```swift
//
//  Persistence.swift
//  ElegantTaskApp
//
//  Created by 이윤지 on 2023/04/14.
//
import CoreData
import SwiftUI

class PersistenceController {
    static let shared = PersistenceController()

    let container: NSPersistentContainer

    init() {
        container = NSPersistentContainer(name: "ElegantTaskApp")
        container.loadPersistentStores(completionHandler: {
         (storeDescription, error) in
            if let error = error as NSError? {
                fatalError("Unresolved error \(error), \(error.userInfo)")
            }
        })
        container.viewContext.automaticallyMergesChangesFromParent = true
    }



    func saveNewTasks(_ updatedTasks: [TaskMetaData]) {
        let viewContext = container.viewContext

        for taskMetaData in updatedTasks {
            // 업데이트된 데이터만 활용하여 새로운 TaskMetaData 객체 생성
            let newTaskMetaData = TaskMetaData(id: taskMetaData.id, task:
             [], taskDate: taskMetaData.taskDate, isOn3: taskMetaData.isOn3)

            guard let title = taskMetaData.task.first?.title else {
                // 제목이 없는 경우, 작업 추가를 건너뜀
                continue
            }

            if isTaskAlreadyExists(for: taskMetaData) {
                // 이미 존재하는 작업인 경우, 작업 추가를 건너뜀
                continue
            }

            let newItem = Item(context: viewContext)
            newItem.id2 = UUID()
            newItem.title2 = title
            newItem.time2 =
             taskMetaData.task.first?.time.addingTimeInterval(-24*60*60)
            newItem.isOn32 = taskMetaData.isOn3
            newItem.isCircleShown = isCircleShown(for: taskMetaData)

            // 새로운 작업 아이템을 저장
            do {
```

```
            try viewContext.save()
            print("⭐새로운 작업이 Core Data에 저장됨: \(newTaskMetaData)")
        } catch {
            print("작업 저장에 실패했습니다: \(error)")
        }
    }
}
```

```
private func isTaskAlreadyExists(for taskMetaData: TaskMetaData) ->
 Bool {
    let viewContext = container.viewContext
    let fetchRequest: NSFetchRequest<Item> = Item.fetchRequest()
    let taskDate = taskMetaData.taskDate
    let taskTitle = taskMetaData.task.first?.title ?? ""

    // Check if there is an existing task with the same date and title
    fetchRequest.predicate = NSPredicate(format: "time2 == %@ && title2
     == %@", taskDate as NSDate, taskTitle)

    do {
        let count = try viewContext.count(for: fetchRequest)
        return count > 0
    } catch {
        print("일정 조회에 실패했습니다: \(error)")
        return false
    }
}
```

```
private func isCircleShown(for taskMetaData: TaskMetaData) -> Bool {
    let viewContext = container.viewContext
    let fetchRequest: NSFetchRequest<Item> = Item.fetchRequest()
    let taskDate = taskMetaData.taskDate

    // 해당 날짜의 일정이 있는지 확인합니다.
    fetchRequest.predicate = NSPredicate(format: "time2 == %@",
     taskDate as NSDate)

    do {
        let count = try viewContext.count(for: fetchRequest)
        return count > 0
    } catch {
        print("일정 조회에 실패했습니다: \(error)")
        return false
    }
```

```
}

func fetchTasks() -> [TaskMetaData] {
    let viewContext = container.viewContext
    let fetchRequest: NSFetchRequest<Item> = Item.fetchRequest()

    do {
        let items = try viewContext.fetch(fetchRequest)
        var taskMetaDataArray: [TaskMetaData] = []

        for item in items {
            if let title = item.title2, !title.isEmpty {
                let task = Task(id: item.id2?.uuidString ?? "",
                                title: title,
                                time:
                                 item.time2?
                                 .addingTimeInterval(24*60*60) ??
                                 Date(),
                                isOn3: item.isOn32)
                let taskMetaData = TaskMetaData(id:
                 item.id2?.uuidString ?? "",
                                                task: [task],
                                                taskDate: item.time2 ??
                                               Date(),
                                                isOn3: item.isOn32)
                taskMetaDataArray.append(taskMetaData)
            }
        }

        return taskMetaDataArray
    } catch {
        print("Failed to fetch tasks: \(error)")
        return []
    }
}




func deleteAllTasks() {
    let viewContext = container.viewContext
    let fetchRequest: NSFetchRequest<NSFetchRequestResult> =
     NSFetchRequest(entityName: "Item")
    let deleteRequest = NSBatchDeleteRequest(fetchRequest: fetchRequest)

    do {
        try viewContext.execute(deleteRequest)
        try viewContext.save()

    } catch {
        print("Failed to delete all tasks: \(error)")
    }
}
```

```
    func delete(_ taskMetaData: TaskMetaData) {
        let viewContext = container.viewContext
        let taskDate = taskMetaData.taskDate

        // 해당 날짜의 일정을 삭제합니다.
        let fetchRequest: NSFetchRequest<Item> = Item.fetchRequest()
        fetchRequest.predicate = NSPredicate(format: "time2 == %@",
         taskDate as NSDate)

        do {
            let items = try viewContext.fetch(fetchRequest)
            for item in items {
                viewContext.delete(item)
            }

            try viewContext.save()

            print("일정이 삭제되었습니다.")
        } catch {
            print("일정 삭제에 실패했습니다: \(error)")
        }
    }




    private func saveUpdatedTasks(_ updatedTasks: [TaskMetaData]) {
        // 여기에서 업데이트된 데이터를 사용하거나 전달할 수 있습니다.
        print("업데이트된 데이터:", updatedTasks)

        // saveNewTasks를 호출하여 업데이트된 데이터를 저장합니다.
        saveNewTasks(updatedTasks)
    }


}







//    func delete(_ taskMetaData: TaskMetaData) {
//        let viewContext = container.viewContext
//        let fetchRequest: NSFetchRequest<Item> = Item.fetchRequest()
//        let taskDate = taskMetaData.taskDate
//
//        // 해당 날짜의 일정을 삭제합니다.
//        fetchRequest.predicate = NSPredicate(format: "time2 == %@",
 taskDate as NSDate)
//
//        do {
//            let items = try viewContext.fetch(fetchRequest)
```

```
//            for item in items {
//                viewContext.delete(item)
//            }
//
//            try viewContext.save()
//           // viewContext.reset()
//           // viewContext.refreshAllObjects()
//
//            DispatchQueue.main.async {
//                        // 뷰 업데이트를 강제로 트리거합니다.
///           viewContext.reset()
///           viewContext.refreshAllObjects()
//                viewContext.reset()
//                viewContext.refreshAllObjects()
//                    }
//
//            let updatedTasks = fetchTasks()
//                  // 여기에서 업데이트된 데이터를 사용하거나 전달할 수 있습니다.
//                  print("업데이트된 데이터:, \(updatedTasks) ")
//
//            print("일정이 삭제되었습니다랑랑.")
//
//
//        } catch {
//            print("일정 삭제에 실패했습니다: \(error)")
//        }
//    }
//}
```