-ban egy új cellát a TableView-hoz
  3. Állítsuk be az Identity inspector-ban, hogy a cella class-a az általunk létrehozott legyen
  4. Kezdhetjük egyedivé alakítani a cellát, az Assistant editor „forráskód” részében a cella swift fájlja fog megjelenni, ide hozzáadhatunk új outleteket is
  5. Ne felejtsünk el cella azonosítót beállítani!

# TableView adatainak módosítása

- Kódból, "manuálisan"
  - > Változtassunk az adatmodellen (pl. Array, adatbázis, stb.) törlése/beszúrás/szerkesztés
  - > TableView újratöltése: (ennek hatására a TableView újra meghívja az adatforrás metódusait és újratölti a cellákat)
    - Teljes TableView újratöltése `reloadData()`
    - Adott cellák újratöltése: `reloadRows(at:with:)`
    - Tetszőleges frissítés: `performBatchUpdates(_:completion:)`
- A TableView beépített szerkesztő funkcióival

# TableView szerkesztő funkciók

- A TableView szerkesztő funkcióival a felhasználó futás közben módosíthatja a TableView tartalmát
  - > Cella törlése
  - > Új cella beszúrása
  - > Cella áthelyezése a listán belül
- A TableView-t szerkesztő (edit) módba kell váltani, hogy megjelenjenek a szerkesztő gombok. Erre két lehetőség van:
  - > A TableView `editing` property-jét `true`-ra kell állítani
  - > A `UITableViewController` `editButtonItem` property-jét kell „használni”, ami egy kész `edit` gombot tartalmaz



# Cella törlése

- Ha a felhasználó megnyomja a delete gombot, az adatforrás `tableView(_:commit:forRowAt:)` metódusában kell kezelni

```
override func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
    if editingStyle == .delete {
        // TODO: Az adatmodellből is ki kell törölni a cellához tartozó bejegyzést
        tableView.deleteRows(at: [indexPath], with: .fade)
    }
}
```



- Ha szeretnénk letiltani egy cella törlését, akkor az adatforrás `tableView(_:canEditRowAt:)` metódusában térjünk vissza false-szal

```
override func tableView(_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {
    // Return false if you do not want the specified item to be editable.
    return true
}
```

# Cella érintés kezelése

- Vagy a UITableViewDelegate metódusával:

> `tableView(_ :didSelectRowAt :)`

- Vagy storyboard segue-jel

> Segue beköthető egy dinamikus TableView cella prototípusra

> `prepare(for:sender:)` meghívódik a Segue előtt és kiolvasható a kiválasztott cella indexe a TableView

`indexPathForSelectedRow` property-jével:

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    let selectedCellIndex = tableView indexPathForSelectedRow  
    ...  
}
```

# Data source vs delegate

- Data source
  - > Mit tartalmazzon a TableView
- Delegate
  - > Hogyan jelenítse meg az elemeket a TableView
  - > Értesítések TableView események bekövetkeztekor
- Egyszerűbb alkalmazásoknál minden két delegate-et a TableView-hoz tartozó ViewController szokta implementálni (ami legtöbbször egy UITableViewController)

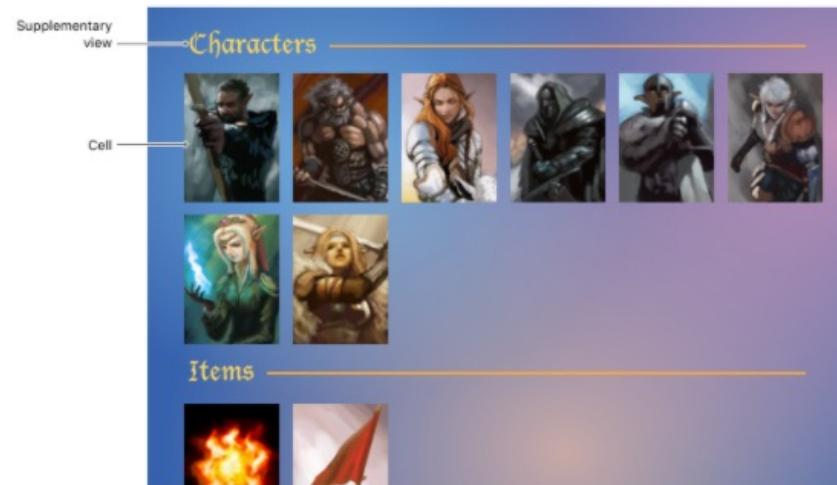
# Collection View



Automatizálási és  
Alkalmazott  
Informatikai Tanszék

# CollectionView

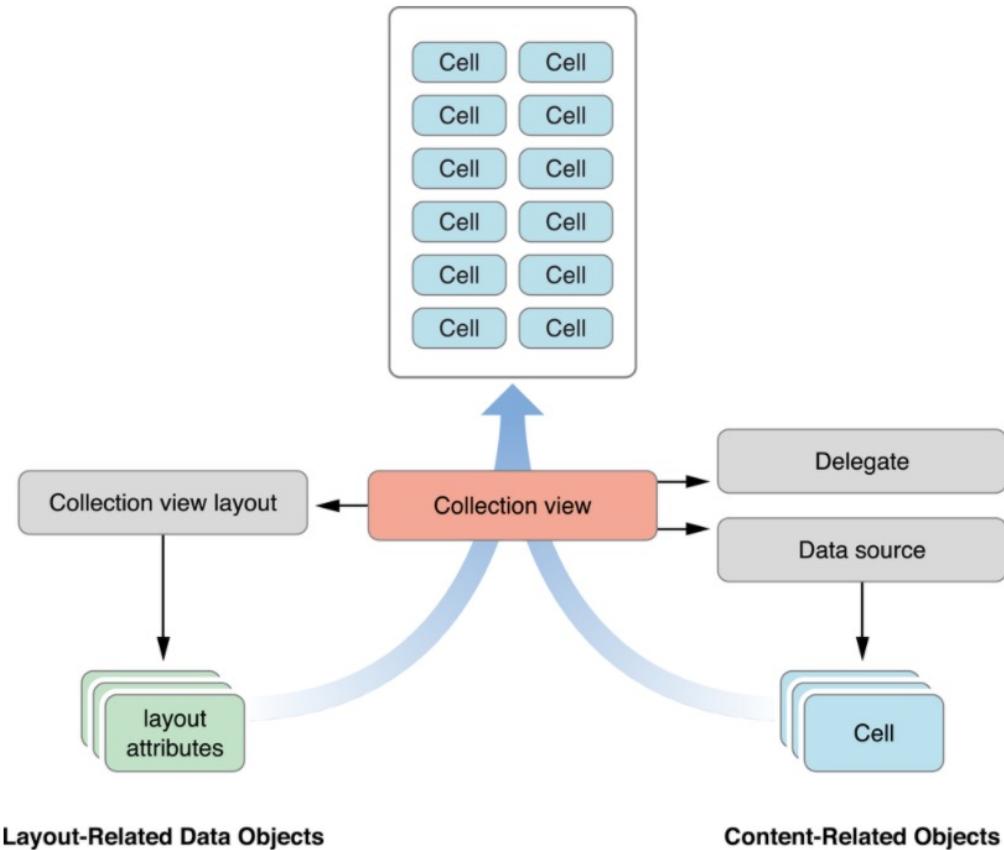
- Felfoghatjuk a TableView általánosításának
- Segítségével elemeket jeleníthetünk meg egy rendkívül rugalmas és testreszabható layout rendszer segítségével
  - > Leggyakrabban a beépített Flow layout-ot használjuk (UICollectionViewFlowLayout), ami egy rácsos elrendezést eredményez



# A CollectionView felépítése

- Alap felépítését tekintve analóg a TableView-val
  - > A TableView tapasztalatait felhasználva írták meg
  - > API nagyon hasonló
- 2 szintű hierarchia
  - > **section** (TableView-nál *row*)
  - > **cell**
- minden cella egy UICollectionViewCell példány
- A CollectionView-hoz hozzá lehet rendelni úgynevezett Supplementary View-kat. Ezek lehetnek:
  - > Fejléc vagy lábléc (section-önként vagy az egész CollectionView-hoz)
  - > Tetszőleges View a CollectionView layout-jának megfelelően

# A Collection View felépítése

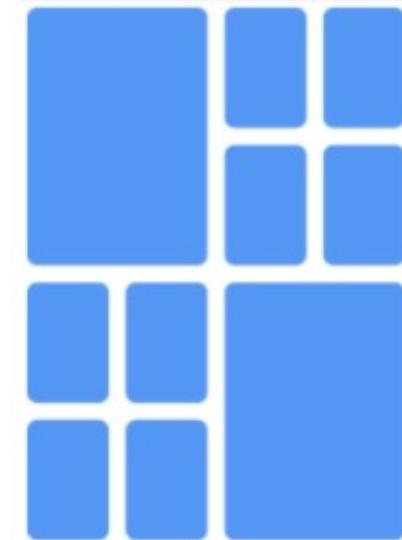


# A CollectionView felépítése II.

- UICollectionViewDataSource, hasonló metódusokkal és viselkedéssel mint a TableView-nál
- UICollectionViewDelegate, hasonló metódusokkal és viselkedéssel mint a TableView-nál
- UICollectionViewLayout
  - > Ez a legnagyobb különbség a TableView-hoz képest!
  - > Ha nincs szükségünk egyedi layout-ra, akkor használjuk a UICollectionViewFlowLayout-ot vagy CompositionalLayout-ot

# CollectionView layout

- Flow Layout:
  - > Grides elrendezést biztosít
  - > Az egyes cellák mérete adható meg, akár dinamikusan is
- Compositional Layout
  - > Az elemeket groupokba szervezi
  - > A groupokra és az egyes elemekre is megmondható, arányaiban mekkora legyen pl. a collectionview szélességéhez képest
  - > Könnyen lehet vele bonyolult layoutokat definiálni



# ScrollView



Automatizálási és  
Alkalmazott  
Informatikai Tanszék

# ScrollView

- UIScrollView
  - > Függőlegesen és/vagy vízszintesen görgethető felület
  - > Támogatja a nagyítást/kicsinyítést (pinch to zoom)
  - > Automatikusan megjeleníti/eltünteti a görgetősávokat (Scrolling Indicator)
    - Teljesen el is tüntethetjük
- A görgethető felület elemei: a Scroll View alnézetei
  - > Ugyanúgy szerkeszthetők mint bármely más nézet alnézetei (addSubview(), stb.)

# ScrollView – propertyk

- **var contentSize: CGSize**
  - > A megjelenített felület teljes mérete
  - > Interface Builderben nem lehet beállítani
- **var contentOffset: CGPoint**
  - > A "scroll ablak" eltolása a tartalomhoz képest
- **var isScrollEnabled: Bool**
  - > Görgetés engedélyezése és letiltása
- **var isDirectionalLockEnabled: Bool**
  - > A görgetés kezdeti irányának megtartása (vízszintes/függőleges)



# ScrollView - lapozás

- A ScrollView támogatja a „laponkénti” (képernyőnkénti) görgetést (*paging*)
- scrollView.isPagingEnabled = true
- A lapokat egyszerűen, alnézetek formájában adjuk hozzá a ScrollView-hoz
- Legtöbbször egy PageControl-t (**UIPageControl**) használunk a lapok számának és az aktuális lap jelöléséhez
  - > Külön nézet, nekünk kell létrehozni és frissíteni lapváltáskor

# ScrollView

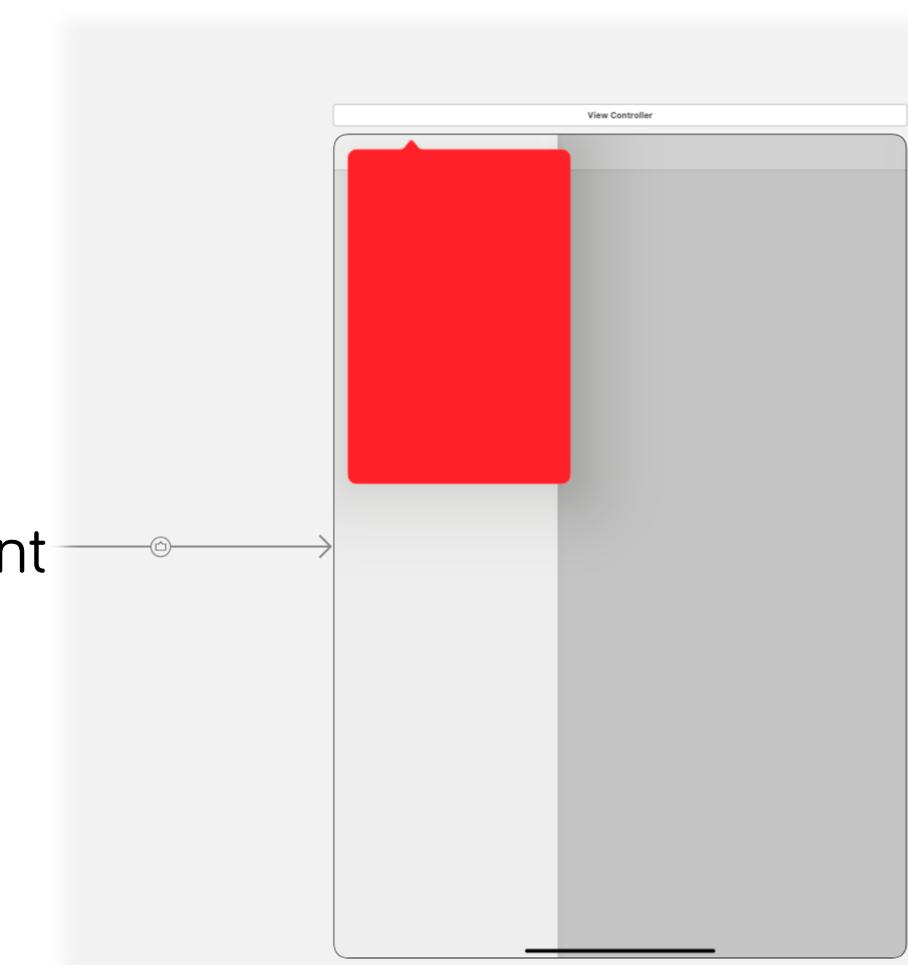
- A görgethető felületű nézetek ősosztálya a **UIScrollView**
- Leszármazott osztályok:
  - > UITableView
  - > UICollectionView
  - > UITextView
  - > ...

# iPad specifikus\* komponensek

# Popover

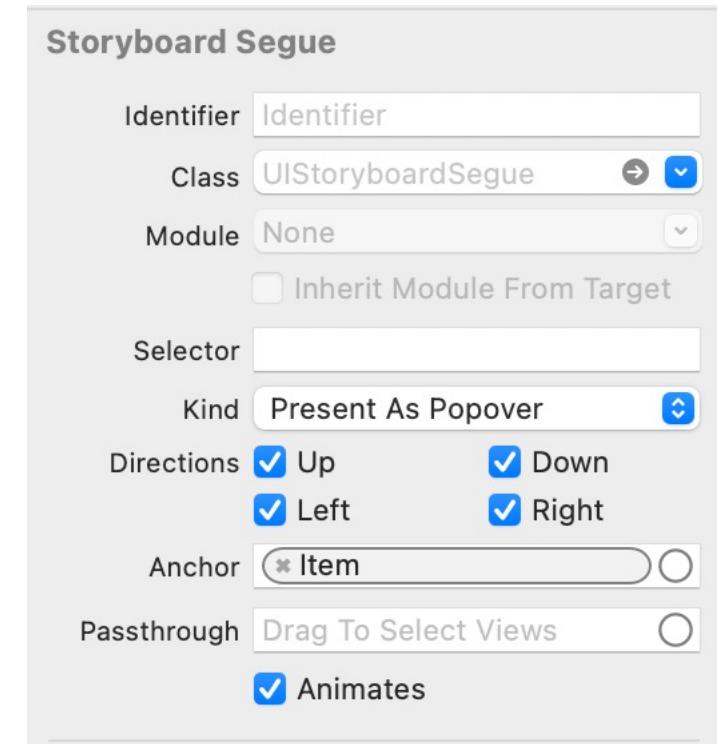
- Egy ViewController megjelenítése egy „popup”-ban
- Korábban csak iPad-re volt.
  - > Már elérhető iPhone-on is.
- Külön segue: **Present As Popover**
  - > Alapesetben iPhone-on modális prezentációként viselkedik, de felüldefiniálható
- Ha „félrekattintunk” bezáródik
- Popover bezárása kódból

```
dismiss(animated: true, completion: nil)
```



# Popover propertyk

- **Anchor:** az a nézet, amelyhez a Popovert hozzákötjük
- **Direction:** meghatározza, hogy a popover az Anchorhoz képest merre jelenjen meg (felül, alul, balra, jobbra)
  - > Pl.: "up": a popover nyila felfelé mutat, tehát a popover az Anchor alatt jelenik meg
- **Passthrough:** azon nézetek listája, melyek megérintése esetén **NEM** záródik be a popover



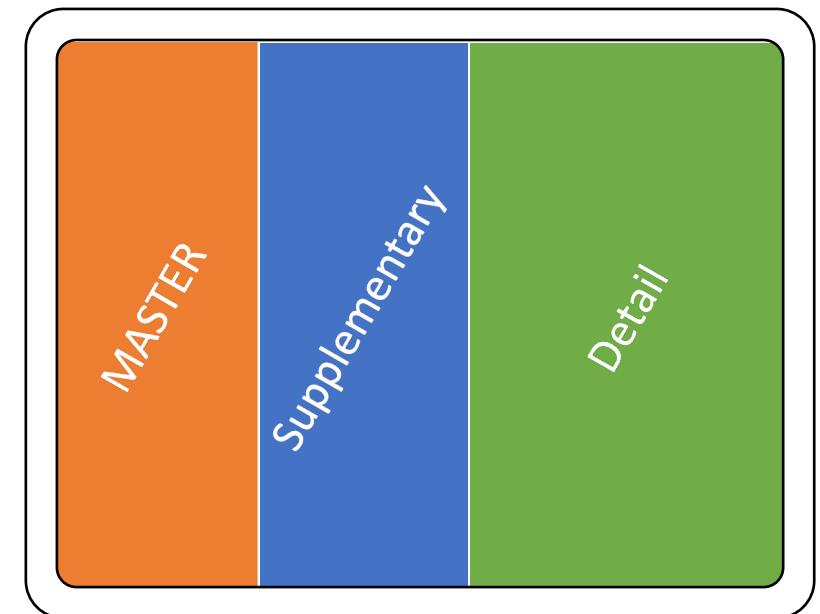
# Popover mérete

- A Popover alapesetben a megjelenítendő ViewController `preferredContentSize` property-je által megadott méretben jelenik meg
  - >`var preferredContentSize: CGSize`



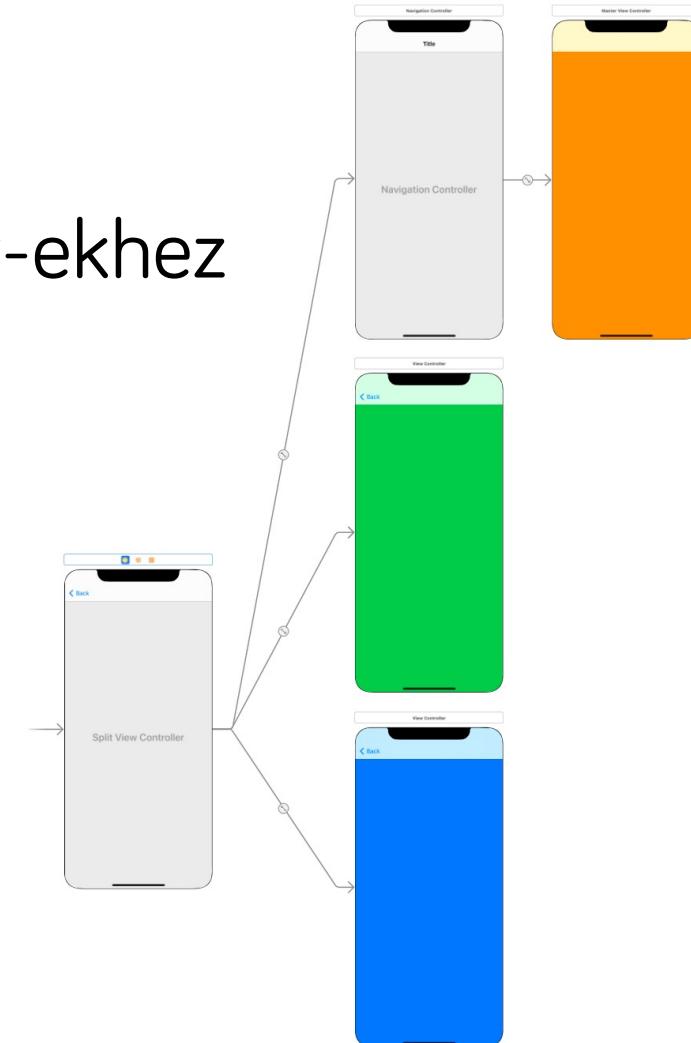
# Split View Controller

- Speciális Container View Controller, mely 2 vagy 3 panelben jeleníti meg egymás mellett a gyerek ViewControllerek-et
- Legtöbbször "master-detail" elrendezésű tartalom megjelenítéséhez
- iOS 7: csak iPad
- iOS 8+: bármilyen eszköz
- iOS 14+: opcionális 3. panel (Supplementary)
- Bal panel: „Master”
- Jobb panel: „Detail”
- Középső panel: „Supplementary”



# Split View Controller - Storyboard

- 3 (2) kapcsolat a gyerek ViewController-ekhez
  - > Master / Primary
  - > Detail / Secondary
  - > Supplementary

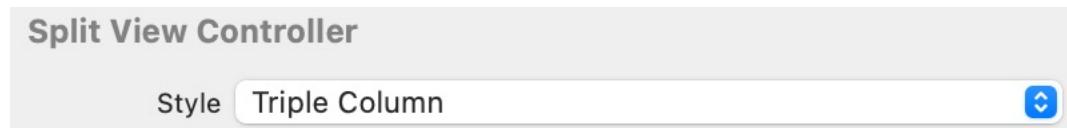


# Split View Controller - Megkötések

- minden gyermek ViewController egy NavigationController-be is be van ágyazva:
  - > Ha ezt elhagyjuk, a rendszer futási időben akkor is belerakja őket
  - > Előny: Megjelenhet a Navigation Bar
    - Speciális gombokhoz hasznos (pl.: megjelenítési mód váltás)
    - Meg például navigálni is lehet
- A gyökér ViewController nem ágyazható bele Navigation View Controller-be
  - > Használjuk alapnézetként ikább

# Split View Controller – Style

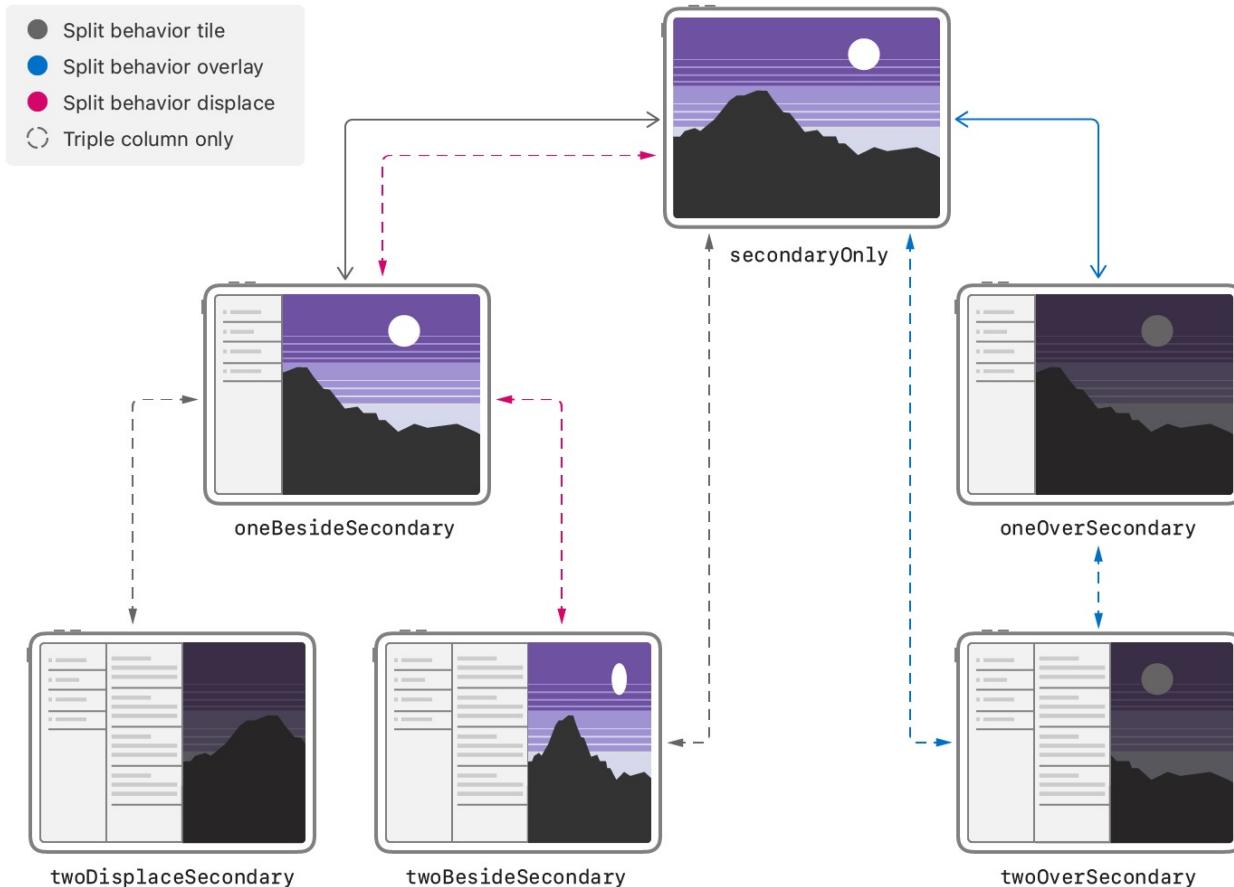
- iOS 14-től a `style` property beállításával megadható, hogy két vagy három alnézetet tartalmazzon a SplitViewController, vagy a hagyományos "classic" stílust használja.
  - > Double Column – 2 nézet összesen
  - > Triple Column – 3 nézet összesen
  - > Unspecified – klasszikus 2 nézet



# Split Behaviour & Display Mode

- A felosztás viselkedés (Split Behaviour) azt határozza meg, hogy az egyes nézetek egymáshoz képest hogy helyezkedjenek el:
  - > Tile : Egymás mellett kitöltve az ablakot
  - > Overlay: A master és a supplementary a secondary fölött vannak
  - > Displace : Egymás mellett, de kilóghatnak az ablakból
- A Split View Controller megjelenítési módját (display mode)
  - > A konkrét megjelenést határozza meg
  - > Függ a beállított felosztás viselkedésétől
  - > a **preferredDisplayMode** property-vel állíthatjuk
  - > Korábban elkérhetünk egy előre bekonfigurált gombot, mely átváltotta a megjelenítési módot, de ez már nem használható

# Display mode & Split Behaviour



# Display mode & Split Behaviour

Split Behaviour	Display mode	Kinézet
Tile	Secondary Only	Alapesetben csak a detail nézet látszódik (a többi előhozható)
	One Besides Secondary	A supplementary (ha nincs, akkor a master) és a detail egymás mellett vannak. Kitöltik a képernyőt.
	Two Besides Secondary	A master, a supplementary és a detail egymás mellett vannak. Kitöltik a képernyőt.
Overlay	Secondary Only	Alapesetben csak a detail nézet látszódik (a többi előhozható)
	One Over Secondary	A supplementary (ha nincs, akkor a master) a detail fölött va.
	Two Over Secondary	A master és a supplementary egymás mellett és a detail fölött vannak.
Displace	Secondary Only	Alapesetben csak a detail nézet látszódik (a többi előhozható)
	One Besides Secondary	A supplementary (ha nincs, akkor a master) és a detail egymás mellett vannak. A detail kilóghat a képernyőről.
	Two Displace Secondary	A master, a supplementary és a detail egymás mellett vannak. A detail kilóghat a képernyőről.

# SplitViewController elérés

- Az egyes gyermek nézetek a `splitViewController` property-n keresztül érik el szülő Split ViewController-t (ha létezik)
- A Split View Controller-től az alábbi metódus segítségével kérhetők el az egyes gyermek ViewController-ek

```
func viewController(for column:  
    UISplitViewController.Column) -> UIViewController?
```

➢ Figyeljünk oda, hogy ha a gyermek ViewController NavigationController-be van ágyazva, akkor azt adja vissza a fenti metódus