THE UNIVERSITY OF CHICAGO

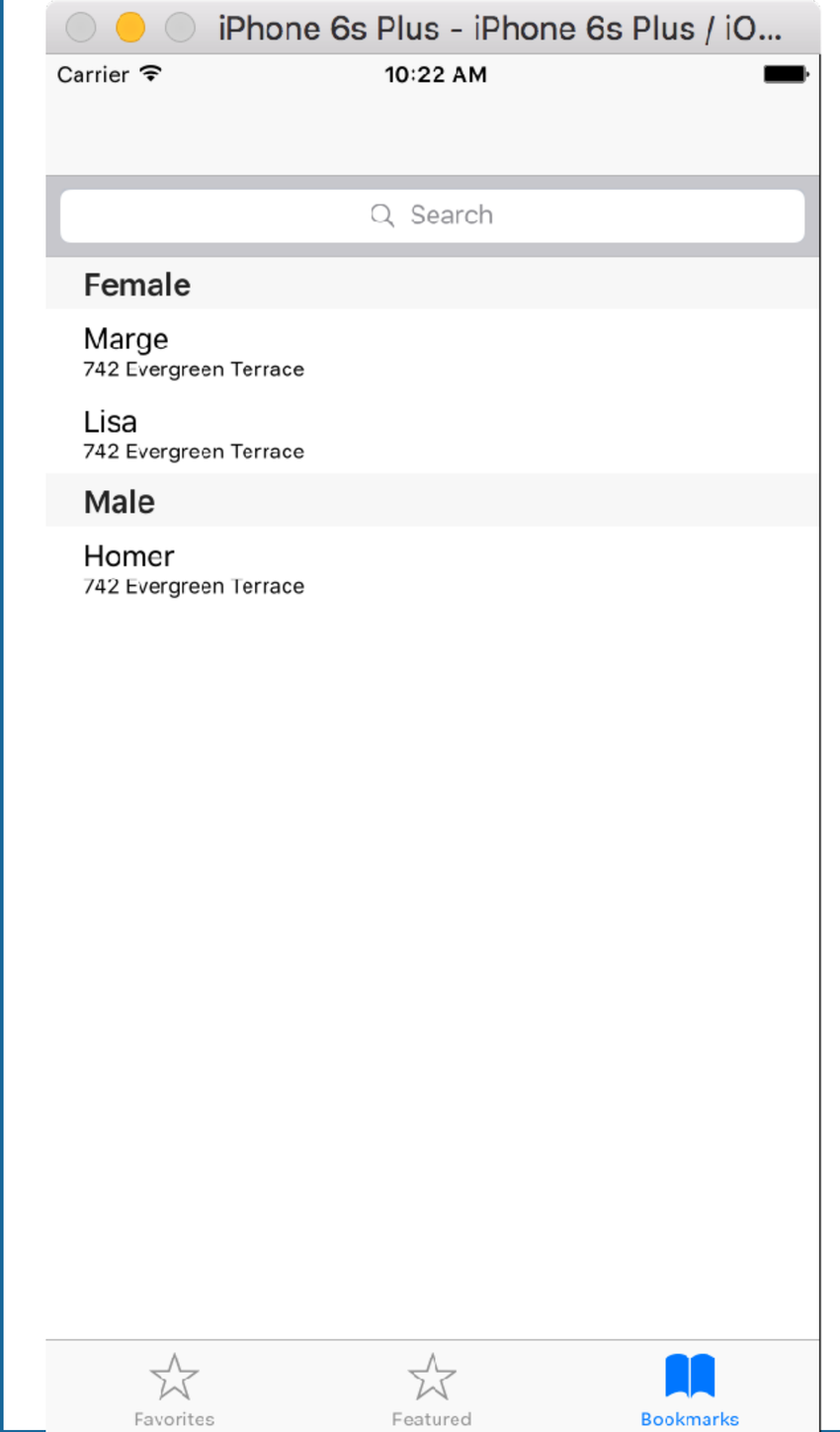
WATCH APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 5

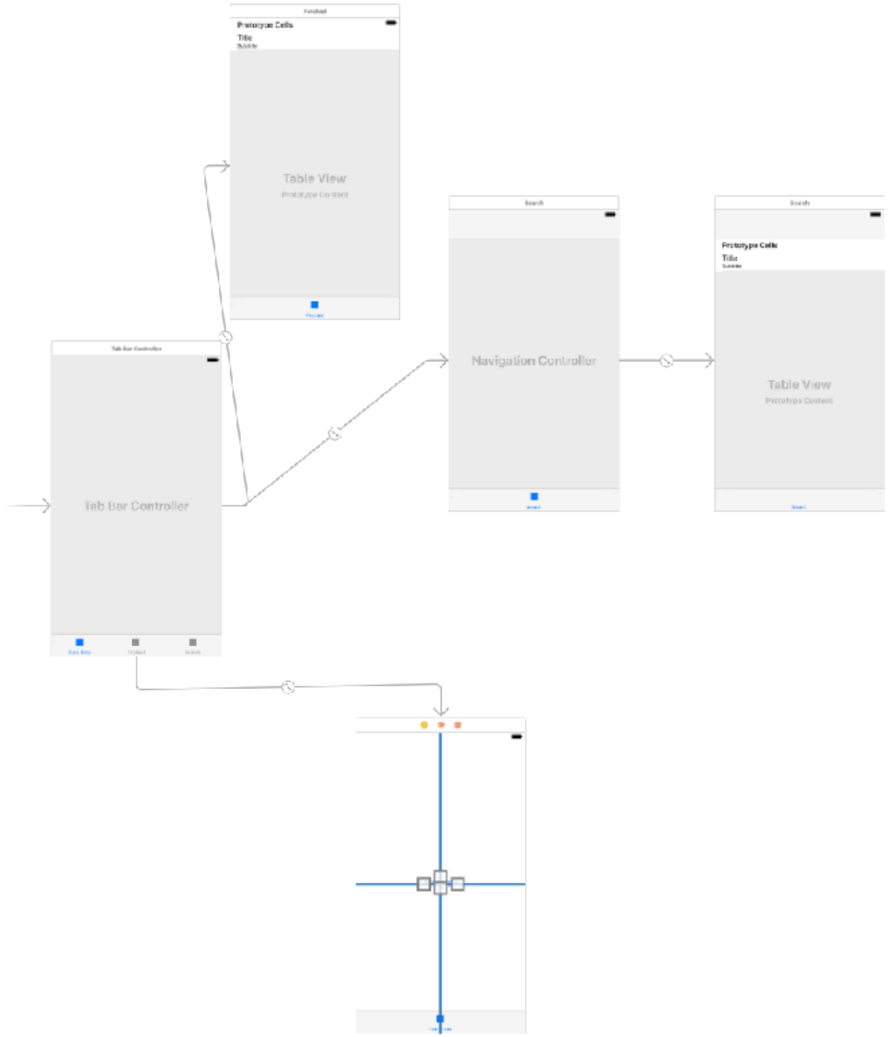
A SIMPLE CORE DATA APPLICATION

A SIMPLE CORE DATA APPLICATION

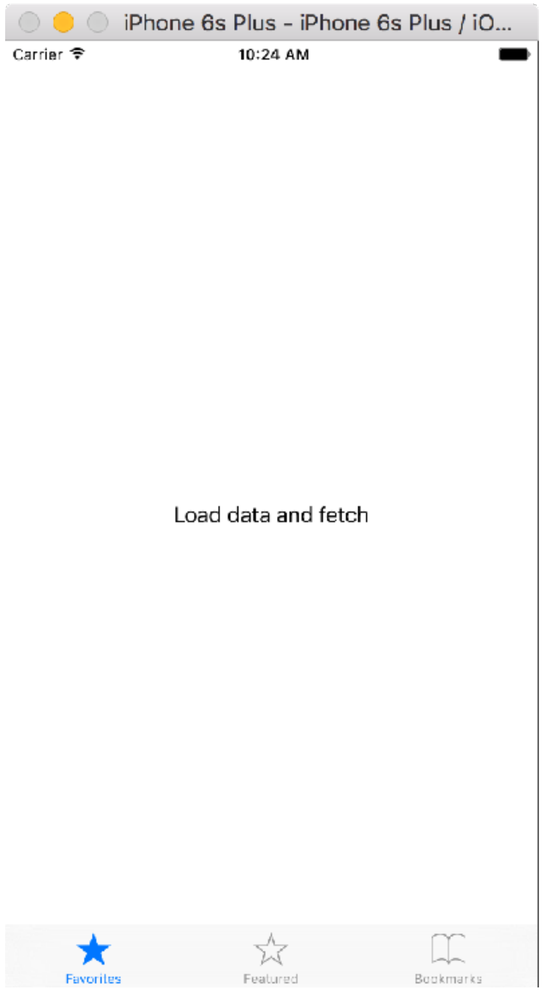
- Session tracker application
  - Session and User model
  - Load and fetch data
  - Show in table with NSFetchedResultsController
  - Use UISearchBar to live filter results

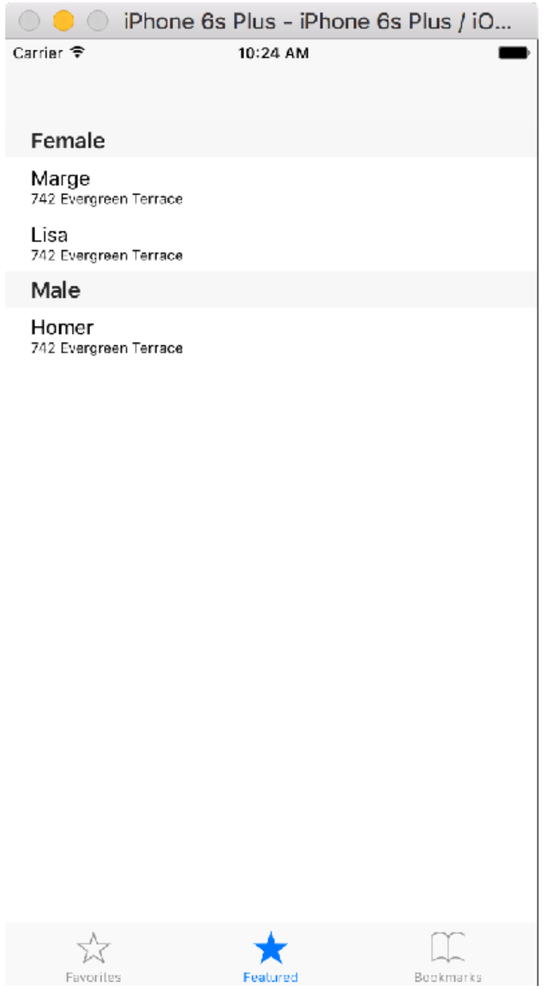


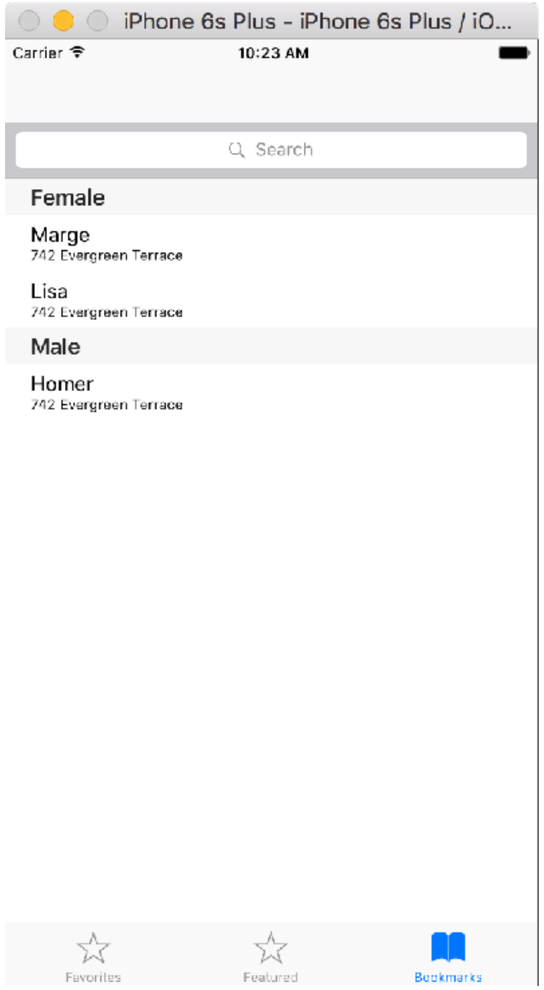
A SIMPLE CORE DATA APPLICATION



A SIMPLE CORE DATA APPLICATION

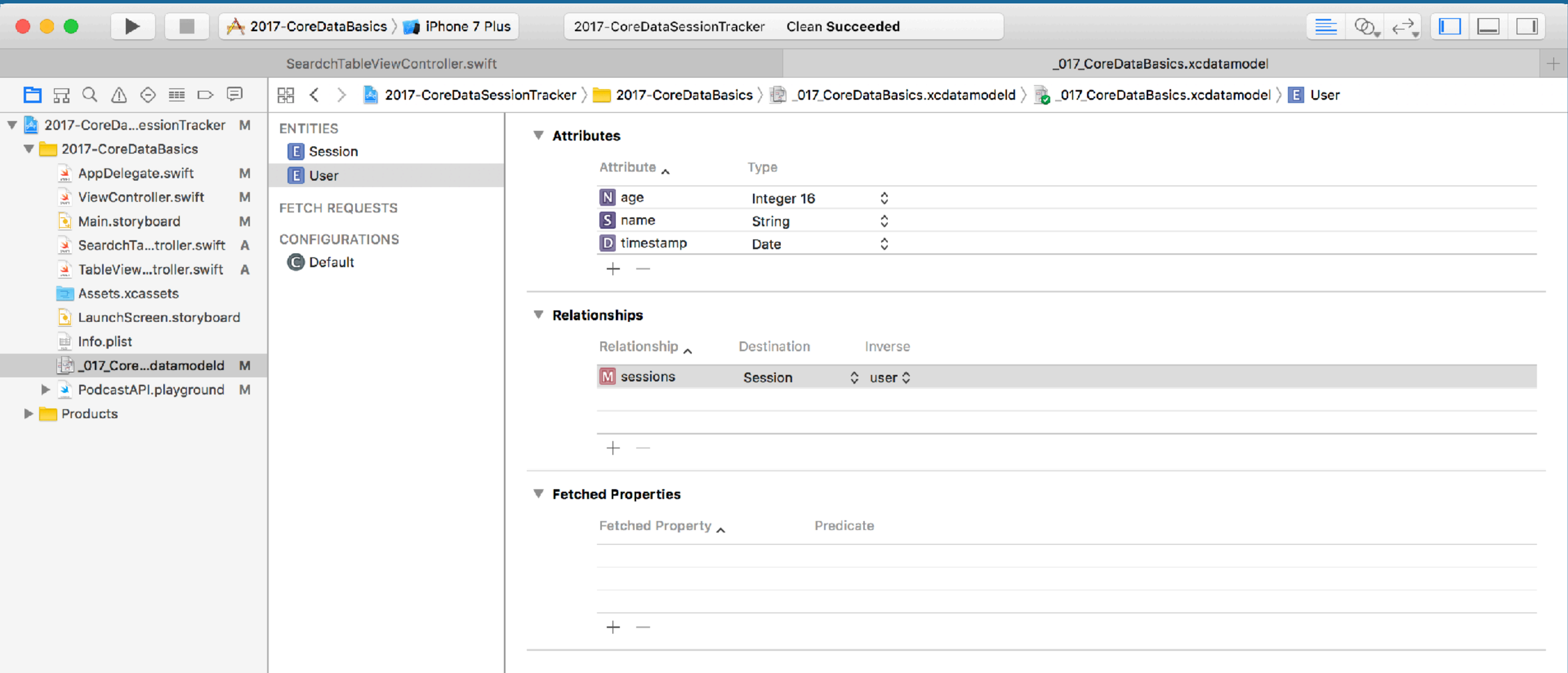




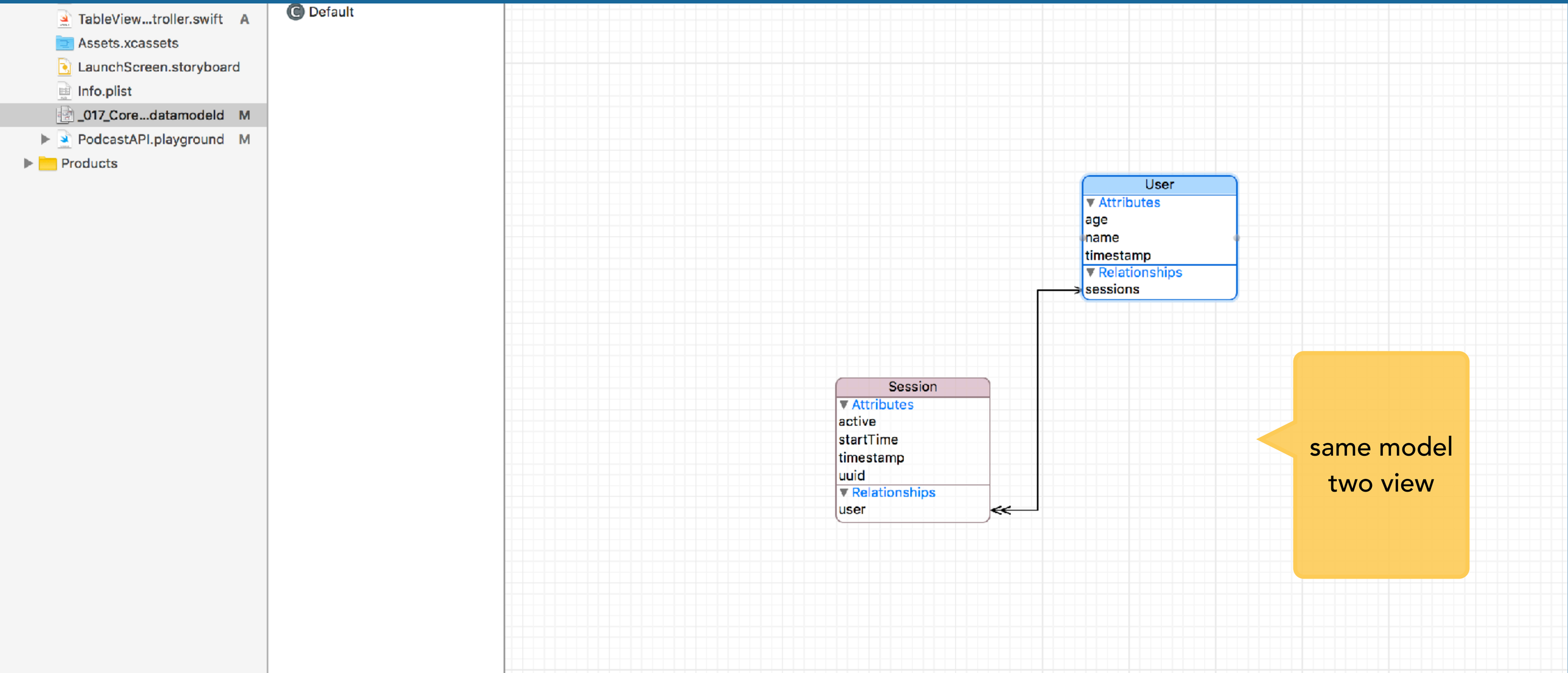


DATA MODEL

DATA MODEL

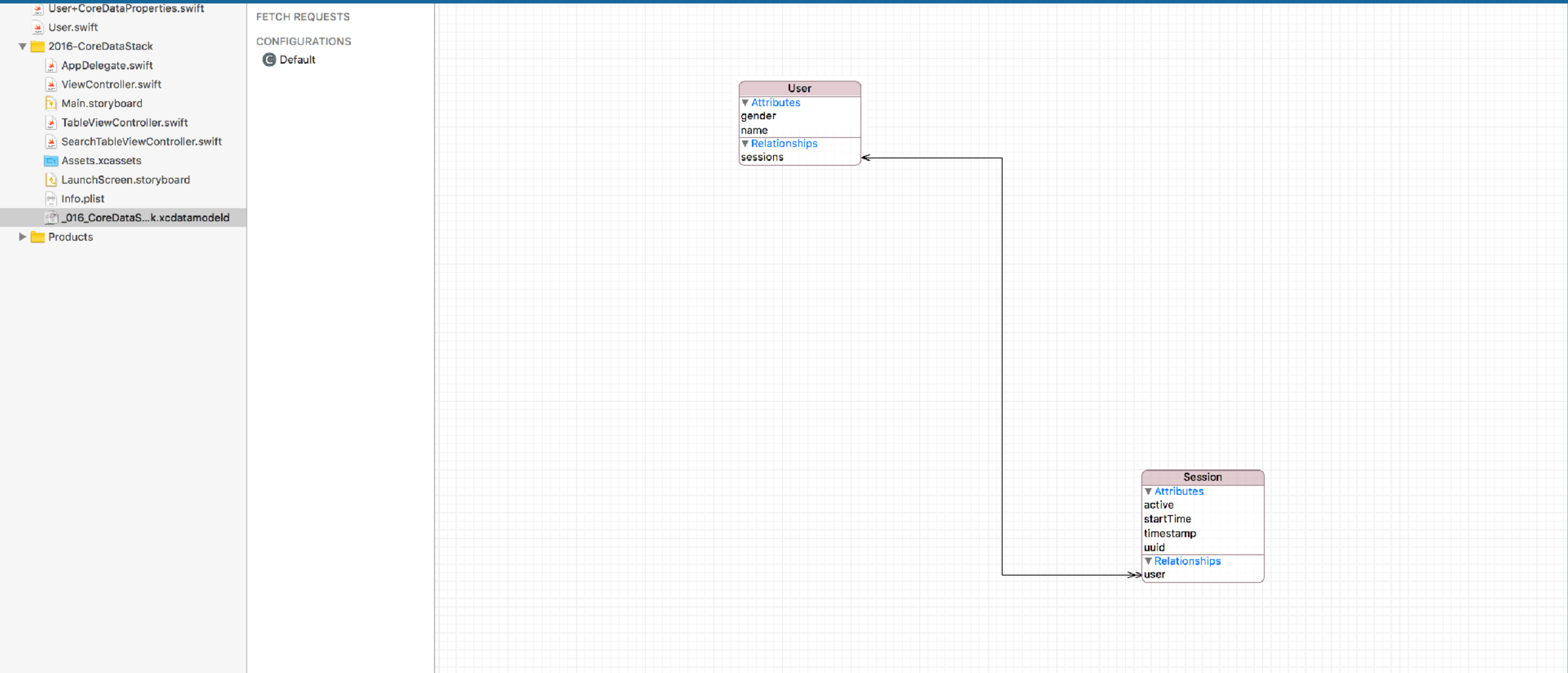

2017-CoreDataBasics
iPhone 7 Plus
2017-CoreDataSessionTracker
Clean Succeeded
SeardchTableViewController.swift
_017_CoreDataBasics.xcdatamodel
2017-CoreDataSessionTracker
2017-CoreDataBasics
_017_CoreDataBasics.xcdatamodeld
_017_CoreDataBasics.xcdatamodel
User
2017-CoreDa...essionTracker M
2017-CoreDataBasics
AppDelegate.swift M
ViewController.swift M
Main.storyboard M
SeardchTa...troller.swift A
TableView...troller.swift A
Assets.xcassets
LaunchScreen.storyboard
Info.plist
_017_Core...datamodeld M
PodcastAPI.playground M
Products
ENTITIES
Session
User
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
age
Integer 16
name
String
timestamp
Date
Relationships
Relationship
Destination
Inverse
sessions
Session
user
Fetched Properties
Fetched Property
Predicate

DATA MODEL

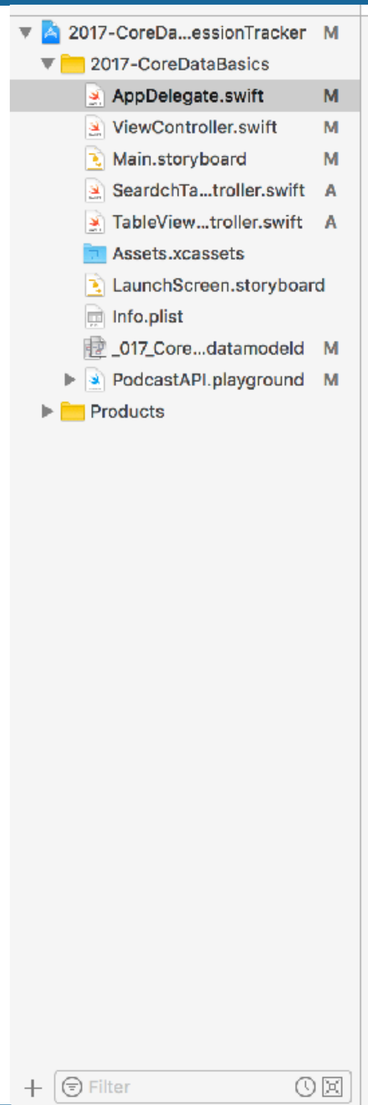

TableView...troller.swift A
Assets.xcassets
LaunchScreen.storyboard
Info.plist
_017_Core...datamodeld M
PodcastAPI.playground M
Products
Default
User
Attributes
age
name
timestamp
Relationships
sessions
Session
Attributes
active
startTime
timestamp
uuid
Relationships
user
same model
two view

ADDING DATA

ADDING DATA


User+CoreDataProperties.swift
User.swift
2016-CoreDataStack
AppDelegate.swift
ViewController.swift
Main.storyboard
TableViewController.swift
SearchTableViewController.swift
Assets.xcassets
LaunchScreen.storyboard
Info.plist
_016_CoreDataS...k.xcdatamodeld
Products
FETCH REQUESTS
CONFIGURATIONS
Default
User
Attributes
gender
name
Relationships
sessions
Session
Attributes
active
startTime
timestamp
uuid
Relationships
user

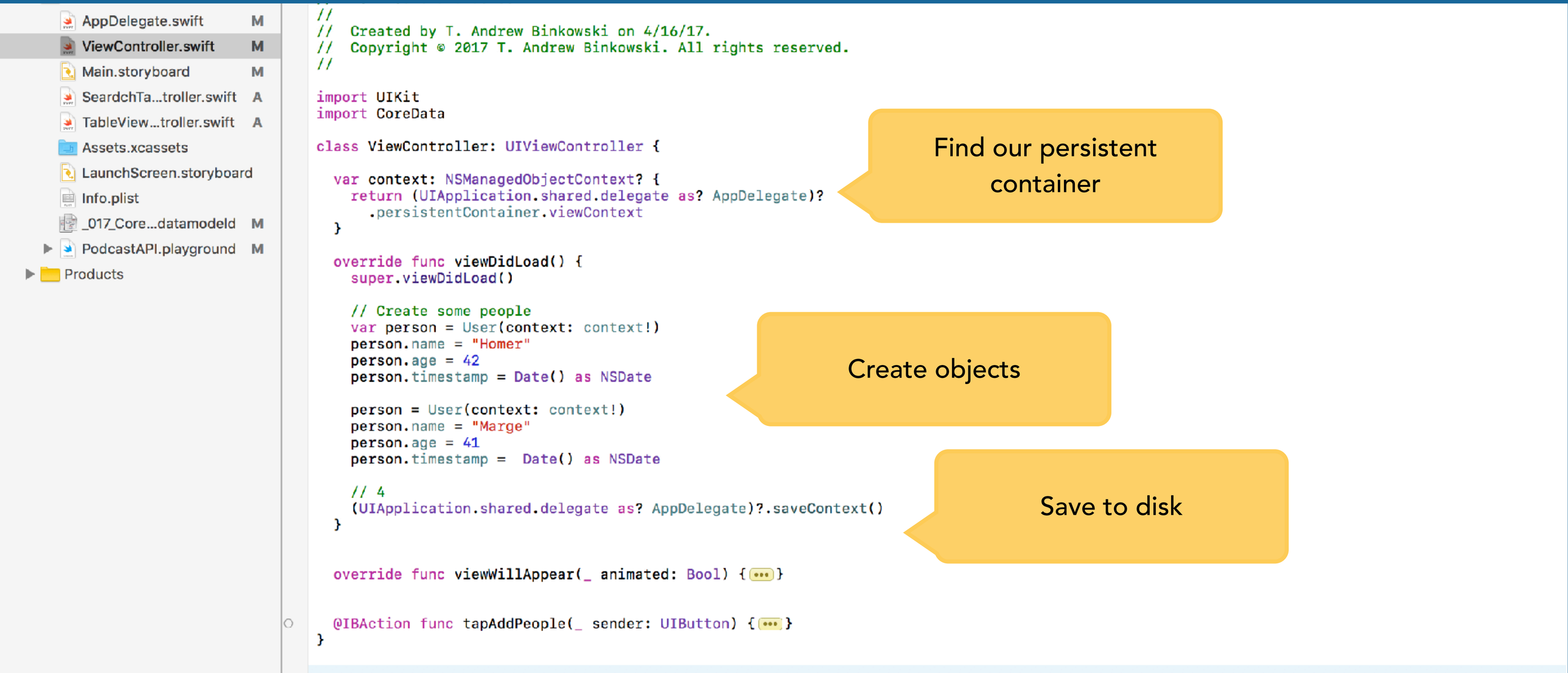
ADDING DATA




```swift
func applicationWillTerminate(_ application: UIApplication) {...}

// MARK: - Core Data stack

lazy var persistentContainer: NSPersistentContainer = {
    /*
     The persistent container for the application. This implementation
     creates and returns a container, having loaded the store for the
     application to it. This property is optional since there are legitimate
     error conditions that could cause the creation of the store to fail.
    */
    let container = NSPersistentContainer(name: "_017_CoreDataBasics")
    container.loadPersistentStores(completionHandler: { (storeDescription, error) in
        if let error = error as NSError? {
            // Replace this implementation with code to handle the error appropriately.
            // fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping
                application, although it may be useful during development.

            /*
             Typical reasons for an error here include:
             * The parent directory does not exist, cannot be created, or disallows writing.
             * The persistent store is not accessible, due to permissions or data protection when the device is locked.
             * The device is out of space.
             * The store could not be migrated to the current model version.
             Check the error message to determine what the actual problem was.
             */
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()

// MARK: - Core Data Saving support

func saveContext () {
    print("Saving...")
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
            dump()


        } catch {
            // Replace this implementation with code to handle the error appropriately.
            // fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping
                application, although it may be useful during development.
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

app delegate

ADDING DATA


AppDelegate.swift M
ViewController.swift M
Main.storyboard M
SeardchTa...troller.swift A
TableView...troller.swift A
Assets.xcassets
LaunchScreen.storyboard
Info.plist
_017_Core...datamodeld M
PodcastAPI.playground M
Products
//
// Created by T. Andrew Binkowski on 4/16/17.
// Copyright © 2017 T. Andrew Binkowski. All rights reserved.
//
import UIKit
import CoreData
class ViewController: UIViewController {
var context: NSManagedObjectContext? {
return (UIApplication.shared.delegate as? AppDelegate)?
.persistentContainer.viewContext
}
override func viewDidLoad() {
super.viewDidLoad()
// Create some people
var person = User(context: context!)
person.name = "Homer"
person.age = 42
person.timestamp = Date() as NSDate
person = User(context: context!)
person.name = "Marge"
person.age = 41
person.timestamp = Date() as NSDate
// 4
(UIApplication.shared.delegate as? AppDelegate)?.saveContext()
}
override func viewWillAppear(_ animated: Bool) {...}
@IBAction func tapAddPeople(_ sender: UIButton) {...}
}
Find our persistent container
Create objects
Save to disk

ADDING DATA

```
override func viewDidLoad() {
  super.viewDidLoad()

  // Create some people
  let user1 = User(context: context!)
  user1.name = "Homer"
  user1.age = 42
  user1.timestamp = Date() as NSDate

  // Create some people
  let user2 = User(context: context!)
  user2.name = "Marge"
  user2.age = 42
  user2.timestamp = Date() as NSDate

  // Let's create a bunch sessesions and add them to difference users
  for i in 0...10 {
    let session = Session(context: context!)
    // Just for fun, make the even sessions true
    session.active = (i % 2 == 0) ? true : false
    session.timestamp = Date() as NSDate
    session.user = user2
    session.user = user1
  }

  // Save
  (UIApplication.shared.delegate as? AppDelegate)?.saveContext()
}
```

ADD SOME RANDOM Sessions

ADDING DATA

```
@IBAction func tapAddPeople(_ sender: UIButton) {

  let person = User(context: context!)
  person.name = "Random Person"
  person.age = 42
  person.timestamp = Date() as NSDate

  let session = Session(context: context!)
  session.active = true
  session.timestamp = Date() as NSDate
  session.user = person

  (UIApplication.shared.delegate as? AppDelegate)?.saveContext()
}
```

ADD SOME RANDOM DATA

ADDING DATA

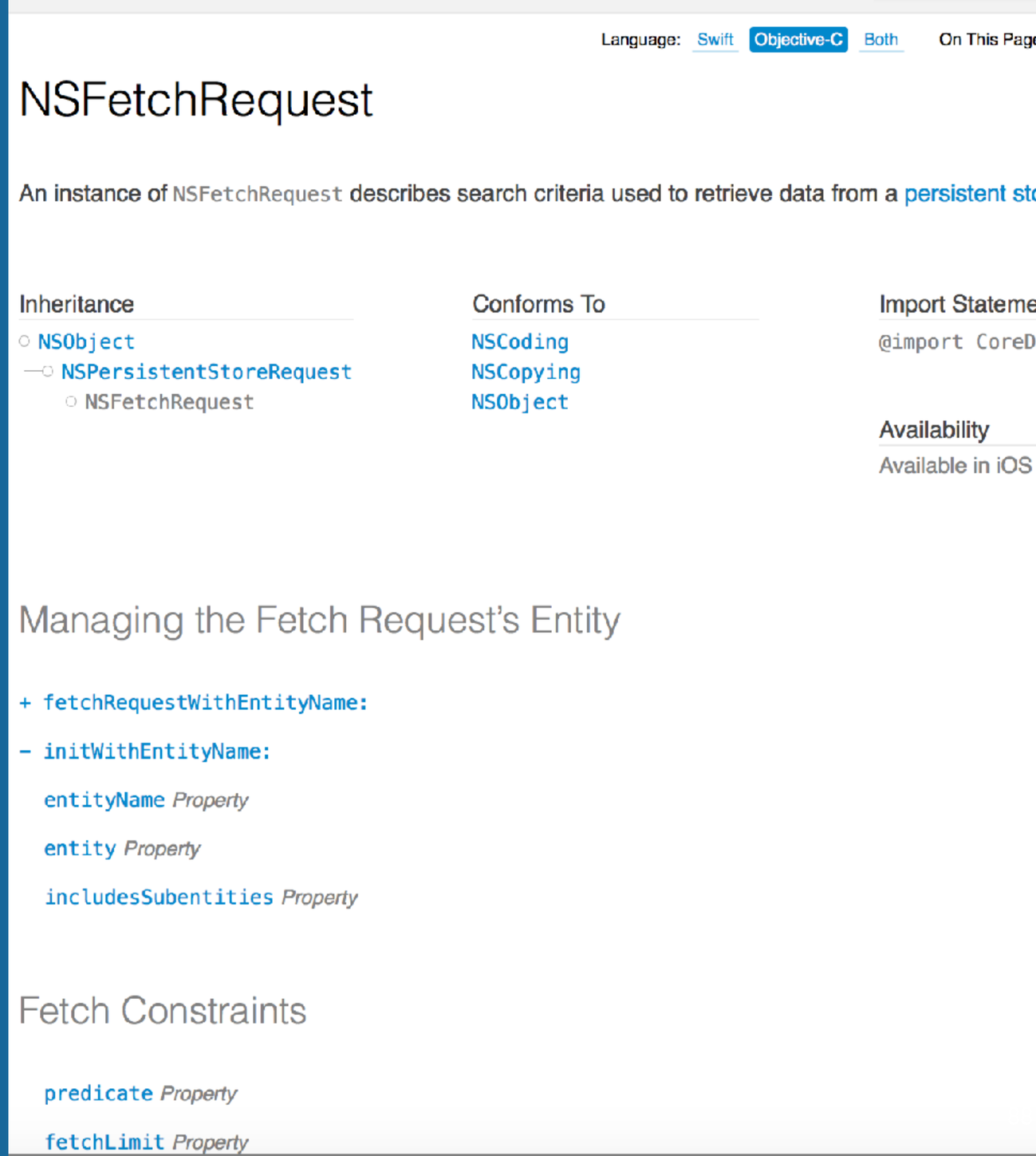
```
func saveContext () {
    print("Saving...")
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
            dump()


        } catch {
            // Replace this implementation with code to handle the error appropriat
            // fatalError() causes the application to generate a crash log and term
                application, although it may be useful during development.
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

FETCHING DATA

FETCHING DATA

SUBTITLE

- Fetch managed objects by sending a request to managed object context
- Use the NSFetchedData method
- Optional
  - Set a predicate (NSPredicate)
  - Set a sort descriptor (NSSortDescriptor)



FETCHING DATA

```swift
// Fetch our person objects
let personFetchRequest: NSFetchRequest<User> = User.fetchRequest()
do {
  let fetchedEntities = try context?.fetch(personFetchRequest)
  // Log it out
  for user in fetchedEntities! {
    print("> \(user.name!) ##########")
    for session in user.sessions! {
      print(session)
    }
  }

} catch {
  // Do something in respon
}
```

```
> Homer ##########
<Session: 0x610000094eb0> (entity: Session; id: 0xd000000000c00000 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/Session/p48> ; data: {
    active = 0;
    startTime = nil;
    timestamp = "2017-04-17 17:23:01 +0000";
    user = "0xd000000000340002 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/User/p13>";
    uuid = nil;
})
<Session: 0x610000097340> (entity: Session; id: 0xd000000000c40000 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/Session/p49> ; data: {
    active = 1;
    startTime = nil;
    timestamp = "2017-04-17 17:23:01 +0000";
    user = "0xd000000000340002 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/User/p13>";
    uuid = nil;
})
<Session: 0x6100000977a0> (entity: Session; id: 0xd000000000a40000 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/Session/p41> ; data: {
    active = 1;
    startTime = nil;
    timestamp = "2017-04-17 17:23:01 +0000";
    user = "0xd000000000340002 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/User/p13>";
    uuid = nil;
})
<Session: 0x610000095f90> (entity: Session; id: 0xd000000000c80000 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/Session/p50> ; data: {
    active = 0;
    startTime = nil;
```

FETCHING DATA

```
// Fetch our person objects
let personFetchRequest: NSFetchRequest<User> = User.fetchRequest()
do {
  let fetchedEntities = try context?.fetch(personFetchRequest)
  // Log it out
  for user in fetchedEntities! {
    print("> \(user.name!) ##########")
    for session in user.sessions! {
      print(session)
    }
  }

} catch {
  // Do something in response to error condition
}
```

setup fetch request

execute fetch

our in-memory
data store

FETCHING DATA

```
let fetchRequest = NSFetchRequest<User>(entityName: "User")
fetchRequest.fetchBatchSize = 10
let dateDescriptor = NSSortDescriptor(key: "timestamp", ascending: false)
fetchRequest.sortDescriptors = [dateDescriptor]
```

- Set sort descriptors for returned results
- Takes array of sort descriptors as arguments

FETCHING DATA

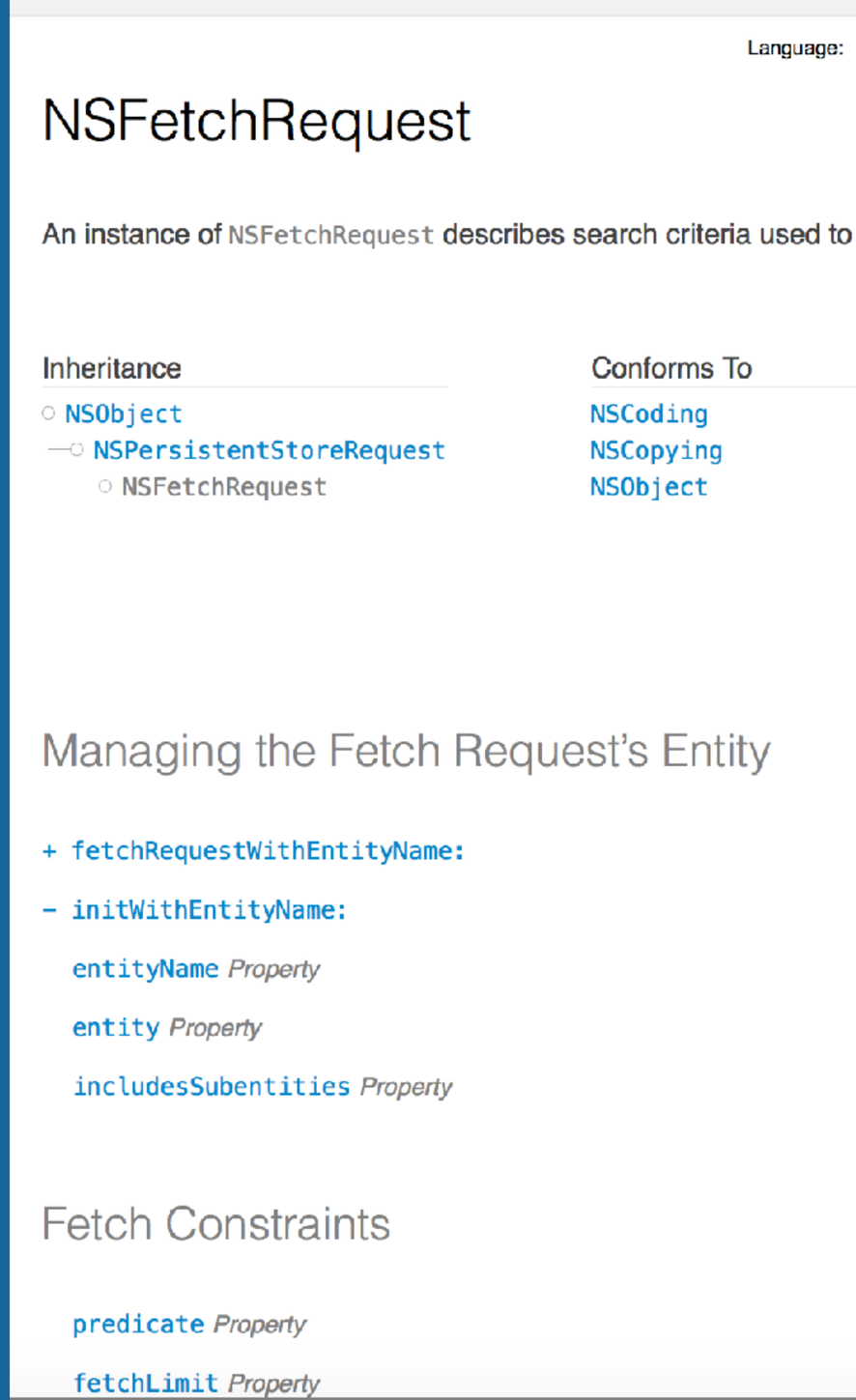
```
// Print data store to the console
func dump() {

  let personFetchRequest: NSFetchRequest<User> = User.fetchRequest()
  personFetchRequest.fetchLimit = 20
  personFetchRequest.relationshipKeyPathsForPrefetching = ["session"]
  personFetchRequest.propertiesToFetch = ["timestamp"]

  do {
    let context = persistentContainer.viewContext

    let fetchedEntities = try context.fetch(personFetchRequest)
    print("Dump")
    for user in fetchedEntities {
      print("# \(user.name!) ##########")
      for session in user.sessions! {
        print(session)
      }
    }

  } catch {
    // Do something in response to error condition
  }

}
```

prepare for efficiency

limit for efficiency

FETCHING DATA

- NSPredicate defines logical conditions
  - Constrain a search
  - In-memory filtering
- Examples
  - grade == "7"
  - firstName like "Bob"
  - name CONTAINS[cd] “son”
  - (firstName begins with 'M') AND (lastName like 'Adderley'



Language:

NSFetchRequest

An instance of NSFetchRequest describes search criteria used to

Inheritance

- NSObject
  - NSPersistentStoreRequest
    - NSFetchRequest

Conforms To

- NSCoding
- NSCopying
- NSObject

Managing the Fetch Request’s Entity

+ fetchRequestWithEntityName:

- initWithEntityName:

entityName *Property*

entity *Property*

includesSubentities *Property*

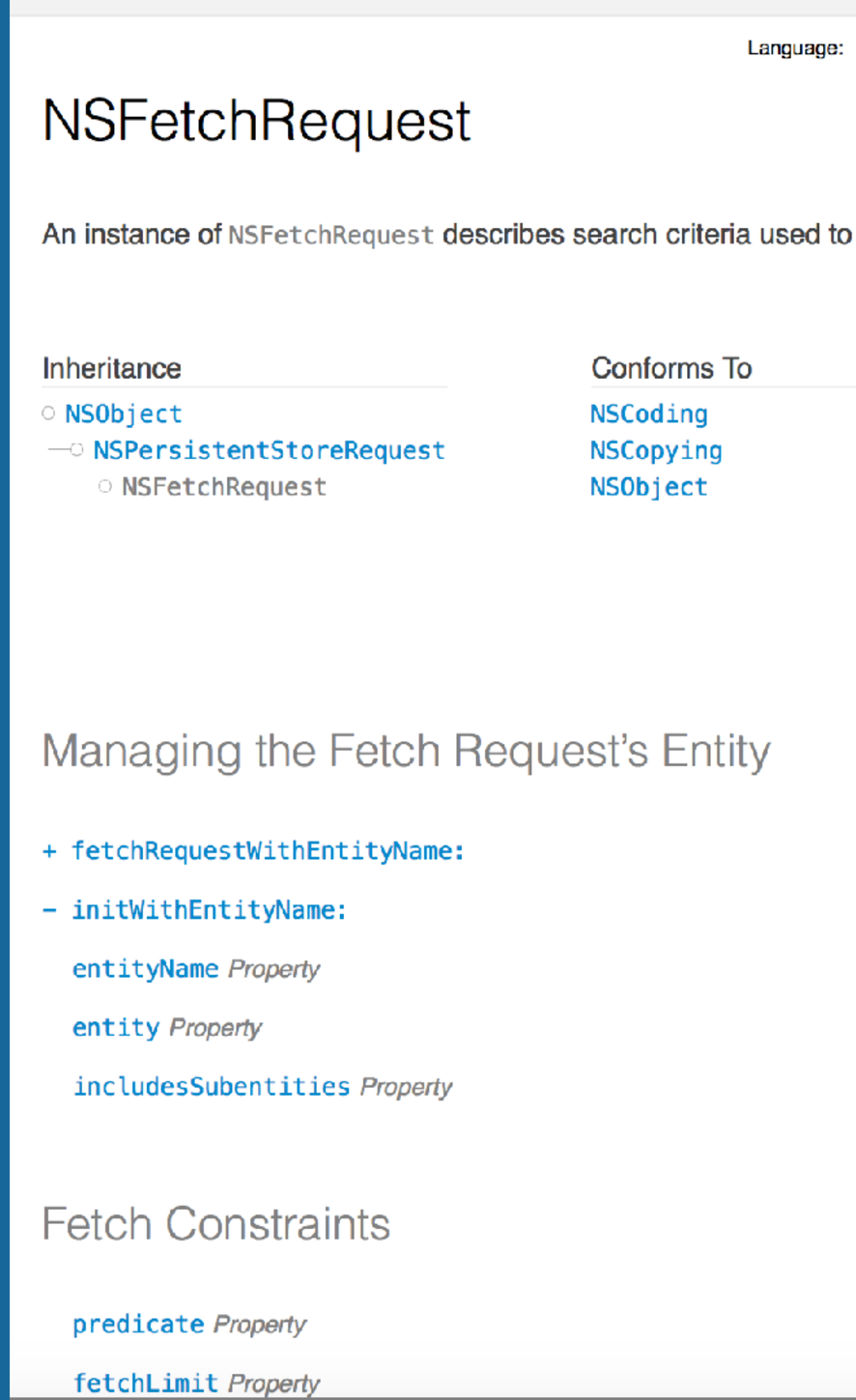
Fetch Constraints

predicate *Property*

fetchLimit *Property*

FETCHING DATA

- NSPredicate can be applied to NSArrays, NSDictionary and NSFetchRequest
- Predicate format string syntax
  - Regular expression
- NSCompoundPredicate
  - Chain multiple predicates together



FETCHING DATA

```swift
let personFetchRequest: NSFetchRequest<User> = User.fetchRequest()
personFetchRequest.fetchLimit = 20
personFetchRequest.relationshipKeyPathsForPrefetching = ["session"]
personFetchRequest.propertiesToFetch = ["timestamp"]

// Find exact match only
personFetchRequest.predicate = NSPredicate(format: "firstName == %@ && age == %@", argumentArray: ["Homer",42])

// Fetch everything
personFetchRequest.predicate = NSPredicate(format: "TRUEPREDICATE")

// Fetch matching string
personFetchRequest.predicate = NSPredicate(format: "name CONTAINS[cd] %@", text)
```

- Predicate examples

DELETING DATA

DELETING DATA

- Fetch the objects
- Iterate through them and delete them
- iOS8 new feature
  - Batch delete
- Save the managed object context
  - Deleting removes from memory
  - Commit the changes to save to the on-disk store

```
let personFetchRequest: NSFetch
personFetchRequest.fetchLimit =
personFetchRequest.relationship
personFetchRequest.propertiesTo

// Fetch everything
personFetchRequest.predicate =

do {
  let context = persistentConta

  let fetchedEntities = try con
  print("Dump")
  for user in fetchedEntities {
    print("# \(user.name!) ####
    for session in user.session
      print(session)
    }
    // You could delete the use
    context.delete(user)

  }

} catch {
  // Do something in response t
}
```

DELETING DATA

- Fetch the objects
- Iterate through them and delete them
- iOS8 new feature
  - Batch delete
- Save the managed object context
  - Deleting removes from memory
  - Commit the changes to save to the on-disk store

```
chRequest: NSFetchRequest<User> = User.fetchRequest()
quest.fetchLimit = 20
quest.relationshipKeyPathsForPrefetching = ["session"]
quest.propertiesToFetch = ["timestamp"]

ything
quest.predicate = NSPredicate(format: "TRUEPREDICATE")


 = persistentContainer.viewContext

Entities = try context.fetch(personFetchRequest)
")
 fetchedEntities {
\(user.name!) ##########")
on in user.sessions! {
ession)

uld delete the user
elete(user)



hing in response to error condition
```

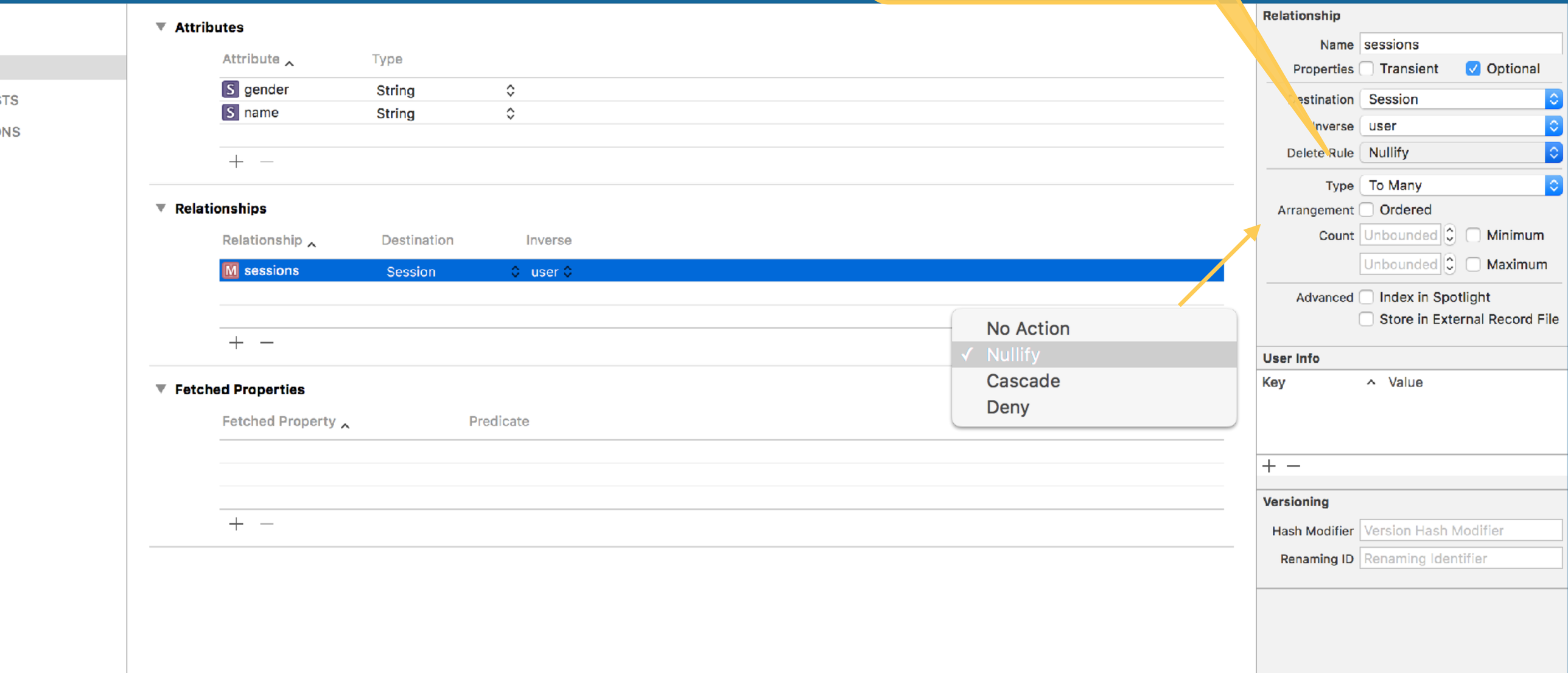
DELETING DATA

```
// Fetch everything
personFetchRequest.predicate = NSPredicate(format: "TRUEPREDICATE")

do {
  let context = persistentContainer.viewContext

  let fetchedEntities = try context.fetch(personFetchRequest)
  print("Dump")
  for user in fetchedEntities {
    print("# \(user.name!) ##########")
    for session in user.sessions! {
      print(session)
    }
    // You could delete the user
    context.delete(user)

  }
```

- Delete managed objects
  - Fetch the objects
  - Iterate through them and delete them

DELETING DATA

Cascading deletes



CORE DATA IN A TABLE

CORE DATA AND TABLES

- Core Data can be tightly integrated with tables views
  - Reduce memory overhead
  - Improve response times
- Loads data for cells being shown
- NSFectchedResultsController has direct methods for creating
  - Index path access
  - Section key path
  - Section groups
  - Index titles

NSFetchedResultsController Cla

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreData.f
Availability	Available in iOS 3.0 and later.
Companion guide	Core Data Programming Guide
Declared in	NSFetchedResultsController.h
Related sample code	CoreDataBooks DateSectionTitles iPhoneCoreDataRecipes TopSongs

Overview

You use a fetched results controller to efficiently manage the resul
`UITableView` object.

While table views can be used in several ways, this object is primar
expects its data source to provide cells as an array of sections mad
fetch request that specifies the entity, an array containing at least
`NSFetchedResultsController` efficiently analyzes the result of t
in the result set, and for the index.

In addition, `NSFetchedResultsController` provides the followin

- It optionally monitors changes to objects in its associated mana
  delegate (see "The Controller's Delegate").
- It optionally caches the results of its computation so that if the
  to be repeated (see "The Cache").

A controller thus effectively has three modes of operation, determi
is set.

CORE DATA AND TABLES

- NSFetchedResultsController managed results from Core Data fetch for UITableView object
  - Provides the “dataSource” for the UITableView
- Create an instance of NSFetchedResultsController
  - Entity
  - NSArray of sort ordering
  - Optional: NSPredicate
- Generates
  - Rows, sections and index

NSFetchedResultsController Cla

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreData.f
Availability	Available in iOS 3.0 and later.
Companion guide	Core Data Programming Guide
Declared in	NSFetchedResultsController.h
Related sample code	CoreDataBooks DateSectionTitles iPhoneCoreDataRecipes TopSongs

Overview

You use a fetched results controller to efficiently manage the resul
`UITableView` object.

While table views can be used in several ways, this object is primar
expects its data source to provide cells as an array of sections mad
fetch request that specifies the entity, an array containing at least
`NSFetchedResultsController` efficiently analyzes the result of t
in the result set, and for the index.

In addition, `NSFetchedResultsController` provides the followin

- It optionally monitors changes to objects in its associated mana
  delegate (see “The Controller’s Delegate”).
- It optionally caches the results of its computation so that if the
  to be repeated (see “The Cache”).

A controller thus effectively has three modes of operation, determi
is set.

CORE DATA AND TABLES

- Create an instance of NSFetchedResultsController
  - Entity
  - NSArray of sort ordering
  - Optional: NSPredicate
- Generates
  - Rows, sections and index

```
jectContext? {
shared.delegate as? AppDelegate)?.persistentContainer.v


results controller in a block; we will only be using
his view controller
ontroller: NSFetchedResultsController<User> = {

etchRequest<User>(entityName: "User")
hSize = 10
SSortDescriptor(key: "timestamp", ascending: false)
iptors = [dateDescriptor]

troller = NSFetchedResultsController(fetchRequest: fetc
                                     managedObjectConte
                                     sectionNameKeyPath
                                     cacheName: nil)

ler.delegate = self
Controller
```

CORE DATA AND TABLES

```
///
var context: NSManagedObjectContext? {
  return (UIApplication.shared.delegate as? AppDelegate)?.persistentContainer.viewContext
}

/// Compute the fetched results controller in a block; we will only be using
/// one controller for this view controller
lazy var fetchedResultsController: NSFetchedResultsController<User> = {

  let fetchRequest = NSFetchRequest<User>(entityName: "User")
  fetchRequest.fetchBatchSize = 10
  let dateDescriptor = NSSortDescriptor(key: "timestamp", ascending: false)
  fetchRequest.sortDescriptors = [dateDescriptor]

  let _fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
                                                             managedObjectContext: self.context!,
                                                             sectionNameKeyPath: "name",
                                                             cacheName: nil)
  _fetchedResultsController.delegate = self
  return _fetchedResultsController

}()
```

CORE DATA AND TABLES

```
///
var context: NSManagedObjectContext? {
  return (UIApplication.shared.delegate as? AppDelegate)?.persistentContainer.viewContext
}

/// Compute the fetched results controller in a block; we will only be using
/// one controller for this view controller
lazy var fetchedResultsController: NSFetchedResultsController<User> = {

  let fetchRequest = NSFetchRequest<User>(entityName: "User")
  fetchRequest.fetchBatchSize = 10
  let dateDescriptor = NSSortDescriptor(key: "timestamp", ascending: false)
  fetchRequest.sortDescriptors = [dateDescriptor]

  let _fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
                                                             managedObjectContext: self.context!,
                                                             sectionNameKeyPath: "name",
                                                             cacheName: nil)
  _fetchedResultsController.delegate = self
  return _fetchedResultsController

}()
```

CORE DATA AND TABLES

```
// MARK: - Table view data source

override func numberOfSections(in tableView: UITableView) -> Int {
  if let sections = fetchedResultsController.sections {
    return sections.count
  }
  return 0
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
  if let sections = fetchedResultsController.sections {
    let currentSection = sections[section]
    return currentSection.numberOfObjects
  }
  return 0

}

override func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
  if let sections = fetchedResultsController.sections {
    let currentSection = sections[section]
    return String(currentSection.name.characters.prefix(1))
  }
  return nil
}
```

CORE DATA AND TABLES

Even handle sectioning a table for you

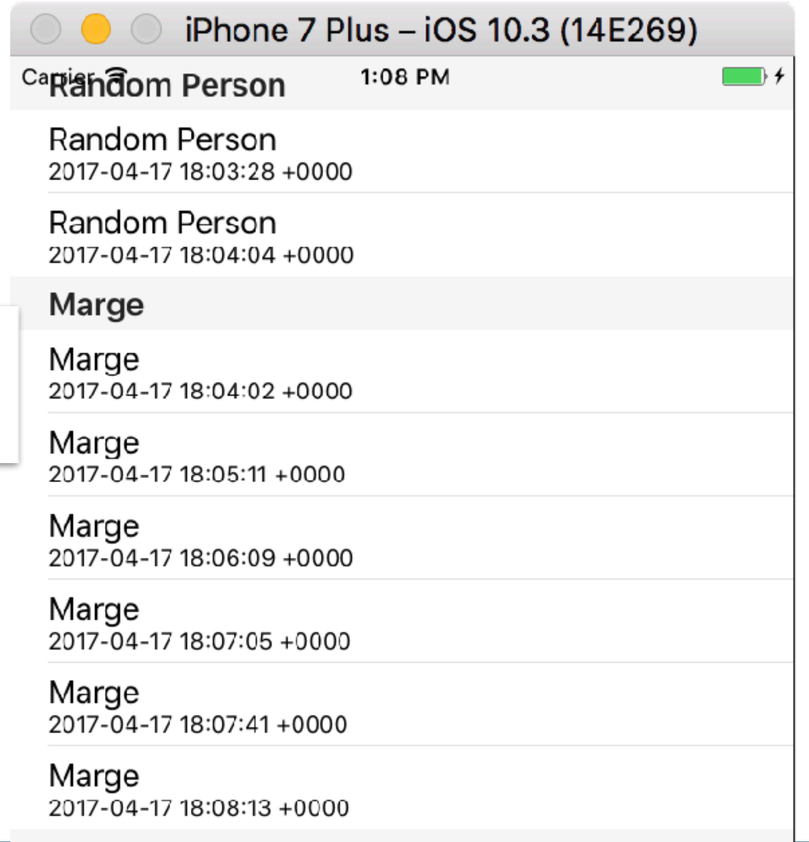
```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
  let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)

  // Configure the cell...
  let user = fetchedResultsController.object(at: indexPath)
  cell.textLabel?.text = user.name
  cell.detailTextLabel?.text = user.timestamp?.description
  return cell
}
```

CORE DATA AND TABLES

```
override func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
  if let sections = fetchedResultsController.sections {
    let currentSection = sections[section]
    return currentSection.name
  }
  return nil
}
```

```
let _fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
                                                           managedObjectContext: self.context!,
                                                           sectionNameKeyPath: "name",
                                                           cacheName: nil)
```

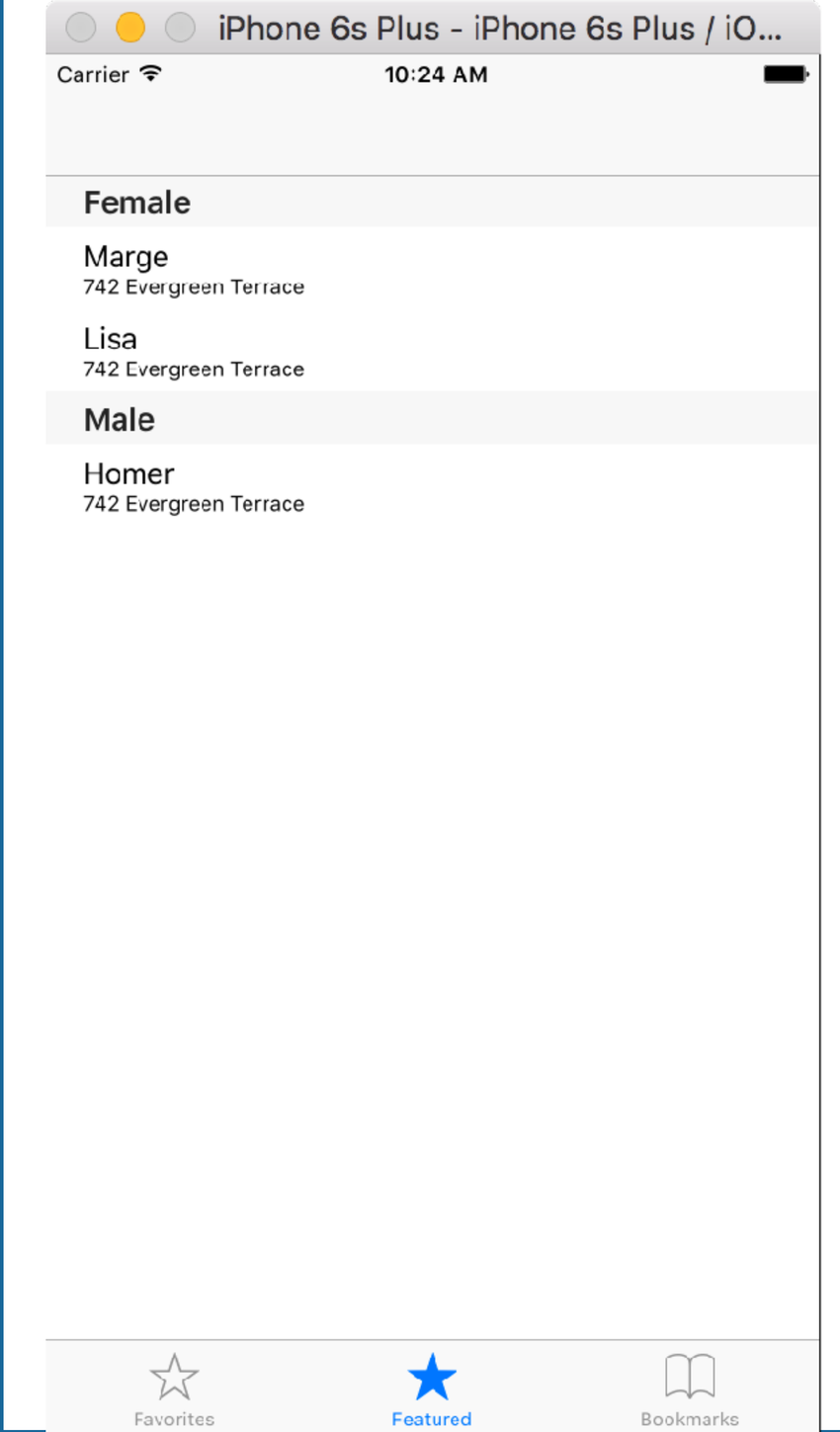
Section headers are created automatically in NSSectionInfo



CORE DATA AND TABLES

- Many more options
  - Display
  - Performance

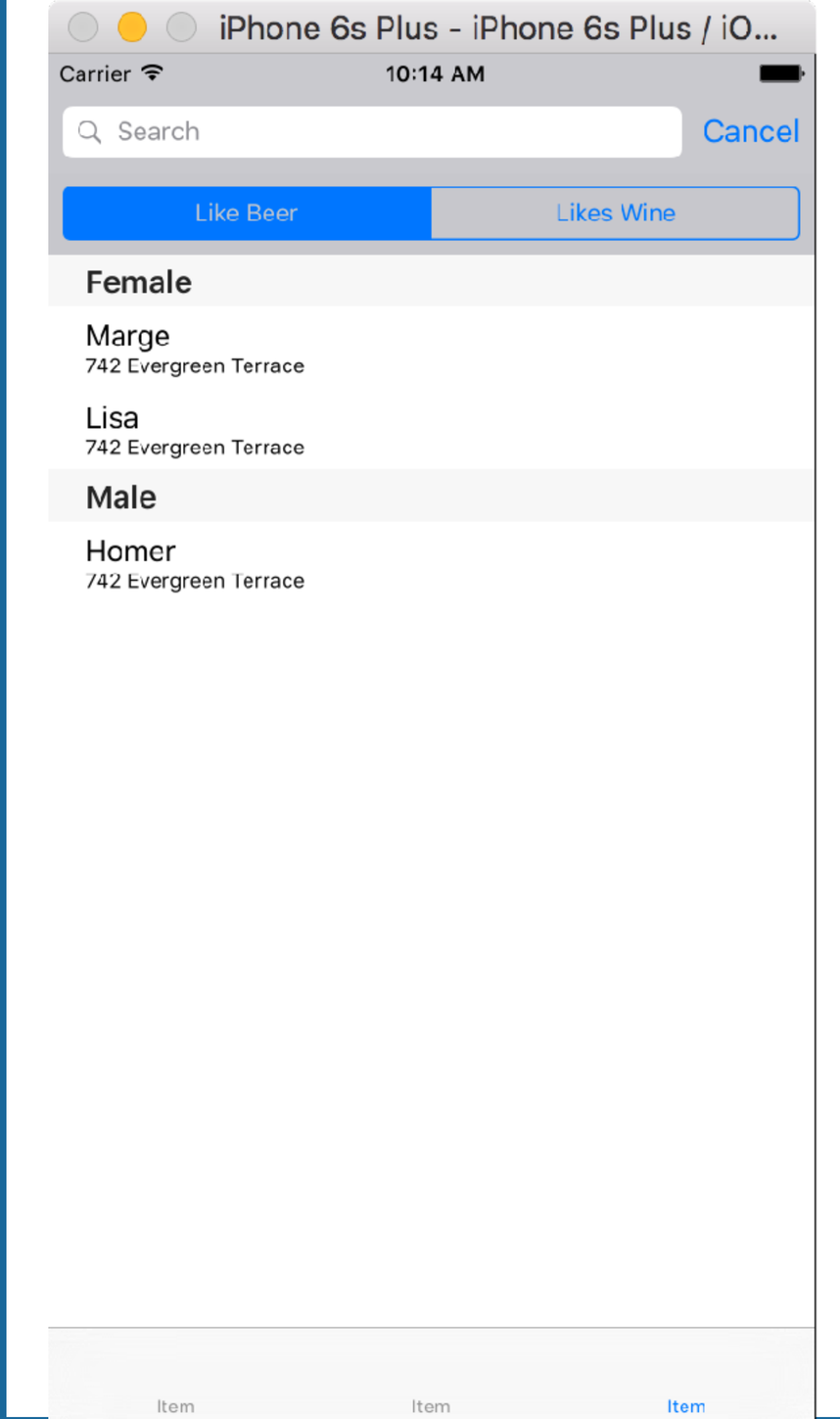
- NSFetchedResultsController delegate
  - Watch changes to the table
  - Update



SEARCH A STORE

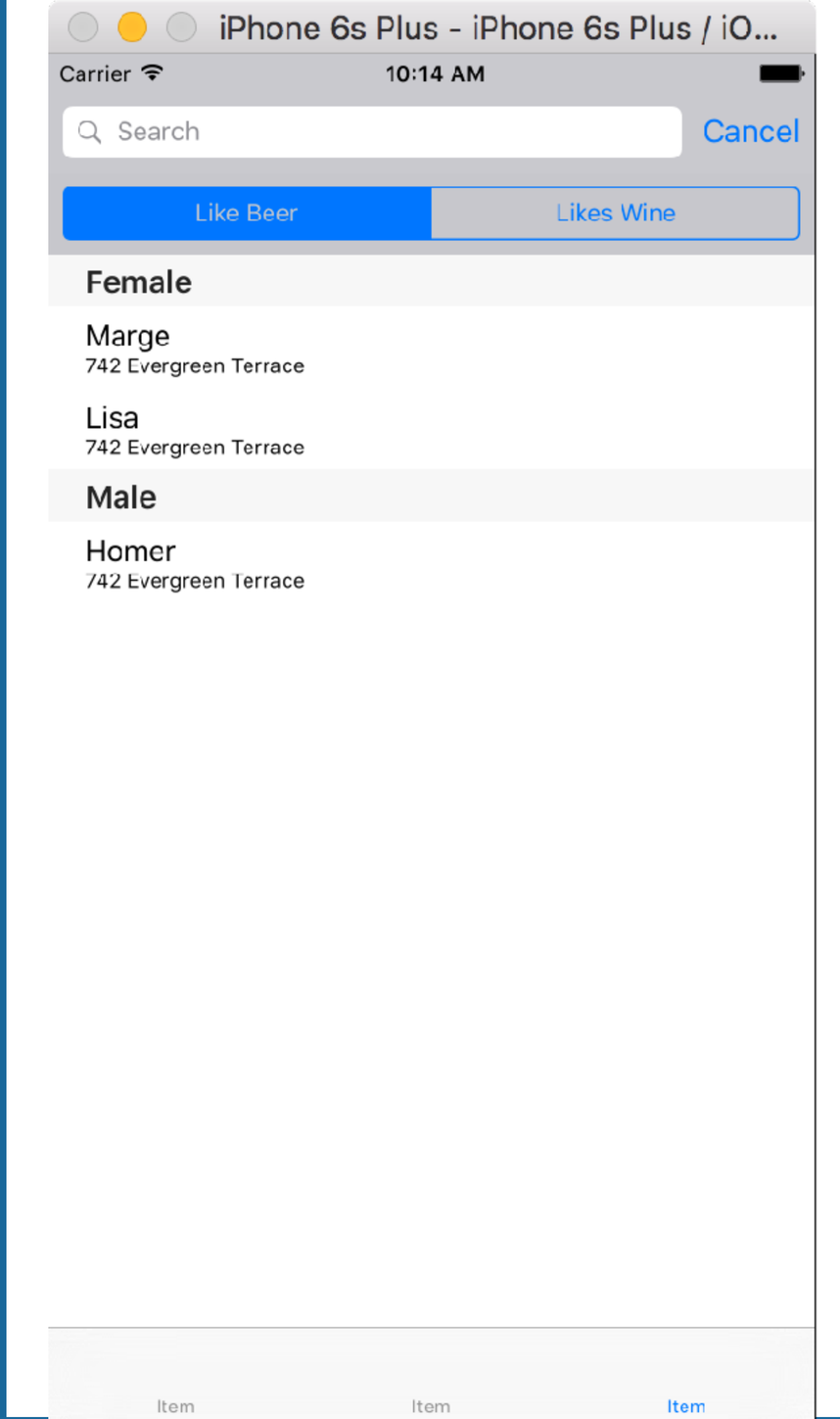
SEARCH A STORE

- There are many options for searching
- Things to consider
  - Data source (internet, local)
  - Filtering algorithm (eg. character limit)
  - Filtering options
  - Where do you want the search bar
  - Data to fetch (all vs. some)



SEARCH A STORE

- Live filtering against a Core Data store
  - UISearch bar that takes text input
  - Initially the search predicate matches everything (all results)
  - As the user types, update the search predicate
  - Canceling or clearing the field returns the predicate to everything



SEARCH A STORE

SUBTITLE

- UISearchController introduced in iOS9

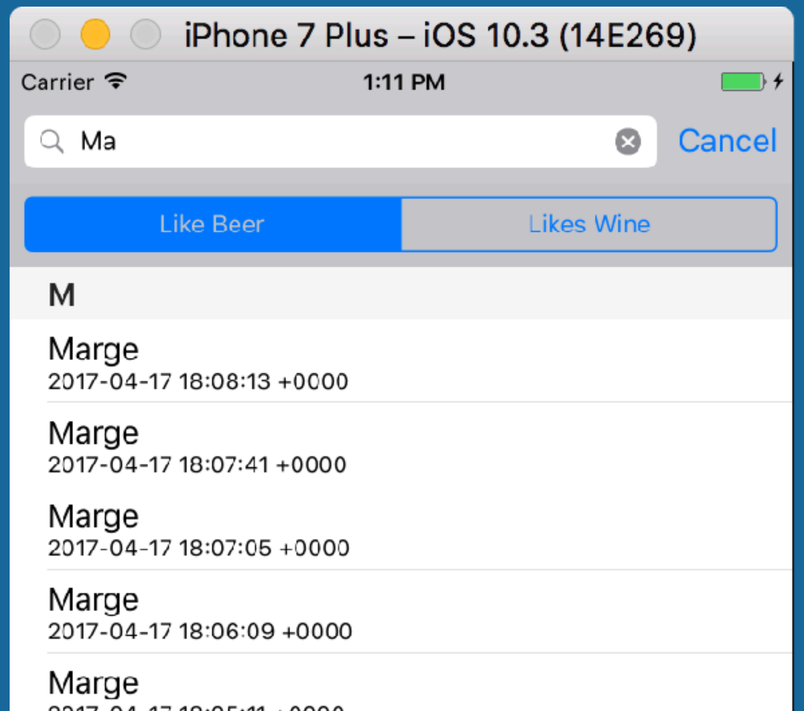


Table Search with UISearchController

Download Sample Code

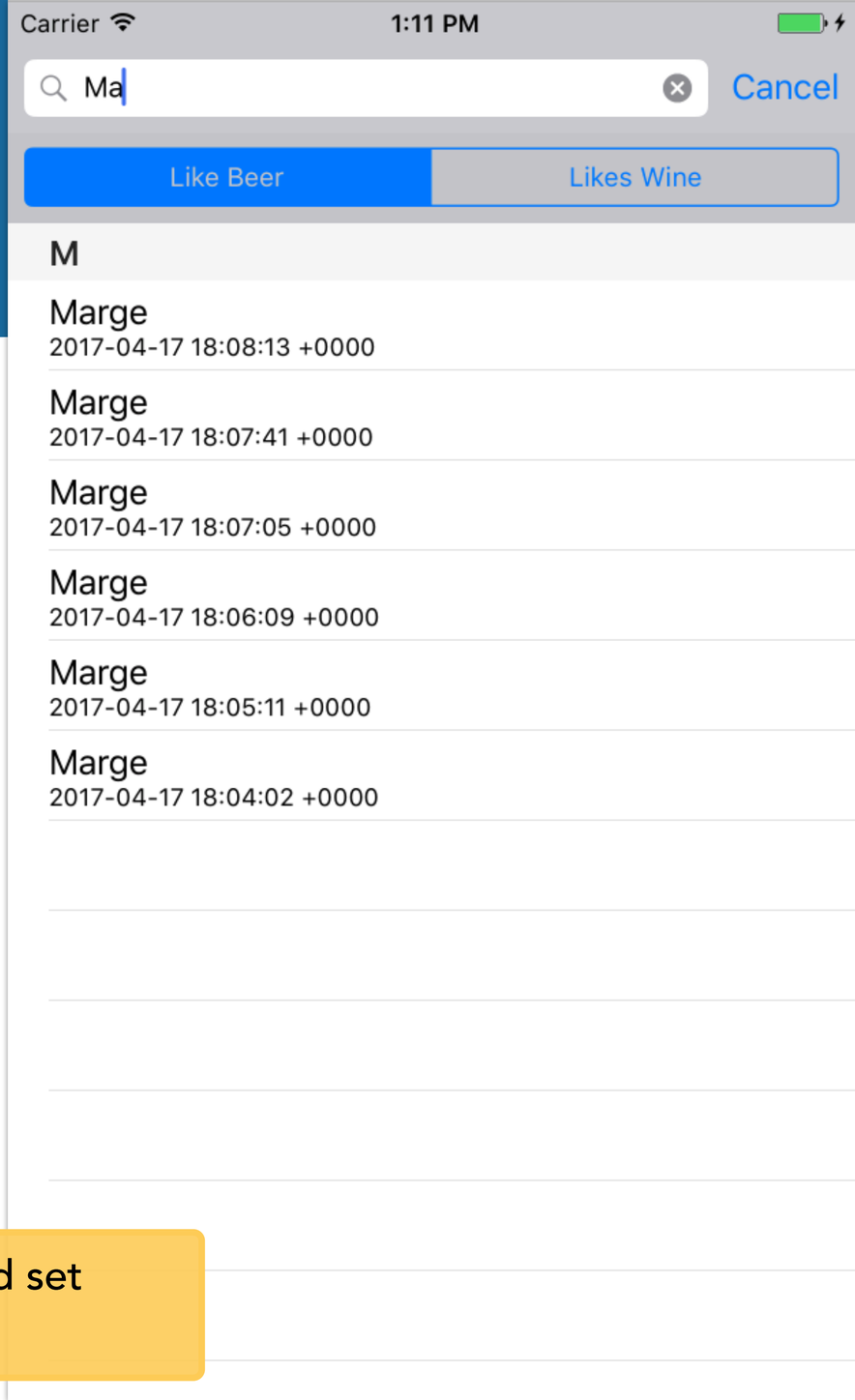
Table Search with UISearchController

Last Revision:	Version 1.3, 2015-09-16 Including Swift project. (Full Revision History)
Build Requirements:	iOS 9 SDK or later
Runtime Requirements:	iOS 8 or later

"Table Search with UISearchController" is an iOS sample application that demonstrates how to use UISearchControll
presentation of a search bar (in concert with the results view controller's content).

SEARCH A STORE

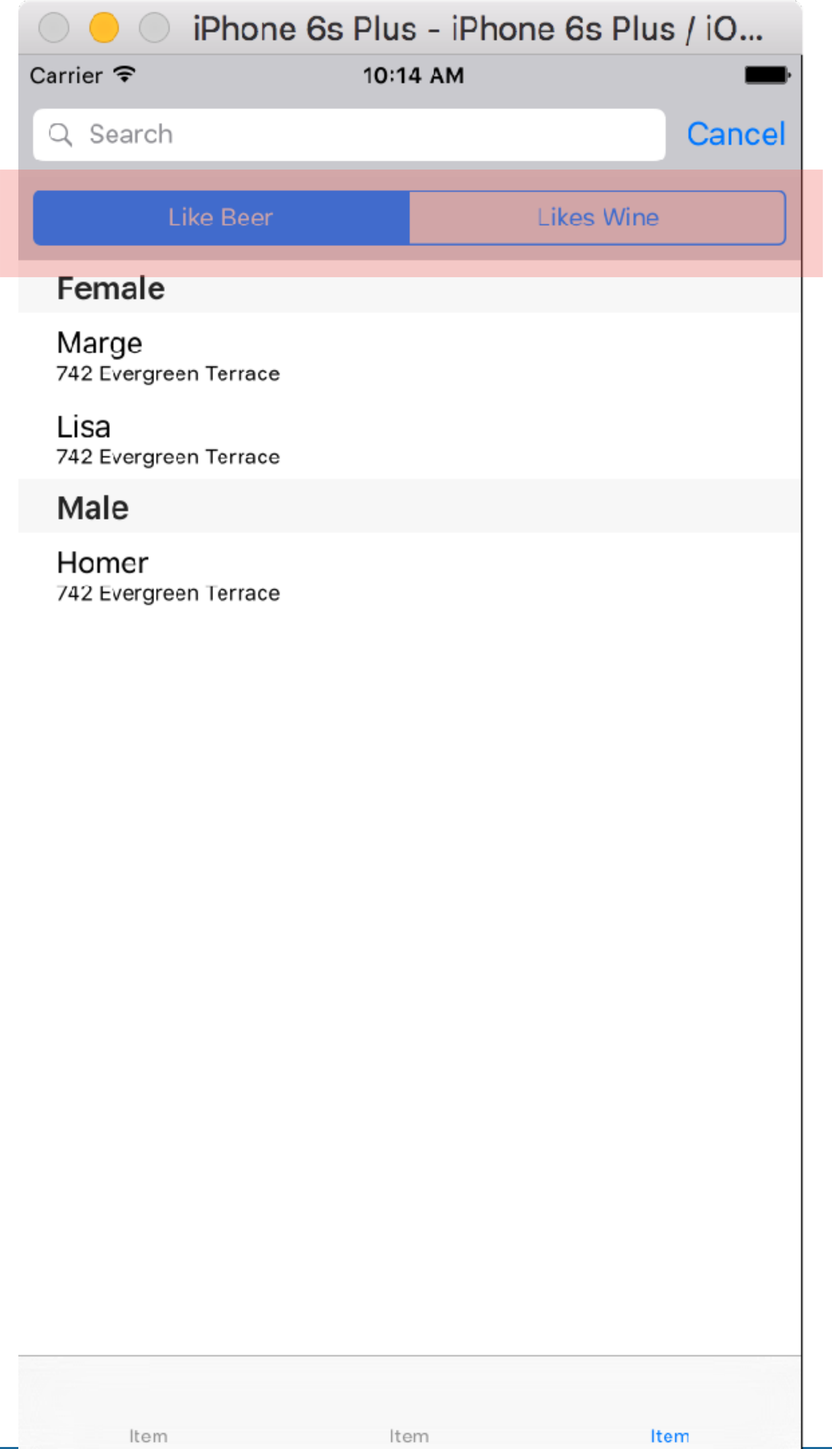
```
/// Search bar
let searchController = UISearchController(searchResultsController: nil)


// MARK: - Lifecycle
override func viewDidLoad() {

  super.viewDidLoad()
  // Load up the search controller.  There are many options on how it displays
  // that do not effect the behavior.
  searchController.searchResultsUpdater = self
  searchController.dimsBackgroundDuringPresentation = false
  searchController.searchBar.searchBarStyle = .prominent
  //searchController.hidesNavigationBarDuringPresentation = false

  // Show a scope bar for additional filtering parameters
  searchController.searchBar.scopeButtonTitles = ["Like Beer", "Likes Wine"]
  searchController.searchBar.delegate = self

  // Place the search bar in the header
  tableView.tableHeaderView = searchController.searchBar
  //navigationItem.titleView = searchController.searchBar
  definesPresentationContext = true

  fetch()

}
```



Add a UISearchBar and set preferences

SEARCH A STORE

```
// Load up the search controller.  There are many options on how it displays
// that do not effect the behavior.
searchController.searchResultsUpdater = self
searchController.dimsBackgroundDuringPresentation = false
searchController.searchBar.searchBarStyle = .Prominent
//searchController.hidesNavigationBarDuringPresentation = false

// Show a scope bar for additional filtering parameters
searchController.searchBar.scopeButtonTitles = ["Like Beer", "Likes Wine"]
searchController.searchBar.delegate = self

// Place the search bar in the header
tableView.tableHeaderView = searchController.searchBar
//navigationItem.titleView = searchController.searchBar
definesPresentationContext = true

// Offset the table to hide the search bar
//tableView.contentOffset = CGPoint(x:0, y:44)
```



SEARCH A STORE

```
/// Compute the fetched results controller in a block; we will only be using
/// one controller for this view controller
lazy var fetchedResultsController: NSFetchedResultsController<User> = {
  let fetchRequest = NSFetchRequest<User>(entityName: "User")
  fetchRequest.fetchBatchSize = 10
  fetchRequest.predicate = self.currentPredicate
  let dateDescriptor = NSSortDescriptor(key: "timestamp", ascending: false)
  fetchRequest.sortDescriptors = [dateDescriptor]

  let _fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
                                                             managedObjectContext: self.context!,
                                                             sectionNameKeyPath: "name",
                                                             cacheName: nil)
  _fetchedResultsController.delegate = self
  return _fetchedResultsController

}()

/// Predicate variables; set default value to everything
var currentPredicate: NSPredicate = NSPredicate(format: "TRUEPREDICATE")

/// Search bar
let searchController = UISearchController(searchResultsController: nil)
```

SEARCH A STORE

```
///
/// UISearchResultsDelegate
///


extension SearchTableViewController: UISearchResultsUpdating {

  /// Update the search results based on the text in the search bar.  We could
  /// also use the scope bar, but we are not in this example
  /// - paremeter searchController: The search controller sending the message
  ///
  func updateSearchResults(for searchController: UISearchController) {
    // Use this is you want to use the scope to filter on as well
    //let searchBar = searchController.searchBar
    //let scope = searchBar.scopeButtonTitles![searchBar.selectedScopeButtonIndex]
    let scope = "Dummy Text to hide scope for now"

    filterResultsForSearchText(searchController.searchBar.text!,scope: scope)
  }
}

///
/// UISearchBarDelegate
///
extension SearchTableViewController: UISearchBarDelegate {

  func searchBar(_ searchBar: UISearchBar, selectedScopeButtonIndexDidChange selectedScope: Int) {
    filterResultsForSearchText(searchBar.text!, scope: searchBar.scopeButtonTitles![selectedScope])
  }
}
```

Get the text

SEARCH A STORE

```
/// Perform a fetch against the Core Data store.  This will be called many
/// different times: initialization, search, cleared search.
func fetch() {
  print("🐶 Fetch with predicate: \(self.fetchedResultsController.fetchRequest.predicate!)")
  do {
    fetchedResultsController.fetchRequest.predicate = currentPredicate
    try self.fetchedResultsController.performFetch()
  } catch {
    let nserror = error as NSError
    print("Unresolved error \(nserror), \(nserror.userInfo)")
  }
}

/// Filter the data store by the text query and scope in the search bar
/// - Note: Scope is not actually used here.
func filterResultsForSearchText(_ text: String, scope: String = "All") {
  print("\t🎹SearchBar: \(text) Scope: \(scope)")

  // Match everything unless there is text in the search bar
  if text.characters.count == 0 {
    self.currentPredicate = NSPredicate(format: "TRUEPREDICATE")
  } else {
    self.currentPredicate = NSPredicate(format: "name CONTAINS[cd] %@", text)
  }
  fetch()
  tableView.reloadData()
}
```

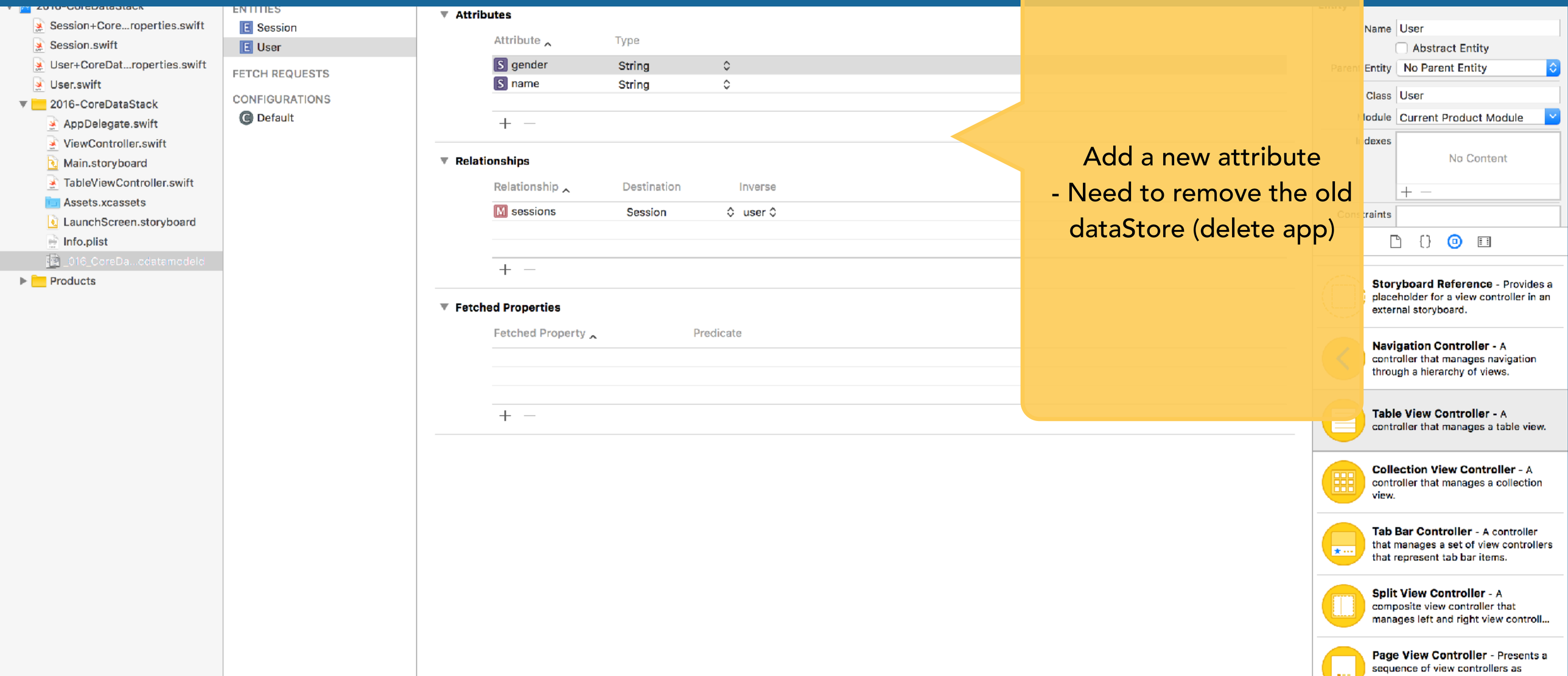
Fetch with updated predicate

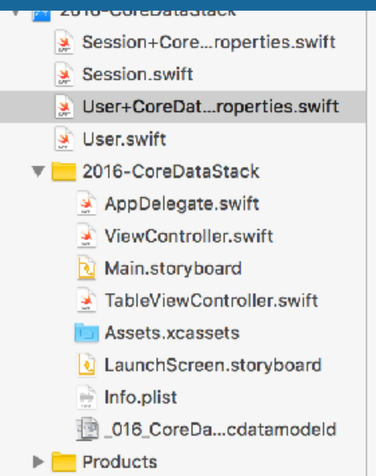
Update the predicate

SEARCH A STORE

```
/// Compute the fetched results controller in a block; we will only be using
/// one controller for this view controller
lazy var fetchedResultsController: NSFetchedResultsController<User> = {
  let fetchRequest = NSFetchRequest<User>(entityName: "User")
  fetchRequest.fetchBatchSize = 10
  fetchRequest.predicate = self.currentPredicate
  let dateDescriptor = NSSortDescriptor(key: "timestamp", ascending: false)
  fetchRequest.sortDescriptors = [dateDescriptor]

  let _fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
                                                             managedObjectContext: self.context!,
                                                             sectionNameKeyPath: "name",
                                                             cacheName: nil)
  _fetchedResultsController.delegate = self
  return _fetchedResultsController

}()
```

SOME THINGS THAT WILL MAKE YOU CRAZY


Session+Core...roperties.swift
Session.swift
User+CoreDat...roperties.swift
User.swift
2016-CoreDataStack
AppDelegate.swift
ViewController.swift
Main.storyboard
TableViewController.swift
Assets.xcassets
LaunchScreen.storyboard
Info.plist
_016_CoreDa...cdatamodeld
Products
ENTITIES
Session
User
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
gender
String
name
String
Relationships
Relationship
Destination
Inverse
sessions
Session
user
Fetched Properties
Fetched Property
Predicate
Add a new attribute
- Need to remove the old dataStore (delete app)
Name
User
Abstract Entity
Entity
No Parent Entity
Class
User
Module
Current Product Module
Indexes
No Content
Constraints
Storyboard Reference - Provides a placeholder for a view controller in an external storyboard.
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Table View Controller - A controller that manages a table view.
Collection View Controller - A controller that manages a collection view.
Tab Bar Controller - A controller that manages a set of view controllers that represent tab bar items.
Split View Controller - A composite view controller that manages left and right view controll...
Page View Controller - Presents a sequence of view controllers as




```swift
//
//  User+CoreDataProperties.swift
//  2016-CoreDataStack
//
//  Created by T. Andrew Binkowski on 4/11/16.
//  Copyright © 2016 The University of Chicago, Department of Computer Science. All rights reserved.
//
//  Choose "Create NSManagedObject Subclass…" from the Core Data editor menu
//  to delete and recreate this implementation file for your updated model.
//

import Foundation
import CoreData

extension User {

    @NSManaged var name: String?
    @NSManaged var gender: String?
    @NSManaged var sessions: NSSet?

}
```

Manually add this

Memory 21 MB
Disk Zero KB/s
Network Zero KB/s
Thread 1 Queue: c...ead (serial)
0 __pthread_kill
2 abort
3 AppDelegate.(persistent...
4 AppDelegate.persistentS...
5 AppDelegate.(managed...
6 AppDelegate.managedO...
7 ViewController.viewDidL...
36 UIApplicationMain
37 main
38 start
39 start
Thread 2 Queue: c...ger (serial)
Thread 3
Thread 4
Thread 5

```
61    lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
62        // The persistent store coordinator for the application. This i
63        // Create the coordinator and store
64        let coordinator = NSPersistentStoreCoordinator(managedObjectMod
65        let url = self.applicationDocumentsDirectory.URLByAppendingPath
66        var failureReason = "There was an error creating or loading the
```

2016-CoreDataStack 〉 Thread 1 〉 3 AppDelegate.(persistentStoreCoordinator.getter).(clos

```
Initialize the application's saved data" UserInfo={NSLocalizedDescription
NSLocalizedFailureReason=There was an error creating or loading the appli
{Error Domain=NSCocoaErrorDomain Code=134100 "(null)" UserInfo={metadata=
    NSPersistenceFrameworkVersion = 641;
    NSStoreModelVersionHashes =     {
        Session = <b8c99403 71b4692f 46c2e939 8d386149 79b301f6 b11259b7
        User = <9640ffde 03148fb5 73041b6a 4b9ba1d8 b7d53a4c 143bc592 108
    };
    NSStoreModelVersionHashesVersion = 3;
    NSStoreModelVersionIdentifiers =     (
        ""
    );
    NSStoreType = SQLite;
    NSStoreUUID = "3B4CC88F-A685-409F-A963-BD53AADCCEA9";
    "_NSAutoVacuumLevel" = 2;
}, reason=The model used to open the store is incompatible with the one u
Failed to initialize the application's saved data, NSLocalizedFailureReas
application's saved data., NSUnderlyingError: Error Domain=NSCocoaErrorDo
    NSPersistenceFrameworkVersion = 641;
    NSStoreModelVersionHashes =     {
        Session = <b8c99403 71b4692f 46c2e939 8d386149 79b301f6 b11259b7 eddce342 d50b90eb>;
        User = <9640ffde 03148fb5 73041b6a 4b9ba1d8 b7d53a4c 143bc592 108b0c83 a9cdc902>;
    };
    NSStoreModelVersionHashesVersion = 3;
    NSStoreModelVersionIdentifiers =     (
        ""
```

In a production application there is quite a bit of housekeeping to do to make changes to you data model or data.

Typically, requires making complete copies of a store and making changes in background and updating.

CORE DATA FIRESIDE CHAT

SUBTITLE

- Core Data is powerful, but frustrating
  - API is getting better but is still verbose and unforgiving
  - Odd integration with InterfaceBuilder
  - Non-trivial schema migration can be difficult
  - Coordinating with backend can add complexity

- Best use is in concert with TableViewController using NSFetchedResultsController
  - Most of the core code is boilerplate from templates
  - Best table performance possible

CORE DATA FIRESIDE CHAT

SUBTITLE

- Shipping your app with preloaded database
- Option 1:
  - Include raw data
  - Create and populate a datastore on first run
- Option 2:
  - Create auxiliary program to populate a datastore
  - Include in your application bundle
  - Copy to documents directory on first run

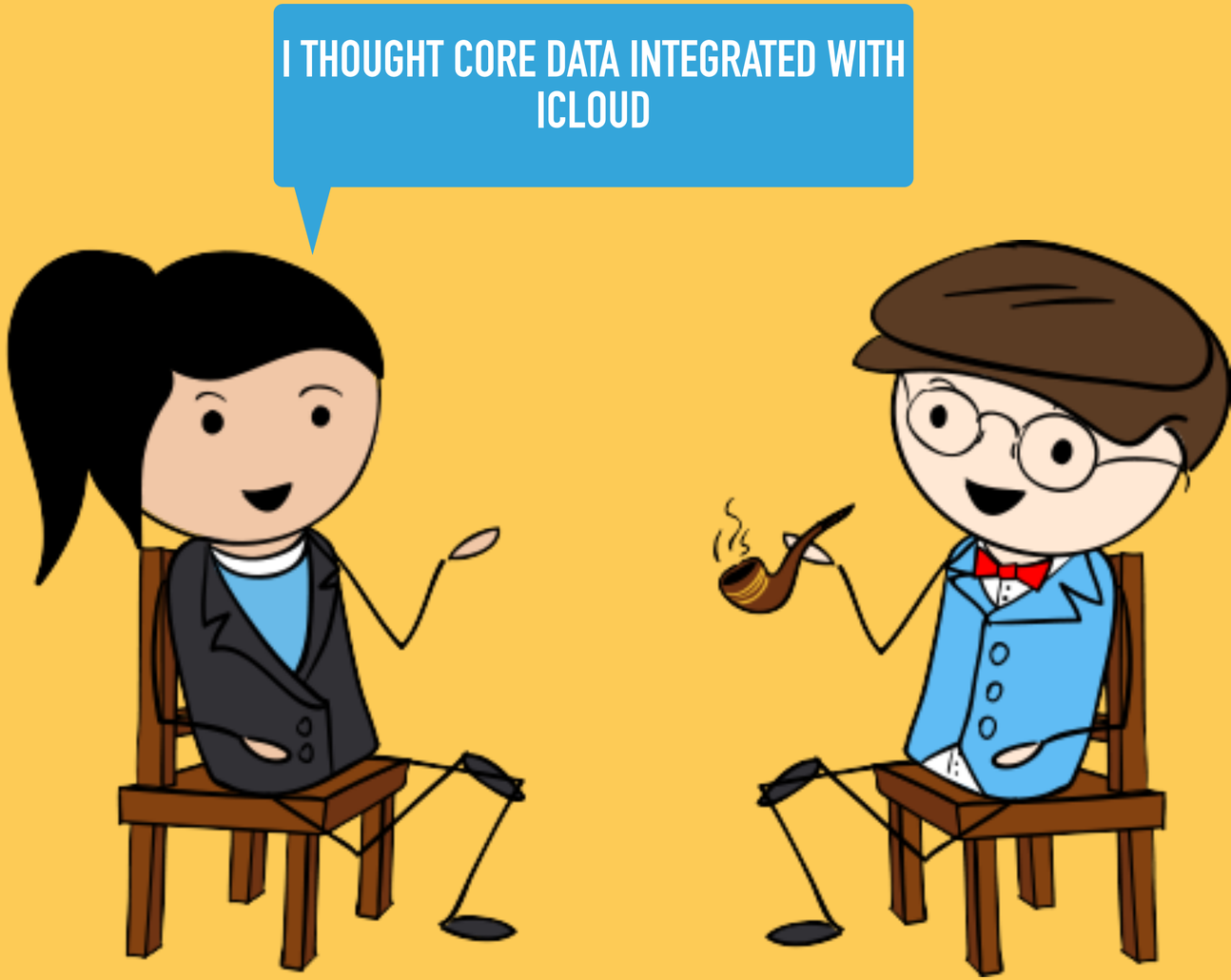
CORE DATA FIRESIDE CHAT

SUBTITLE

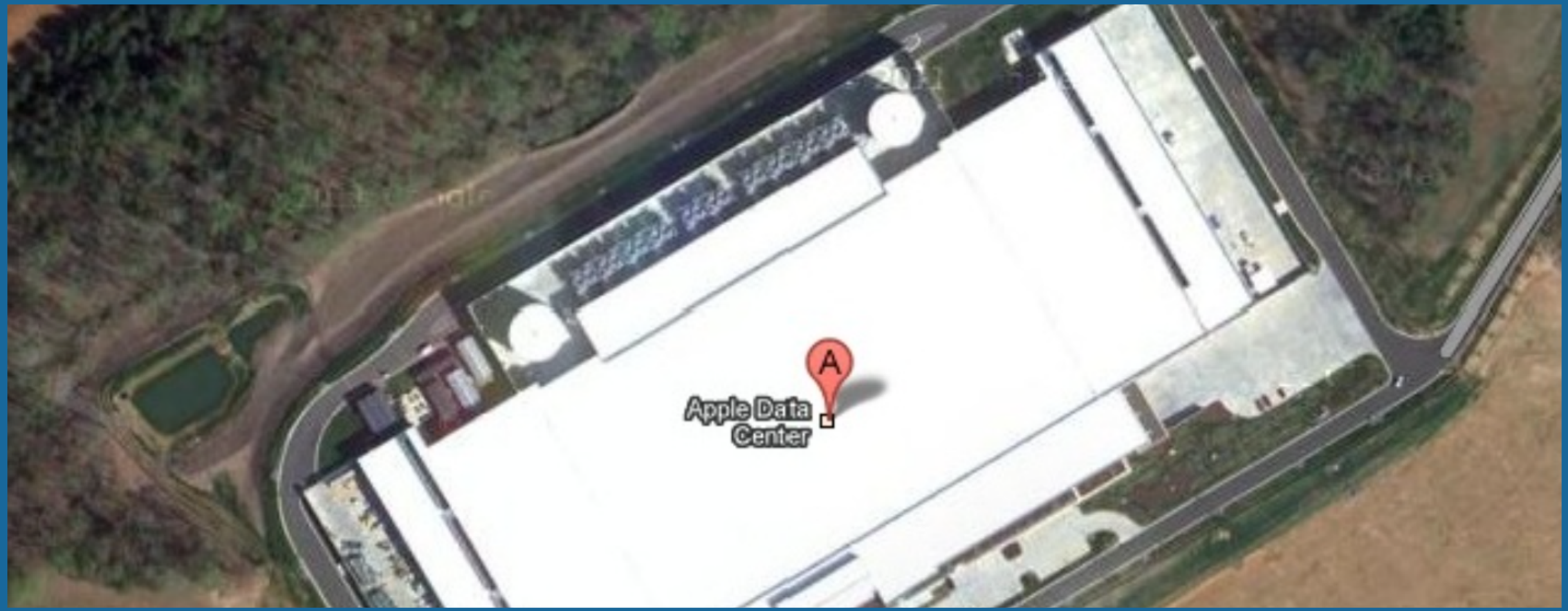
- Finding the file on your Mac
  - -(NSURL*)applicationDocumentsDirectory
  - ~/Library/ApplicationSupport/Simulator
- Loading in sqlite3
  - `> sqlite3 file.db`

- Versioning you Core Data store
  - Easy to add new entity or entity attributes
  - Difficult to change relationships
- Mapping Model to perform large scale changes

ICLOUD



ICLOUD



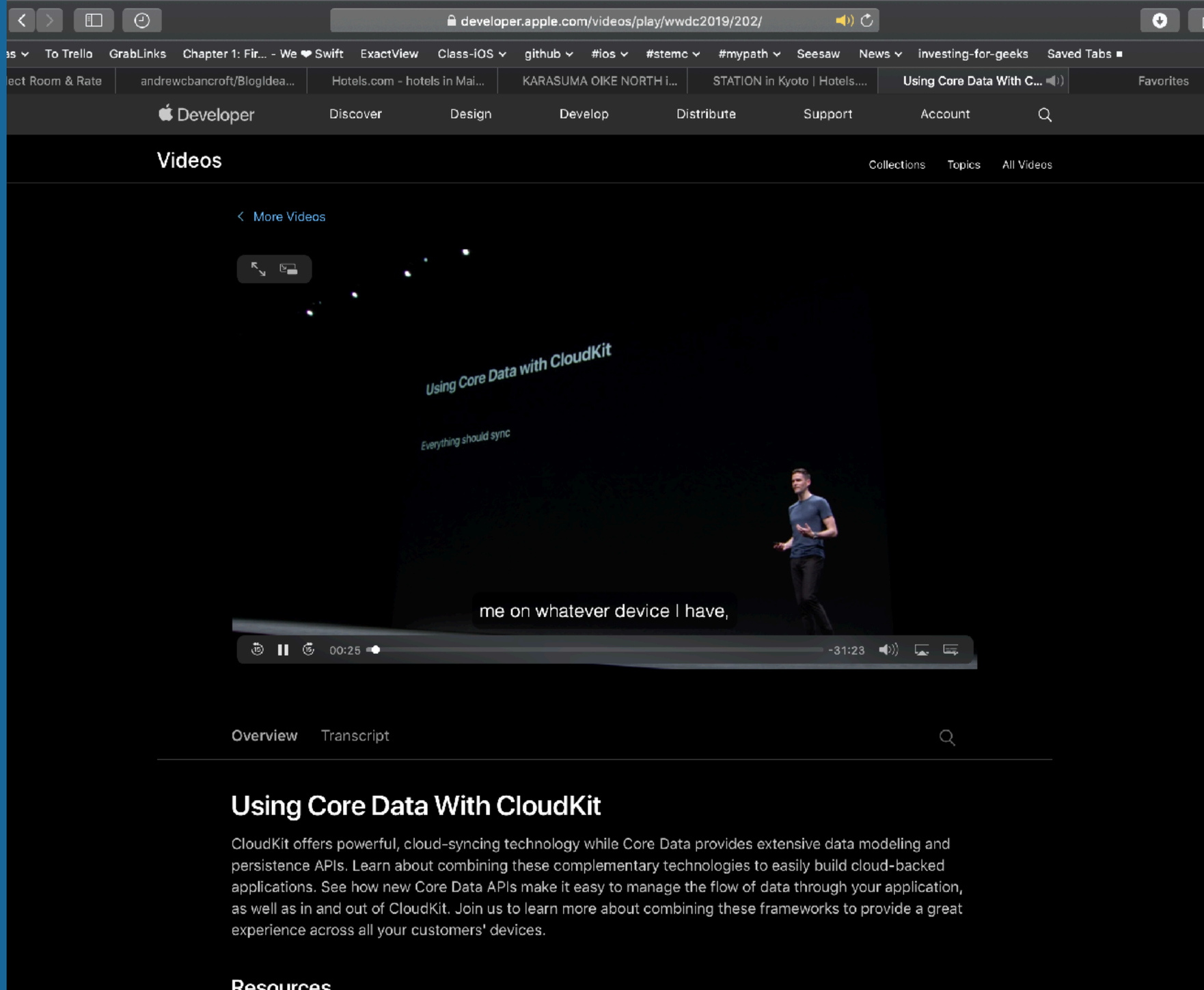
- 6028 Startown Road, Maiden, North Carolina

CORE DATA FIRESIDE CHAT

- As of iOS10 built-in coordination was deprecated
- But....

CLOUDKIT AND CORE DATA

- WWDC 2019 announcement



CLOUDKIT AND CORE DATA

- There are actually 2 ways to use CloudKit
  - Manually
  - Fully managed with Core Data Sync

etting Up Core Data with CloudKit

t up the classes and capabilities that sync your store to CloudKit.

Framework

Core Data

On This Page

Overview

See Also

verview

sync your Core Data store to CloudKit, you enable the CloudKit capability for your app. You also up the Core Data stack with a persistent container that is capable of managing one or more al persistent stores that are backed by a CloudKit private database.

onfigure a New Xcode Project

en you create a new project, you specify whether you want to add support for Core Data with udKit directly from the project setup interface. The resulting project instantiates an `PersistentCloudKitContainer` in your app's delegate. Once you enable CloudKit in your ject, you use this container to manage one or more local stores that are backed with a CloudKit abase.

1. Choose File > New > Project to create a new project.
2. Select a project template to use as the starting point for your proje
3. Select the Use Core Data and Use CloudKit checkboxes.
4. Enter any other project details and click Next.
5. Specify a location for your project and click Create.

THIS METHOD IS MORE CORE DATA THAN

THE UNIVERSITY OF CHICAGO

WATCH APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 5