UserDefaults.standard.set(30, forKey: "UserAge")

UserDefaults.standard.set("John Doe", forKey: "UserName")

UserDefaults.standard.set(true, forKey: "HasAccount")

// ...

let userAge = UserDefaults.standard.integer(forKey: "UserAge")

let userName = UserDefaults.standard.string(forKey: "UserName")

let hasOnboarded = UserDefaults.standard.bool(forKey: "HasAccount")

class UserProfile: NSObject, NSCoding {

var name: String

var email: String

var age: Int

init(name: String, email: String, age: Int) {

self.name = name

self.email = email

self.age = age

}

required init?(coder aDecoder: NSCoder) {

name = aDecoder.decodeObject(forKey: "name") as? String ?? ""

email = aDecoder.decodeObject(forKey: "email") as? String ?? ""

age = aDecoder.decodeInteger(forKey: "age")

}

func encode(with aCoder: NSCoder) {

aCoder.encode(name, forKey: "name")

aCoder.encode(email, forKey: "email")

aCoder.encode(age, forKey: "age")

}

}

if let userData = try? NSKeyedArchiver.archivedData(withRootObject: userProfile,

requiringSecureCoding: false) {

// Write userData to disk

}

if let loadedUserData = // ... load data from disk,

let userProfile = try? NSKeyedUnarchiver.unarchiveTopLevelObjectWithData(loadedUserData) as? UserProfile {

// Use userProfile

}

let fileManager = FileManager.default

let documentsDirectory = fileManager.urls(for: .documentDirectory, in: .userDomainMask).first!

let filePath = documentsDirectory.appendingPathComponent("MyFile.txt")

// Writing to a file

let content = "Hello, User!"

try? content.write(to: filePath, atomically: true, encoding: .utf8)

// Reading from a file

if let fileContent = try? String(contentsOf: filePath) {

print(fileContent)

}

let newDirectoryPath = documentsDirectory.appendingPathComponent("MyFolder")

do {

try fileManager.createDirectory(at: newDirectoryPath, withIntermediateDirectories: true)

} catch {

print(error)

}

do {

let attributes = try fileManager.attributesOfItem(atPath: filePath.path)

let fileName = filePath.lastPathComponent

if let fileSize = attributes[.size] as? NSNumber {

print("\(fileName) size: \(fileSize.intValue) bytes")

}

} catch {

print(error)

}

let backupPath = documentsDirectory.appendingPathComponent("MyFile\_backup.txt")

try? fileManager.copyItem(at: filePath, to: backupPath)

enum KeychainError: Error {

case unhandledError(status: OSStatus)

case itemNotFound

case unexpectedItemData

case unhandledException

}

let account = "UserAccountName"

let password = "UserPassword".data(using: .utf8)!

let query: [String: Any] = [

kSecClass as String: kSecClassGenericPassword,

kSecAttrAccount as String: account,

kSecValueData as String: password

]

let status = SecItemAdd(query as CFDictionary, nil)

guard readstatus == errSecSuccess else {

throw KeychainError.unhandledError(status: status) }

var readQuery: [String: Any] = [

kSecClass as String: kSecClassGenericPassword,

kSecAttrAccount as String: account,

kSecMatchLimit as String: kSecMatchLimitOne,

kSecReturnAttributes as String: true,

kSecReturnData as String: true

]

var item: CFTypeRef?

let readStatus = SecItemCopyMatching(readQuery as CFDictionary, &item)

guard readStatus == errSecSuccess else { throw KeychainError.unhandledError(status: status) }

if let existingItem = item as? [String: Any],

let passwordData = existingItem[kSecValueData as String] as? Data,

let password = String(data: passwordData, encoding: .utf8) {

// Use password

}

struct User: Codable {

var name: String

var age: Int

}

struct User: Codable {

var name: String

var age: Int

var email: String

enum CodingKeys: String, CodingKey {

case name

case age

case email = "email\_address"

}

}

struct User: Codable {

var name: String

var age: Int

var friends: [User] // Array of Users

var preferences: [String: String] // Dictionary of preferences

}

do {

let db = try Connection("\(pathToDocuments)/mydatabase.sqlite3")

print("Successfully connected to the database")

} catch {

print("Error connecting to the database: \(error)")

}

let users = Table("users")

let id = Expression<Int>("id")

let name = Expression<String>("name")

let email = Expression<String>("email")

do {

try db.run(users.create { table in

table.column(id, primaryKey: true)

table.column(name)

table.column(email, unique: true)

})

print("Table 'users' created successfully")

} catch {

print("Error creating table: \(error)")

}

let insertUser = users.insert(name <- "John Doe", email <- "john@example.com")

do {

let rowID = try db.run(insertUser)

print("Inserted user with ID: \(rowID)")

} catch {

print("Error inserting user: \(error)")

}

for user in try db.prepare(users) {

print("ID: \(user[id]), Name: \(user[name]), Email: \(user[email])")

}

let userToUpdate = users.filter(id == 1)

let updatedData = userToUpdate.update(name <- "Updated Name")

do {

if try db.run(updatedData) > 0 {

print("User updated successfully")

} else {

print("User not found")

}

} catch {

print("Error updating user: \(error)")

}

let userToDelete = users.filter(id == 1)

do {

if try db.run(userToDelete.delete()) > 0 {

print("User deleted successfully")

} else {

print("User not found")

}

} catch {

print("Error deleting user: \(error)")

}

let index = Index("index\_email\_on\_users", on: users, columns: [email])

try db.run(index.create())

let newItem = NSEntityDescription.insertNewObject(forEntityName: "Item",

into: context)

newItem.setValue("Example", forKey: "name")

let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: "Item")

let results = try context.fetch(fetchRequest)

existingItem.setValue("New Value", forKey: "name")

try context.save()

context.delete(existingItem)

try context.save()