# iOS alapú szoftverfejlesztés - Labor 09

A laborsegédletet összeállította: Dr. Kelényi Imre - imre.kelenyi@aut.bme.hu

A labor témája:

* Core Data alapok
* Adatmodell definiálása: attribútumok és kapcsolatok (relationships)
* Objektumok létrehozása és lekérdezése
* NSFetchedResultsController

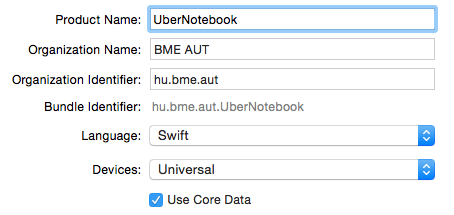
A laborhoz tartozó nagyobb kódrészletek a következő url-en érhetők el copy-paste barát formában:

https://gist.github.com/imrekel/379d8fdf937011ed7031

# UberNotebook

## Alkalmazás váz, Core Data alapok

Hozzunk létre egy új, **Single View** alkalmazást, iPhone támogatással, UberNotebook névvel. Ne felejtsük el bekapcsolni a "**Use Core Data**" opciót a projekt generálásakor!



Töröljük ki a projektből a generált ViewController.swift-et és a Main.storyboard-ból is távolítsuk el az ott felvett jelenetet (View Controller).

Érdemes megvizsgálni az AppDelegate-ben a Core Data stack-hez kapcsolódó metódusokat.

A **persistentStoreCoordinator** property definíciójában láthatjuk, hogy a perzisztens tároló egy SQLite adatbázis, ami az UberNotebook.sqlite-ban tárolja az adatokat

A **managedObjectContext** property segítségével fogjuk tudni elérni a kontextust, amin keresztül a Core Data műveleteket elvégezhetjük

A **saveContext()** metódust használjuk a kontextus mentéséhez. Egyrészt rögtön loggolja az esetleges hibát, másrészt csak akkor fog ténylegesen menteni, ha az előző mentés óta változott valami a kontextusban.

## Adatmodell definiálása

Nyissuk meg a **.xcdatamodeld** fájlt és vegyünk fel:

* új entitást a Notebook névvel:
  + title (String) attribútummal
* új entitást Note névvel
  + content (String) attribútummal
  + creationDate (Date) attribútummal

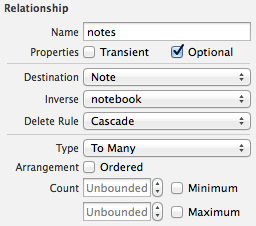
Vegyünk fel a Notebook-ba egy "**notes**" relationship-et, mely Note-ra hivatkozik!

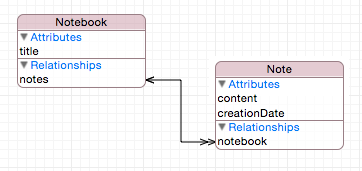
Vegyünk fel Note-ba egy "**notebook**" relationshipet, mely Notebook-ra hivatkozik!

Mindkét relationshipnél állítsuk be az inverz relációt a másikra!

A Core Data-ban relációknál mindig meg kell adnunk egy inverz relációt is. Erre azért van szükség, hogy az objektum gráf ne kerülhessen inkonzisztens állapotba, például törlés esetén (ha egy entitásra van egy reláció, de ennek a relációnak nincs inverze, akkor az entitás törlése esetén nem lehetne értesíteni a reláció tulajdonosát, hogy törlődött egy hivatkozott objektum).

Állítsuk be az "notes" relációt "To-Many"-re és "Cascade" törlési szabályra (így ha törlődik a Notebook, a bejegyzései is törlődnek vele együtt).





Itt az ideje az adatmodell kipróbálásának! Az App Delegate **didFinisLaunchingWithOptions()**-ban hozzunk létre egy új Notebook-ot és benne egy Note-ot:

let notebook =   
 NSEntityDescription.insertNewObjectForEntityForName("Notebook",   
 inManagedObjectContext: managedObjectContext) as NSManagedObject

notebook.setValue("Notebook \(arc4random\_uniform(10000))",   
 forKey: "title")

let note = NSEntityDescription.insertNewObjectForEntityForName("Note",   
 inManagedObjectContext: managedObjectContext) as NSManagedObject

note.setValue("\(arc4random\_uniform(10000)) a kedvenc véletlen számom!",   
 forKey: "content")

note.setValue(NSDate(), forKey: "creationDate")

note.setValue(notebook, forKey: "notebook")

saveContext()

https://gist.github.com/imrekel/379d8fdf937011ed7031#file-insertnotebook-swift

A sikeres mentés esetén kérdezzük le és listázzuk ki az összes elmentett jegyzetet (mivel az alkalmazás indításakor mindig létrehozunk egy új Notebook-ot és benne egy Note-ot, a log-ban is minden indítás után egyre hosszabb felsorolást kapunk):

let fetchRequest = NSFetchRequest(entityName:"Note")

do {

if let notes = try managedObjectContext.executeFetchRequest(

fetchRequest) as? [NSManagedObject] {

for note in notes {

let content = note.valueForKey("content") as! String

print(content)

}

}

} catch let error as NSError {

print("Could not fetch \(error), \(error.userInfo)")

}

https://gist.github.com/imrekel/379d8fdf937011ed7031#file-fetchnotes-swift

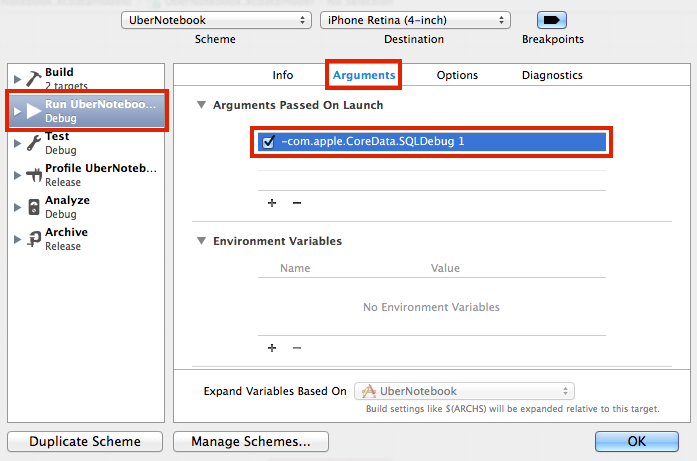
Figyeljük meg a **do-catch** párost, mely a Swift 2.0-ban bevezetett új hibakezelés. Bármilyen metódus, mely hibával térhet vissza, csak egy **do** blokkon belül hívható továbbá ezen metódusok hívását külön meg kell jelölni a **try** kulcsszóval. A hibát a **catch** blokkban tudjuk feldolgozni.

Próbáljuk ki, hogy bár a lekérdezésben csak Note-okat kérünk le, a lekérdezett objektumok relációs property-jein keresztül el tudunk érni más entitásokat is (ilyen esetekben a Core Data automatikusan elvégzi a lekérdezést a háttérben). Esetünkben le tudjuk kérni a Note-hoz tartozó Notebook-ot:

let notebook = note.valueForKey("notebook") as! NSManagedObject

print(notebook.valueForKey("title") as! String)

Kapcsoljuk be a Product/Scheme/Edit Scheme menüben, hogy a futtatáskor a konzolon megjelenjenek a Core Data használata közben kiadott SQL utasítások. Ehhez a "**-com.apple.CoreData.SQLDebug 1**" argumentumot kell felvenni:



Miután kipróbáltuk az alkalmazást érdemes kikapcsolni az SQL loggolást.

## Adatmodell osztályok

A Core Data programozás során mindenre használhatunk NSManagedObject típusú objektumokat, de ennél sokkal kényelmesebb és biztonságosabb, ha külön osztályt definiálunk a különálló entitásoknak.

Ehhez az Xcode is segítséget nyújt, az adatmodellben az entitást kiválasztva, az **Editor/Create NSManagegObject Subclass...** opcióval legenerálja az osztályokat az entitásokhoz. Tegyük ezt a Note és Notebook entitásokkal.

Ügyeljünk rá, hogy Swift fájlokat generáljunk!

Xcode 7-től minden managed object alosztályhoz két külön Swift fájl jön létre. Az Entity+CoreDataProperties.swift-ban egy külön kategóriába kerülnek a generált propertyk, míg az egyedi, általunk hozzáadott, kód az Entity.swift fájlba kerülhet.

Előfordulhat, hogy a relációs property-knél NSManagedObject típusú mutató szerepel a konkrét osztály típusa (esetünkben pl. Notebook) helyett. Ilyen akkor történik, ha még nem generáltuk le a reláció osztályát mikor a fájl keletkezett. Átírhatjuk kézzel is de, ha újrageneráljuk a fájlt, az is segít!

## Notebook-ok megjelenítése

Hogy megkönnyítsük a Managed Object Context elérését, vegyünk fel egy osztálymetódust UNAppDelegate-be:

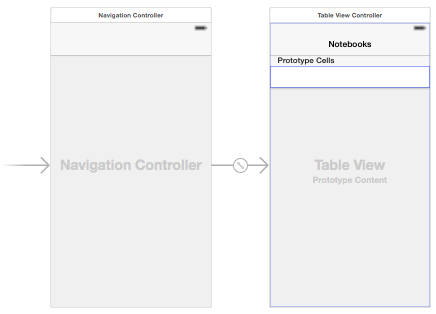
class var sharedAppDelegate: AppDelegate {

return UIApplication.sharedApplication().delegate as! AppDelegate

}

Hozzunk létre egy új UITableViewController-ből származó osztályt **NotebookViewController** névvel.

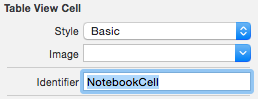
A Main.storyboardban vegyünk fel egy új Table View Controllert és ágyazzuk be egy Navigation Controller-be, amit jelöljünk ki a kezdeti view controllernek:



A .stroyboard fájlban válasszuk ki a Table View Controllert és

1. A Navigation Bar fejlécére klikkelve nevezzük át "Notebooks"-ra
2. Az Identity Inspector-ban változtassuk át az osztályát "NotebookViewController"-re

Állítsuk be a cella prototípus azonosítóját „**NotebookCell**”-re, a típusát **Basic**-re:



NotebookViewController.swift fájlban importáljuk be a **CoreData** modult és vegyünk fel egy property-t a jegyzetfüzetek tárolására:

import CoreData

var notebooks = [Notebook]()

Definiáljunk egy metódust, mely lekéri a Notebook-okat:

func fetchNotebooks() {

let moc = AppDelegate.sharedAppDelegate.managedObjectContext

let fetchRequest = NSFetchRequest(entityName:"Notebook")

do {

if let notebooks = try moc.executeFetchRequest(fetchRequest) as?   
 [Notebook] {

self.notebooks = notebooks

}

} catch {

print("Could not fetch")

}

}

https://gist.github.com/imrekel/379d8fdf937011ed7031#file-notebookviewcontroller-fetchnotebooks-swift

Hívjuk meg fetchNotebooks()-ot viewDidLoad()-ban:

override func viewDidLoad() {

super.viewDidLoad()

fetchNotebooks()

}

A UITableViewDataSource metódusok közül töröljük ki a szekciók számát megadót és térjünk vissza a sorok számával:

override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

return notebooks.count

}

Térjünk vissza egy új cellával a tömbből kiolvasott megfelelő indexű Notebook címével:

override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {

let cell = tableView.dequeueReusableCellWithIdentifier("NotebookCell",   
 forIndexPath: indexPath) as UITableViewCell

let notebook = notebooks[indexPath.row]

cell.textLabel?.text = notebook.title

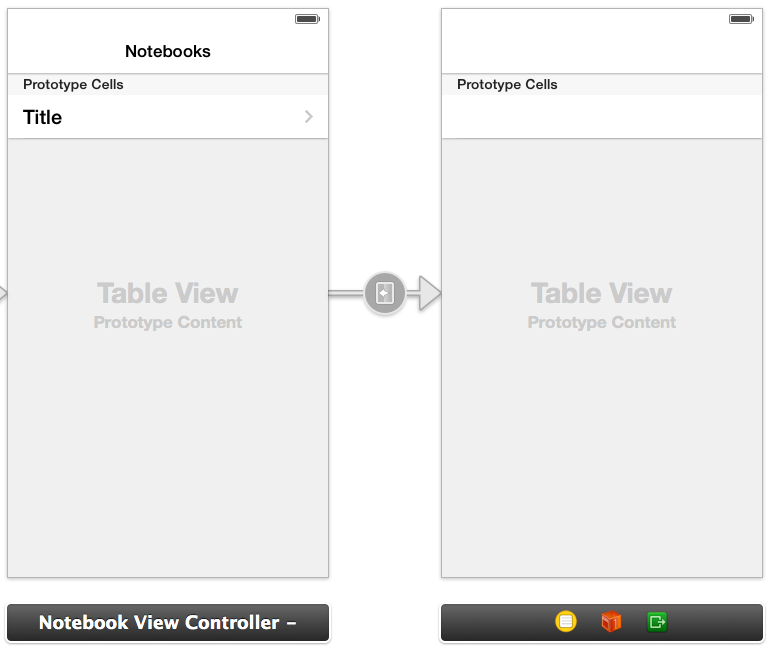
return cell

}

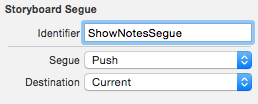
Ellenőrizzük le, hogy megjelennek a jegyzetek a nézetben!

## Jegyzetek, NSFetchedResultsController

A .storyboard-ba hozzunk létre egy új Table View Controllert! Kössük be egy Push vagy Show segue-el a Notebooks View Controller cellájára:

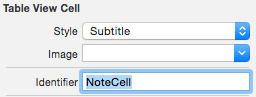


Válasszuk ki a segue-t és adjuk meg azonosítóul: "**ShowNotesSegue**"



Hozzunk létre egy új osztály, **NoteViewController** névvel, mely UITableViewController-ből származik, majd a storyboardban állítsuk be ezt az osztályt az új jelenethez.

A cella prototípus stílusát állítsuk "Subtitle"-re, az Identifier attribútumot pedig "NoteCell"-re:



NoteViewController.swift-ben importáljuk a CoreData modult és vegyünk fel egy notebook property-t, melyben azt tároljuk el, hogy melyik Notebook jegyzeteit mutatja a nézet:

import CoreData

var notebook: Notebook!

Nyissuk meg NotebookViewController.swift-tet és definiáljuk felül a prepareForSegue() metódust, melyben átadhatjuk a megjelenő jegyzet nézetnek a kiválasztott Notebook-ot:

override func prepareForSegue(segue: UIStoryboardSegue,   
 sender: AnyObject?) {

if segue.identifier == "ShowNotesSegue" {

let noteVC = segue.destinationViewController as! NoteViewController

noteVC.notebook = notebooks[tableView.indexPathForSelectedRow!.row]

}

}

https://gist.github.com/imrekel/379d8fdf937011ed7031#file-notebookviewcontroller-prepareforsegue-swift

Váltsunk a NoteViewController.swift-re és vegyünk fel egy NSFetchedResultsController típusú property-t:

var fetchedResultsController: NSFetchedResultsController!

A viewDidLoad() metódusban hozzuk létre a Note-okat visszaadó lekérdezést és rendeljük egy újonnan létrehozott NSFetchedResultsController-hez:

override func viewDidLoad() {

super.viewDidLoad()

self.navigationItem.title = notebook.title

let moc = AppDelegate.sharedAppDelegate.managedObjectContext

let fetchRequest = NSFetchRequest(entityName: "Note")

// szűrés azon Note-okra, melyek a kiválasztott Notebook-hoz tartoznak

let predicate = NSPredicate(format: "notebook == %@", notebook)

fetchRequest.predicate = predicate

// rendezés creationDate szerint, csökkenő sorrendben

let sortDescriptor = NSSortDescriptor(key: "creationDate",   
 ascending: false)

fetchRequest.sortDescriptors = [sortDescriptor]

// egyszerre max 30 Note lekérdezése

fetchRequest.fetchBatchSize = 30

fetchedResultsController = NSFetchedResultsController(  
 fetchRequest: fetchRequest, managedObjectContext: moc,   
 sectionNameKeyPath: nil, cacheName: nil)

try! fetchedResultsController.performFetch()

}

https://gist.github.com/imrekel/379d8fdf937011ed7031#file-noteviewcontroller-viewdidload-swift

A UITableViewDataSource metódusoknál töröljük ki a szekciók számát megadót, a sorok számánál pedig térjünk vissza a Fetched Results Controllertől elkért értékkel:

override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

let sectionInfo = fetchedResultsController.sections![section] as   
 NSFetchedResultsSectionInfo

return sectionInfo.numberOfObjects

}

A cellForRowAtIndexPath()-ban szintén a controllertől kérjük el a megfelelő indexű Note-ot és ez alapján konfiguráljuk a cellát. Vegyünk fel egy külön metódust a cella adatainak beállításához (később még ennek hasznát vehetjük, ha már egy létező cellát akarunk frissíteni):

func configureCell(cell: UITableViewCell, atIndexPath indexPath: NSIndexPath) {  
 let note = fetchedResultsController.objectAtIndexPath(indexPath) as! Note

cell.textLabel?.text = note.content

let dateFormatter = NSDateFormatter()

dateFormatter.dateStyle = .MediumStyle

dateFormatter.timeStyle = .MediumStyle

cell.detailTextLabel?.text =   
 dateFormatter.stringFromDate(note.creationDate!)

}

https://gist.github.com/imrekel/379d8fdf937011ed7031#file-notebookviewcontroller-configurecell-swift

override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {

let cell = tableView.dequeueReusableCellWithIdentifier("NoteCell",   
 forIndexPath: indexPath) as UITableViewCell

configureCell(cell, atIndexPath: indexPath)

return cell

}

Próbáljuk ki az alkalmazást és ellenőrizzük, hogy megjelennek-e a Note-ok a Notebook-okon belül!

## Jegyzetek felvétele

Definiáljunk egy createNoteWithContent() nevű metódust, mely létrehoz egy új jegyzetet és hozzárendeli az aktuális Notebook-hoz:

func createNoteWithContent(content: String) {

let moc = AppDelegate.sharedAppDelegate.managedObjectContext

let entity = NSEntityDescription.entityForName("Note",

inManagedObjectContext: moc)

let note = Note(entity: entity!,

insertIntoManagedObjectContext:moc)

note.content = content

note.creationDate = NSDate()

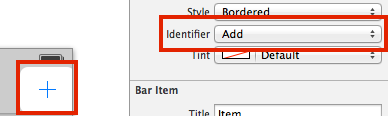
note.notebook = notebook

AppDelegate.sharedAppDelegate.saveContext()

}

https://gist.github.com/imrekel/379d8fdf937011ed7031#file-noteviewcontroller-createnotewithcontent-swift

A storyboardban vegyünk fel egy Bar Button Item-et a Note View Controllerre a Navigation Bar jobb szélére:



Kössünk be hozzá egy addNoteButtonTap() akció metódust, melyben jelenítsünk meg egy alert controllert az új jegyzet szövegének bekéréséhez:

@IBAction func addNoteButtonTap(sender: AnyObject) {

let createNoteAlert = UIAlertController(title: "Create Note", message:   
 "Enter the content", preferredStyle: .Alert)

createNoteAlert.addTextFieldWithConfigurationHandler { textField in

textField.placeholder = "Note content"

}

let cancelAction = UIAlertAction(title: "Cancel", style: .Cancel,   
 handler: nil)

createNoteAlert.addAction(cancelAction)

let createAction = UIAlertAction(title: "Create", style: .Default) {   
 action in

let textField = createNoteAlert.textFields!.first! as UITextField

self.createNoteWithContent(textField.text!)

}

createNoteAlert.addAction(createAction)

presentViewController(createNoteAlert, animated: true, completion: nil)

}

https://gist.github.com/imrekel/379d8fdf937011ed7031#file-noteviewcontroller-addnotebuttontap-swift

Ha kipróbáljuk az alkalmazást, azt láthatjuk, hogy nem jelenik meg az új jegyzet a mentés után. Ez azért van mert a Table View még nem értesül a kontextus módosításáról. Ekkor tűnik fel a képben az NSFetchedResultsController, mely képes értesítéseket küldeni, ha megváltozik az általa figyelt lekérdezés, esetünkben azok a Note objektumok, amik az éppen kiválasztott Notebook-hoz tartoznak.

Adjuk hozzá NoteViewController-hez az **NSFetchedResultsControllerDelegate** protocolt:

class NoteViewController: UITableViewController,   
 NSFetchedResultsControllerDelegate {

A viewDidLoad: metódusban állítsuk be a delegate-et:

fetchedResultsController.delegate = self

És valósítsuk meg a protokoll következő 3 műveletét:

func controllerWillChangeContent(controller: NSFetchedResultsController) {

tableView.beginUpdates()

}

func controllerDidChangeContent(controller: NSFetchedResultsController) {

tableView.endUpdates()

}

func controller(controller: NSFetchedResultsController, didChangeObject anObject: AnyObject, atIndexPath indexPath: NSIndexPath?, forChangeType type: NSFetchedResultsChangeType, newIndexPath: NSIndexPath?) {

switch type {

case .Insert:

tableView.insertRowsAtIndexPaths([newIndexPath!],   
 withRowAnimation: .Fade)

default:

break

}

}

https://gist.github.com/imrekel/379d8fdf937011ed7031#file-noteviewcontroller-nsfetchedresultscontrollerdelegate-swift

## Jegyzetek törlése

Engedélyezzük a "Swipte to delete" funkciót a canEditRowAtIndePath() metódussal:

override func tableView(tableView: UITableView, canEditRowAtIndexPath indexPath: NSIndexPath) -> Bool {

return true

}

Majd végezzük el a törlést a UITableViewDataSource-ból adoptált tableView(commitEditingStyle) metódusban:

override func tableView(tableView: UITableView, commitEditingStyle editingStyle: UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {

if editingStyle == .Delete {

let moc = AppDelegate.sharedAppDelegate.managedObjectContext

let noteToDelete =   
 fetchedResultsController.objectAtIndexPath(indexPath) as! Note

moc.deleteObject(noteToDelete)

}

}

https://gist.github.com/imrekel/379d8fdf937011ed7031#file-noteviewcontroller-tableviewcommiteditingstyle-swift

A törlésről értesül a fetched results controller és meghívja az előbb bemutatott delegate metódust, ebben vegyük fel a törlés eseményhez, hogy a table view kitörölje a megfelelő elemet:

case .Delete:

tableView.deleteRowsAtIndexPaths([indexPath!], withRowAnimation: .Fade)

## További műveletek

A fetched results controller jelez ha bármelyik objektum módosításra kerül (pl. átírták egy attribútumát) vagy ha megváltozik a pozíciója a lekérdezésen belül. Bár ezeket a laboron nem használjuk, érdemes a standard implementációt ezekhez is felvenni:

case .Update:

let cell = tableView.cellForRowAtIndexPath(indexPath!)!

configureCell(cell, atIndexPath: indexPath!)

case .Move:

tableView.deleteRowsAtIndexPaths([indexPath!], withRowAnimation: .Fade)

tableView.insertRowsAtIndexPaths([newIndexPath!],   
 withRowAnimation: .Fade)

## Managed Object Context mentése

A jelenlegi implementáció rögtön elmenti a változásokat a perzisztens tárolóba. Sok módosítás esetén (vagy ha esetleg szeretnénk érvényteleníteni a legutóbbi változtatásokat), érdemes lehet a módosításokat csak akkor menteni, mikor az alkalmazás a háttérbe kerül.

Ehhez AppDelegate-ben módosítsuk az applicationDidEnterBackground() metódust:

func applicationDidEnterBackground(application: UIApplication) {

saveContext()

}

# Önálló feladatok

Építsük be NotebookViewController-be is új Notebook-ok felvételének és törlésének lehetőségét!