**Summary: Class DataController: ObservableObject:**

**Class DataController: ObservableObject:**

Variables

container: NSPersistentCloudKitContainer

* responsible for loading/storing/syncing all data

Preview:

* Previewing data for testing

@Published var selectedFilter: Filter? = Filter.all

* Default filtering of data
* Automatically filters by all data

@Published var selectedIssue: Issue?

* Selected Issue in the list in contentView

@Published var filterText = “”

* Text used for the user to filter through results.
* It fills as things get typed into the searchable box

@Published var filterTokens = [Tag]()

* Empty tag array

@Published var filterEnabled = false

* Global on/off for the filter

@Published var filterPriority = -1

* The default priority

@Published var filterStatus = Status.al

* Default all issues

@Published var sortType = SortType.dateCreated

* When the issue was created

@Published var sortNewestFirst = true

* The most recent created issue

@Published var sortByTitle = true

* Used to sort by title

Var suggestedFilterTolens: [Tag]

* Let trimemdFilter

Private var saveTask: Task<Void, Error>?

* Does not return a value
  + Because its just calling save()
* Might throw an error before calling save

Functions

Init(inMemory: Bool = false

* Creates data to work with that wont be saved onto the disk
* Automatically merges data
* Chooses local changes over remote changes

createSampleData()

* Creates data to test the UI of the app

func save()

* Updates persistent storage if data has changed

Func queueSave()

* Cancels save
* Task runs on mainactor
* Creates a new task to wait 3 seconds to see if anything changes
* If nothing changes it call save

Private func delete(\_ fetchRequest: NSFetchRequest<NSFetchRequestResult>)

* Used with createSample Data
* Private for testing purposes
* Takes all objects and deletes them at once

func deleteAll()

* Deletes all Tags and Issues
* If adding more entities must add to delete all

Func remoteStoreChanged(\_ notification: Notification)

* Announce to the world a change has happened

Func missingTags(from issue: Issue) -> [Tag]

* Returns an array of Tags missing from a specific issue
* Requesting a Tag
* Create a Set of all tags
  + (Set removes all duplicate tags)
* Returns the the tags sorted

Func issuesForSelectedFilter() -> [Issue]

* Creates the search results for the selected filter
* Creates a array of predicates
  + Search critiera for the query
* predicate for Tag and minModificationDate
  + adds them to the predicates array
* creates an NSCompoundPredicate using AND
  + so the result must contain a Tag AND be modified recently
* creates an NSCompound using OR
  + so the results contains title OR content
* creates a search for tokens (Tags)
  + returns any issues that contains that tag
* creates a search based on priorty
* creates a serach based on completed or not
* returns the results if the search if not an empty array

func newIssue()

* creates a new issue with default title and date

Containers

Enum SortType: String

Case dateCreated = “creationDate”

* creatationDate connects to the core data value

case dateModified = “modificationDate”

* connects to core data value

case titleOrder = “title”

* connects to core data title

Enum status

Case all, open, closed

* all issues
* open/closed issues

**Class DataController: ObservableObject:**

* ObservableObject
  + Any view can create an instance and watch it if it updates
  + Exists to be watched

Static var preview: DataController = {

* Premade Data controller for viewing sample data

Let dataController = DataController(inMemory: true)

dataController.createSampleData()

return dataController

Var suggestedFilterTolens: [Tag]

* Let trimemdFilterText = String(filterText).trimmingCharacters(in: .whitespaces)
  + Removes all spaces
* Let request = Tag.fetchRequest()
  + Fetches all Tags
* If trimmedFilterText.isEmpty == false
  + request.predicate = NSPredicat(format: “name CONTAINS[c] %@”, trimmedFilterText)
    - searches for all tags
* return (try? container.viewContext.fetch(request).sorted()) ?? []
  + returns an array of all tags if not an empty array

Init(inMemory: Bool = false)

* inMemory = true data is created on disk (in RAM) wont be saved
* inMemory = false data can be saved

container= NSPersistentCloudKitContainer(name: “Main”)

* loads data Model “Main”

if inMemory

container.persistentStoreDescriptions.first?.url = URL(filePath: “/dev/null”)

* “/dev/null” means never save it write it to nowhere

Container.viewContext.automaticallyMergesChangesFromParent = true

* Updates no matter what device they are on

Container.viewContext.mergePolicy = NSMergePolicy.mergeByPropertyObjectTrump

* Update by local change over remote change
* Inmemory changes are more important than remote changes

container.loadPersistentStores { storeDescription, error in

if let error

fatalError(“Fatal error loading store: \(error.localizedDescription”)

* Failed to load the core data file and program crashes immediately

func createSampleData()

let viewContext = container.viewContext

* Data that is loaded from disk

for indexTag in 1...5

let tag = Tag(context: viewContext)

* Creates a Tag instance and which viewContext its inside

tag.id = UUID()

tag.name = “Tag \(indexTag)

for indexIssue in 1…10 {

let issue= Issue(context: viewContext)

* Creates a Issue instance and which viewContext its inside

Issue.title = “ “

Issue.exercise = “”

…

tag.addToIssues(issue)

* Core Data created method this adds tags to Issues

func save()

if container.viewContext.hasChanges

try? container.viewContext.save()

* Changes in data then call save() but only call save if there are changes

func delete(\_ object NSManagedObject)

* viewContexts own delete method and will delete a tag / issue

objectWillChange.send()

* sends out to SwiftUI that data has changed so update

container.viewContext.delete(object)

* Delete from coredata

save()

func queueSave()

saveTask?.cancel

* Cancels the save

saveTask = Task { @MainActor in

* Tells the task it must run on the main actor

try await Task.sleep(For: seconds(3))

* + Waits for a minimum 3 seconds before calling save

save()

* + Then call save if nothing has happened within the 3 seconds

Func remoteStoreChanged(\_ notification: Notification)

objectWillChange.send()

* Announce to the world a change has happened

private func delete(\_ fetchRequest: NSFetchRequest<NSFetchRequestResult>)

* Used with createSampleData but deletes all data
* Private because its used to test data
* NSFetchRequest: description used to retrieve data from persistent storage

let batchDeleteRequest = NSBatchDeleteRequest(fetchRequest: fetchRequest)

* NSBatchDeleteRequest: deletes objects from persistent storage without loading them

batchDeleteRequest.resultType = .resultTypeObjectIDs

* Returns the type of object that was deleted

if let delete = try? container.viewContext.execute(batchDeleteRequest) as? NSBatchDeleteRequest

* create delete object to execute as a batchDeleteRequest

let changes = [NSDeleteObjectsKey: delete.result as? [NSManagedObjectID] ?? []]

* the return the ID of the deleted object as a dictionary

NSManagedObjectContext.mergeChanges(fromRemoteContextSave: changes,

into: [container.viewContext])

* Merge the deleted objects with the main memory and update

func deleteAll()

let request1: NSFetchRequest<NSFetchRequest> Tag.fetchRequest()

* Delete all Tags

delete(result1)

let request2: NSFetchRequest<NSFetchRequest> Issue.fetchRequest()

* Delete all Issues

delete(result2)

save()

Func missingTags(from issue: Issue) -> [Tag]

Let request = Tag.fetchRequest()

* Requesting all tags

Let allTags = (try? container.viewContext.fetch(request)) ?? []

* Request all tags if there are none then an empty arry

Let allTAgSet = Set(allTags)

* Converting allTags into a set

Let difference = allTagsSet.symmetricDifference(issue.issueTags)

* Returns the items not in issueTags
  + (issueTags == selected Tags)

Return difference.sorted()

Func issuesForSelectedFilter() -> [Issue]

Var predicates =[NSPredicate]()

* predicates will be added to help the users filtering
* Filters the array based on specific filter or all

Let filter = selectedFilter ?? .all

* Selected filter if not default to all

if let tag = filter.tag

let tagPredicate = NSPredicate(format: “tags CONTAINS %@”, tag)

* + searches for the tags relationship to the particular issue

predicates.append(tagPredicate)

* + adding this search predicate to the predicates array

Else

Let datePredicate = NSPredicate(format: “modifcationDate > %@”, filter.minModificationDate as NSDate)

* + Issues based on their minModificationDate

predicates.append(datePredicate)

* + adding the search to the predicate array

let trimmedFilterText = filterText.trimmingCharacters(in: .whitespaces)

* + this takes the users search and removes the spaces

if trimmedFilterText.isEmpty == false

letTitlePredicate = NSPredicate(format: “title CONTAINS[c] %@”, trimmedFilterText)

* + Searches for issues containing title
    - [c] = case insensitive

letContentPredicate = NSPredicate(format: “content CONTAINS[c] %@”, trimmedFilterText)

* + - Searches for issues based on content

Let combinedPredicate =NSCompoundPredicate(orPredicateWithSubPredciates:[titlePredicate, contentPredicate], predicates.append(combinedPredicate)

* + compoundPredicate returns issues that contain title OR content

If filterTokens.isEmpty == false

* Let tokenPredicate = NSPredicate(format: “Any tags IN %@”, filterTokens)
  + ANY Token(Tags) search predicate looking for tags

Predicates.append(tokenPredicate)

* + Another search criteria to add to the predicates array

If filterEnabled

If filterPriority >= 0

Let priorityFilter = NSPredicate(format: “priority = %d”, filterPriority)

* + - Search for prediates based on priority

Predicates.append(priorityFilter)

* + - Add the priority to the criteria serach

If filterStatus != .all

Let lookForClosed = filterStatus == .closed

* + - * Only set to true if looking for closed

Let statusFilter = NSPredicate(format: “completed = %@”, NSNumber(value: lookForClosed))

* + - * Converts the bool to 1 or 0 depending on T/F

Predicates.append(statusFilter)

let request = Issue.FetchRequest

* Requesting all issues

Request.Predicate = NSCompoundPredicate(andPredicateWithSubpredicates: predicates)

* + Combines the search criteria into one predicate
    - Both Tag AND Modification date

Let request.sortDescriptors = [NSSortDescriptor(key: sortType.rawValue, ascending: sortNewestFirst)]

* + Sorts by the rawValue and defaults to newestfirst

Let allIssues = (try? container.viewContext.fetch(request)) ?? []

* + Try and fetch all issues that match the predicates
  + If not return an empty array

return allIssues

all tokens to replace any above

* If filterTokens.isEmpty == false
  + For filterToken in filterTokens
    - Let tokenPredicate = NSPredicate(format: “tags CONTAINS %@”, filterTokens)
      * ALL Token(Tags) search predicate looking for tags
  + Predicates.append(tokenPredicate)
    - Another search criteria to add to the predicates array

Func newIssue()

Let issue = Issue(context: container.viewContext)

* Creates a new issue in core data

Issue.title = “New Issue”

* Default name

Issue.creationDate = .now

* Default date

Issue.priority = 1

If let tag = selectedFilter?.tag

* Current tag being viewed

Issue.addToTags(tag)

selectedIssue = issue

Save()

Func newTag()

Let tag = Tag(context: container.viewContext)

* Creates a tag

Tag.id = UUID()

Tag.name = “New Tag”

Save()

Func completeRename()

tagToRename?.name = tagName

* Setting the tag name

dateController.save()

* Calling save to save the new name

Func count<T>(for fetchRequest: NSFetchRequest<T>) -> Int

(try? container.viewContext.count(for: fetchRequest)) ?? 0

* Returns the number of items for each fetch request

Func hasEarned(award: Award)-> Bool

Switch award.criterion

* Award.criterion comes from the award json file

Case “issues”

Let fetchrequiest = Issue.fetchRequest

Let awardCount = count(For: fetchRequest)

Return awardCount >= award.value

* Fetch all Issues and count them
* Check if awardCount >= award.value on the awards JSON

Case “closed”:

Let fetchRequest = Issue.fetchRequest()

Fetchrequest.predicate = NSPredicate(fomat: “completed = true”)

Let awardCount = count(for: fetchRequet)

Return awardCount>= award.value

* Fetch all completed Issues
* Check if awardCount is >= award.value on the JSON

Case “tags”:

Let fetchRequest = Tag.fetchRequest()

Let awardCoun t = count(for: fetchRequest)

Return awardCount >= award.value

* Fetch all Tags
* Check if awardCount is >= award.value on the JSON

Default:

Return false