

IOS Depolama İşlemleri

Kasım ADALAN

Elektronik ve Haberleşme Mühendisi
Senior Android and IOS Developer

İçerik

- UserDefaults
- File İşlemleri
- CoreData
- Sqlite

User Default

User Default

- Basit verilerin tutulduğu depolama birimidir.
- Dosya tabanlı bir depolamadır.
- Kayıt yapıldığı anda uygulama silinmediği sürece uygulamada kalır.
- Uygulama açılış süresini olumsuz etkileyebileceğinden dolayı çok fazla bilgi bu depolanmamalıdır.
- Veriler key - value şeklinde depolanır.
- Uygulama ayarları veya oturum bilgilerini kullanmak için idealdir.

User Default

- Kullanılabilecek veri türleri sınırlıdır.
 - Integer
 - Double
 - Float
 - Bool
 - Array
 - Dictionary (Sadece [String:String] türünde)

Veri kaydı

```
let d = UserDefaults.standard //Kullanım için kurulum

d.set("Ahmet", forKey: "ad")//String
d.set(18, forKey: "yas")//Integer
d.set(1.78, forKey: "boy")//Double
d.set(true, forKey: "medeniDurum")//Bool

let arkadasListesi = ["ali","veysel","ece"]

d.set(arkadasListesi, forKey: "liste")//Array

let sehirlerListe:[String:String] = ["16":"Bursa","34":"Istanbul","6":"Ankara"]
//Dictionary kullanılabilir fakat sadece [String:String] türünde geçerlidir.

d.set(sehirlerListe, forKey: "dict")//Dictionary
```

Veri Okuma

```
let d = UserDefaults.standard //Kullanım için kurulum

let ad = d.string(forKey: "ad") ?? "isim yok"
// ?? işaretini bir sıkıntı olursa ad değişkenine aktarılacak veriyi gösterir.
// Yani varsayılan değeri temsil eder.String için geçerlidir.
// Int : 0 Bool : false Double : 0.0 in varsayılan değerleri otomatik aktarılır.

let yas = d.integer(forKey: "yas")
let boy = d.double(forKey: "boy")
let medeniDurum = d.bool(forKey: "medeniDurum")
```

Veri Okuma (Listeler)

```
let arkadasListesi = d.array(forKey: "liste") as? [String] ?? [String]()
//as? den sonra gelen dönüştürülecek tip.
//?? den sonra gelen userdefaulttan veri alırken sıkıntı olursa
//varsayılan olarak boş bir string liste aktarılır.

print(arkadasListesi[0])
print(arkadasListesi[1])
print(arkadasListesi[2])

let sehirlerListe = d.dictionary(forKey: "dict") as? [String:String] ?? [String:String]()
print(sehirlerListe["16"]!)
print(sehirlerListe["34"]!)
print(sehirlerListe["6"]!)
```

Veri Silme

```
let d = UserDefaults.standard //Kullanım için kurulum  
  
d.removeObject(forKey: "ad")  
//Silinmek istenilen bilginin key bilgisi yazılır ve silinir.
```

Veri Güncelleme

```
let d = UserDefaults.standard //Kullanım için kurulum  
d.set("yeni ahmet", forKey: "ad")  
//Güncelleme aslında veri kayıt işleminin tekrarıdır.
```

Uygulama : Sayaç



```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var labelSayac: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()

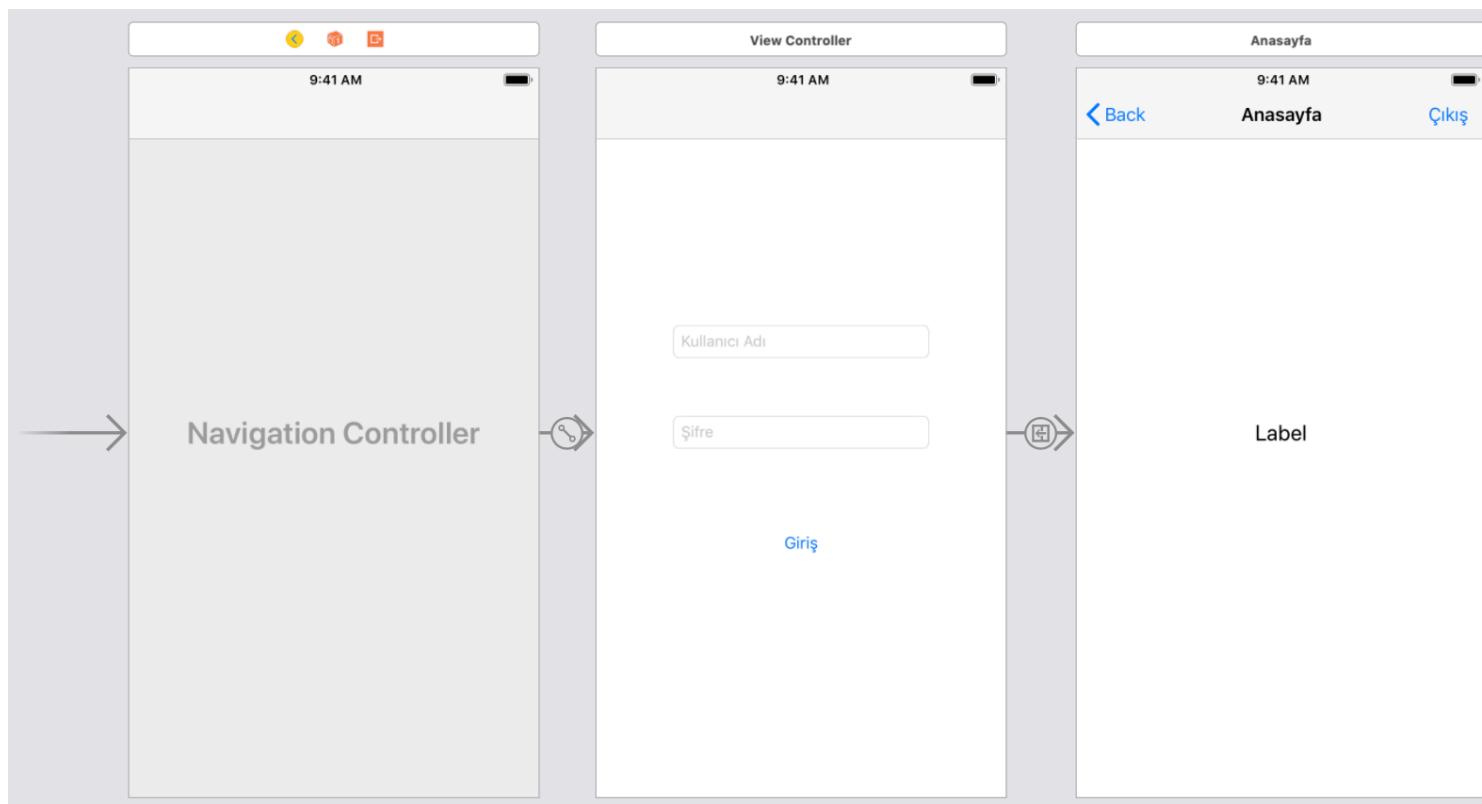
        let d = UserDefaults.standard //Kullanım için kurulum
        var sayac = d.integer(forKey: "sayac")//Uygulama açılır açılmaz veri alınır.

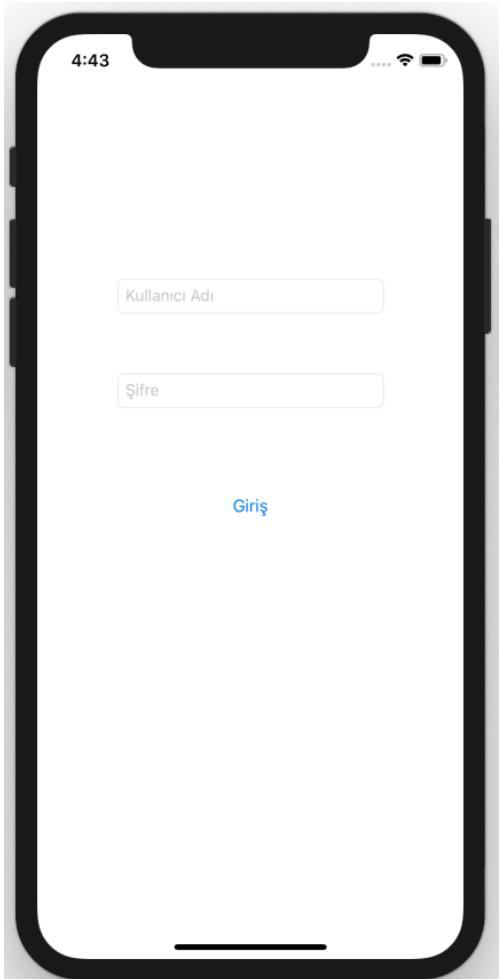
        sayac = sayac + 1//sayacı 1 artırırız

        d.set(sayac, forKey: "sayac")//Veri tekrar kayıt edilir.

        labelSayac.text = "Sayaç : \(sayac)"
    }
}
```

Uygulama : Login





```
class ViewController: UIViewController {

    @IBOutlet weak var textfieldKullaniciAdi: UITextField!
    @IBOutlet weak var textfieldSifre: UITextField!

    let d = UserDefaults.standard //Kullanım için kurulum

    override func viewDidLoad() {
        super.viewDidLoad()
        //Veriler uygulama açıldığı gibi verileri almaya çalışır.
        let ka = d.string(forKey: "kullaniciAdi") ?? "bos"
        let s = d.string(forKey: "sifre") ?? "bos"

        if ka != "bos" && s != "bos" {//Eğer her iki veride dolu ise anasayfaya geçiş yapılır.
            performSegue(withIdentifier: "giristoanasayfa", sender: nil)
            //Story board üzerindeki geçiş kodlama ile aktif edebiliriz.
            //bu kodlama prepare metodunu çalıştırır.
        }
    }

    @IBAction func girisYap(_ sender: Any) {
        //Birden fazla girdiyi bu şekilde , yardımıyla ayırip binding yapabiliriz.
        if let ka = textfieldKullaniciAdi.text , let s = textfieldSifre.text {

            if ka == "admin" && s == "123456" {

                d.set(ka, forKey: "kullaniciAdi")//Giriş doğru ise userdefault üzerine kayıt yapılır
                d.set(s, forKey: "sifre")

                //Geçiş sağlanır.
                performSegue(withIdentifier: "giristoanasayfa", sender: nil)
            }else{
                print("Hatalı Giriş")
            }
        }
    }
}

override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    //Sayfa görünür olduğunda navigation bar gizlenir.
    //Tam sayfa çalışılabilir.
    navigationController?.isNavigationBarHidden = true
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    //Bu sayfadan başka bir sayfaya geçerken eski görünür haline getirmek gereklidir.
    //Aksi halde bütün sayfalardan navigation bar gizlenir ve görünmez.
    navigationController?.isNavigationBarHidden = false
}
```



```
import UIKit

class ViewController2: UIViewController {

    @IBOutlet weak var labelSonuc: UILabel!

    let d = UserDefaults.standard //Kullanım için kurulum

    override func viewDidLoad() {
        super.viewDidLoad()

        navigationItem.hidesBackButton = true // Geri dönüsü engellemek için
                                                görünmez yapılır.

        let ka = d.string(forKey: "kulllaniciAdi") ?? "bos"

        labelSonuc.text = ka
    }

    @IBAction func cikisYap(_ sender: Any) {

        d.removeObject(forKey: "kulllaniciAdi")
        d.removeObject(forKey: "sifre")

        exit(-1) //Uygulamayı kapatır ve home screen'e döner
    }
}
```

File İşlemleri

Veri Yazma

```
let mesaj:String = textfieldGirdi.text!          Kayıt için kullanıcıya özel alan seçilir
//Textfield üzerinden kayıt edilecek veri alınır.
if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
    //doküman yolu alınır.
    let dosyaYolu = dir.appendingPathComponent("dosyam.txt")
    //Kayıt edilmek istenen dosya yolu ve dosya oluşturulur.

    //yazma işlemi          Veriler byte türüne dönüştürülsün mü ?
    do {
        try mesaj.write(to: dosyaYolu, atomically: false, encoding: String.Encoding.utf8)

        textfieldGirdi.text = ""//Yazı kayıt edildikten sonra textfield boşaltılır.
    }
    catch {/* hata işlemleri burda yapılır */}
}
```

Veri Okuma

```
if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
    //doküman yolu alınır.
    let dosyaYolu = dir.appendingPathComponent("dosyam.txt")
    //Okunmak istenen dosya adı ile dosya yolu alınır.

    //okuma işlemi
    do {
        textfieldGirdi.text = try String(contentsOf: dosyaYolu, encoding:
            String.Encoding.utf8)
        //okunan veri textfield içine aktarılır.
    }
    catch {/* hata işlemleri burda yapılır */}
}
```

Veri Silme

```
if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {  
    //doküman yolu alınır.  
    let dosyaYolu = dir.appendingPathComponent("dosyam.txt")  
    //Silinmek istenen dosya adı ile dosya yolu alınır.  
  
    if FileManager.default.fileExists(atPath: dosyaYolu.path) {  
        //Silinmek istenen dosya var mı yok mu kontrolü yapılır.  
  
        //Silme işlemi  
        do {  
            try FileManager.default.removeItem(at: dosyaYolu)  
            textfieldGirdi.text = ""//Yazı kayıt edildikten sonra textfield boşaltılır.  
        } catch {  
            print("Silme işlemi yapılamadı: \(error)")  
        }  
    }  
}
```

Resim Kayıt



```
if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
    //doküman yolu alınır.
    let dosyaYolu = dir.appendingPathComponent("resimim.png")
    //Kayıt edilmek istenen dosya yolu ve dosya oluşturulur.

    let resim = UIImage(named: "resim")
    //asset dosyasından resmi alıyoruz.

    //yazma işlemi
    do {
        try resim!.pngData()?.write(to: dosyaYolu)
    }
    catch {/* hata işlemleri burda yapılır */}
}
```

Resim okuma

```
if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {  
    //doküman yolu alınır.  
    let dosyaYolu = dir.appendingPathComponent("resimim.png")  
    //Okunmak istenen dosya adı ile dosya yolu alınır.  
  
    self.resimTutucu.image = UIImage(contentsOfFile: dosyaYolu.path)  
    //Alınan resim imageview içine aktarılır.  
}
```

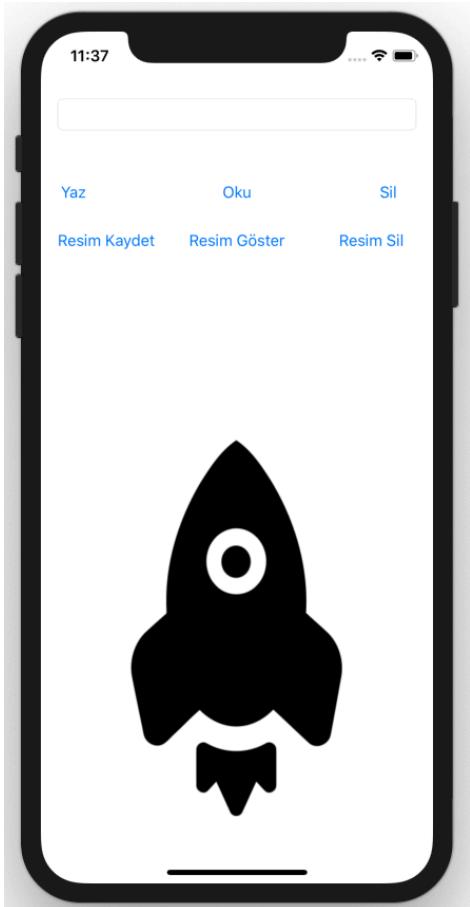
Resim Sil

```
if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
    //doküman yolu alınır.
    let dosyaYolu = dir.appendingPathComponent("resimim.png")
    //Silinmek istenen dosya adı ile dosya yolu alınır.

    if FileManager.default.fileExists(atPath: dosyaYolu.path) {
        //Silinmek istenen dosya var mı yok mu kontrolü yapılır.

        //Silme işlemi
        do {
            try FileManager.default.removeItem(at: dosyaYolu)
        } catch {
            print("Silme işlemi yapılamadı: \(error)")
        }
    }
}
```

UYGULAMA :File İşlemleri



```
class ViewController: UIViewController {

    @IBOutlet weak var textfieldGirdi: UITextField!
    @IBOutlet weak var resimTutucu: UIImageView!

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

    @IBAction func yaz(_ sender: Any) {

        let mesaj:String = textfieldGirdi.text!
        //Textfield üzerinden kayıt edilecek veri alınır.
        if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
            //doküman yolu alınır.
            let dosyaYolu = dir.appendingPathComponent("dosyam.txt")
            //Kayıt edilmek istenen dosya yolu ve dosya oluşturulur.

            //yazma işlemi
            do {
                try mesaj.write(to: dosyaYolu, atomically: false, encoding: String.Encoding.utf8)

                textfieldGirdi.text = ""//Yazı kayıt edildikten sonra textfield boşaltılır.
            }
            catch {/* hata işlemleri burda yapılır */}
        }
    }
}
```

```
@IBAction func oku(_ sender: Any) {

    if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
        //doküman yolu alınır.
        let dosyaYolu = dir.appendingPathComponent("dosyam.txt")
        //Okunmak istenen dosya adı ile dosya yolu alınır.

        //okuma işlemi
        do {
            textfieldGirdi.text = try String(contentsOf: dosyaYolu, encoding:
                String.Encoding.utf8)
            //okunan veri textfield içine aktarılır.
        }
        catch {/* hata işlemleri burda yapılır */}
    }

}

@IBAction func sil(_ sender: Any) {

    if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
        //doküman yolu alınır.
        let dosyaYolu = dir.appendingPathComponent("dosyam.txt")
        //Silinmek istenen dosya adı ile dosya yolu alınır.

        if FileManager.default.fileExists(atPath: dosyaYolu.path) {
            //Silinmek istenen dosya var mı yok mu kontrolü yapılır.

            //Silme işlemi
            do {
                try FileManager.default.removeItem(at: dosyaYolu)
                textfieldGirdi.text = ""//Yazı kayıt edildikten sonra textfield boşaltılır.
            } catch {
                print("Silme işlemi yapılamadı: \(error)")
            }
        }
    }
}
```

```
@IBAction func resimKaydet(_ sender: Any) {

    if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
        //doküman yolu alınır.
        let dosyaYolu = dir.appendingPathComponent("resimim.png")
        //Kayıt edilmek istenen dosya yolu ve dosya oluşturulur.

        let resim = UIImage(named: "resim")
        //asset dosyasından resmi alıyoruz.

        //yazma işlemi
        do {
            try resim!.pngData()?.write(to: dosyaYolu)
        }
        catch {/* hata işlemleri burda yapılır */}
    }

}

@IBAction func resimGoster(_ sender: Any) {

    if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
        //doküman yolu alınır.
        let dosyaYolu = dir.appendingPathComponent("resimim.png")
        //Okunmak istenen dosya adı ile dosya yolu alınır.

        self.resimTutucu.image = UIImage(contentsOfFile: dosyaYolu.path)
        //Alınan resim imageview içine aktarılır.
    }

}

@IBAction func resimSil(_ sender: Any) {

    if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
        //doküman yolu alınır.
        let dosyaYolu = dir.appendingPathComponent("resimim.png")
        //Silinmek istenen dosya adı ile dosya yolu alınır.

        if FileManager.default.fileExists(atPath: dosyaYolu.path) {
            //Silinmek istenen dosya var mı yok mu kontrolü yapılır.

            //Silme işlemi
            do {
                try FileManager.default.removeItem(at: dosyaYolu)
            } catch {
                print("Silme işlemi yapılamadı: \(error)")
            }
        }
    }

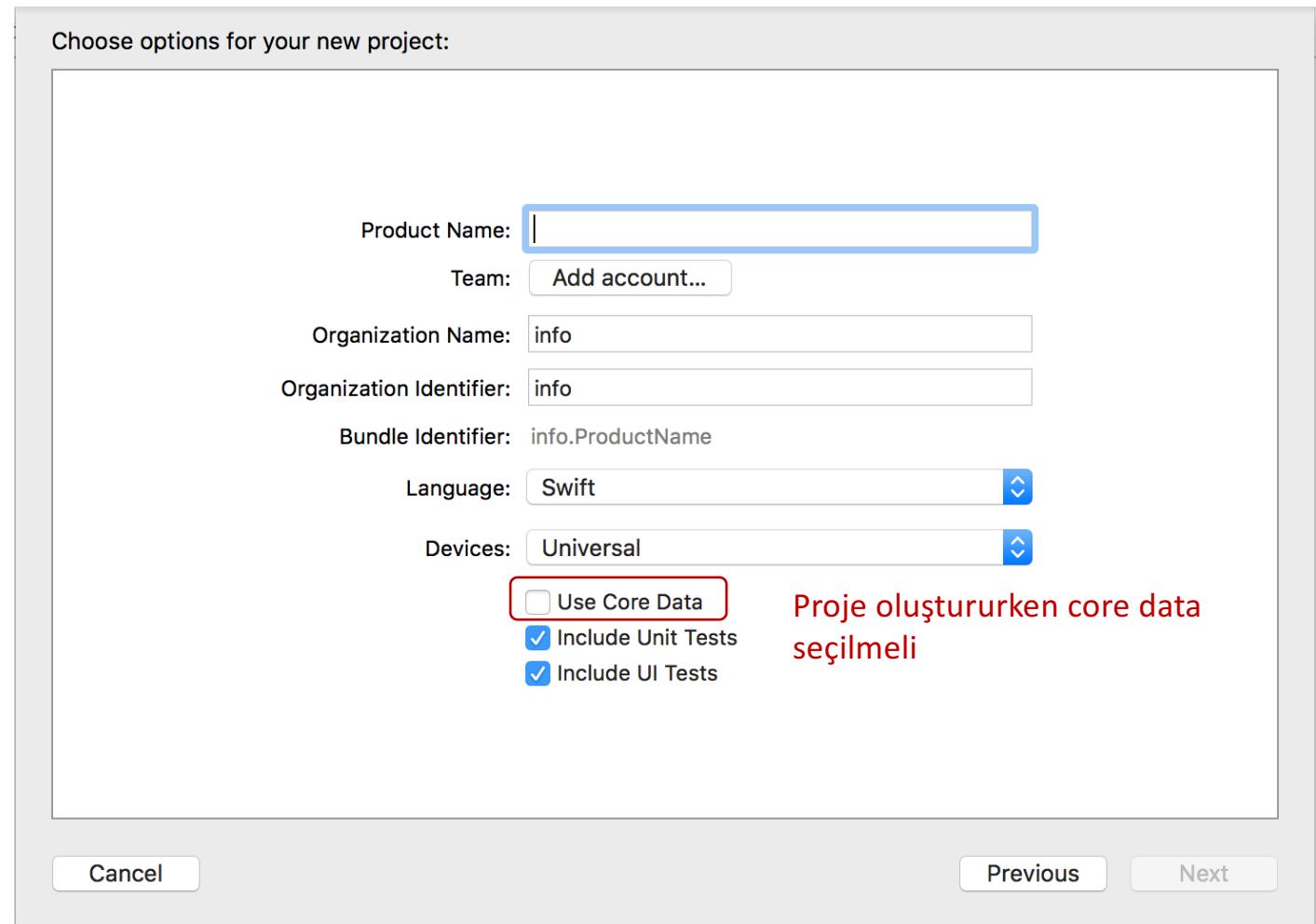
}
```

Core Data

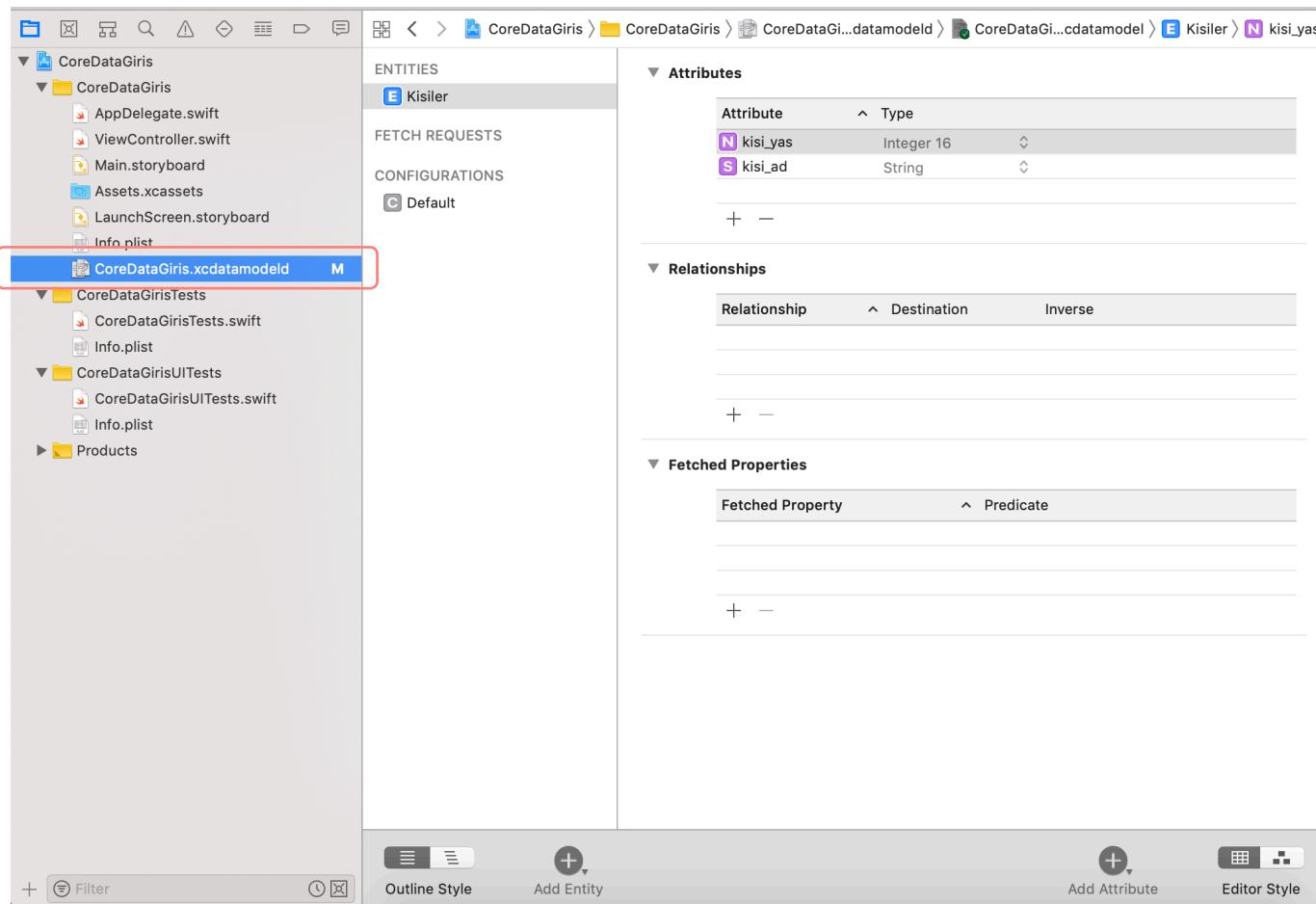
Core Data

- Core data nesnelerin düzenli şekilde saklanmasıdır.
- Core data geleneksel bir veri tabanı değildir.
- Sql sorguları içermez.
- Xcode içinde yer alan native bir framework'tur.
- Her bir veri tablosu için sınıflar oluşturmaktadır.
- Veri tabloları arasındaki ilişki nesne tabanlı dillerin sahip olduğu composition'dan yararlanmaktadır.

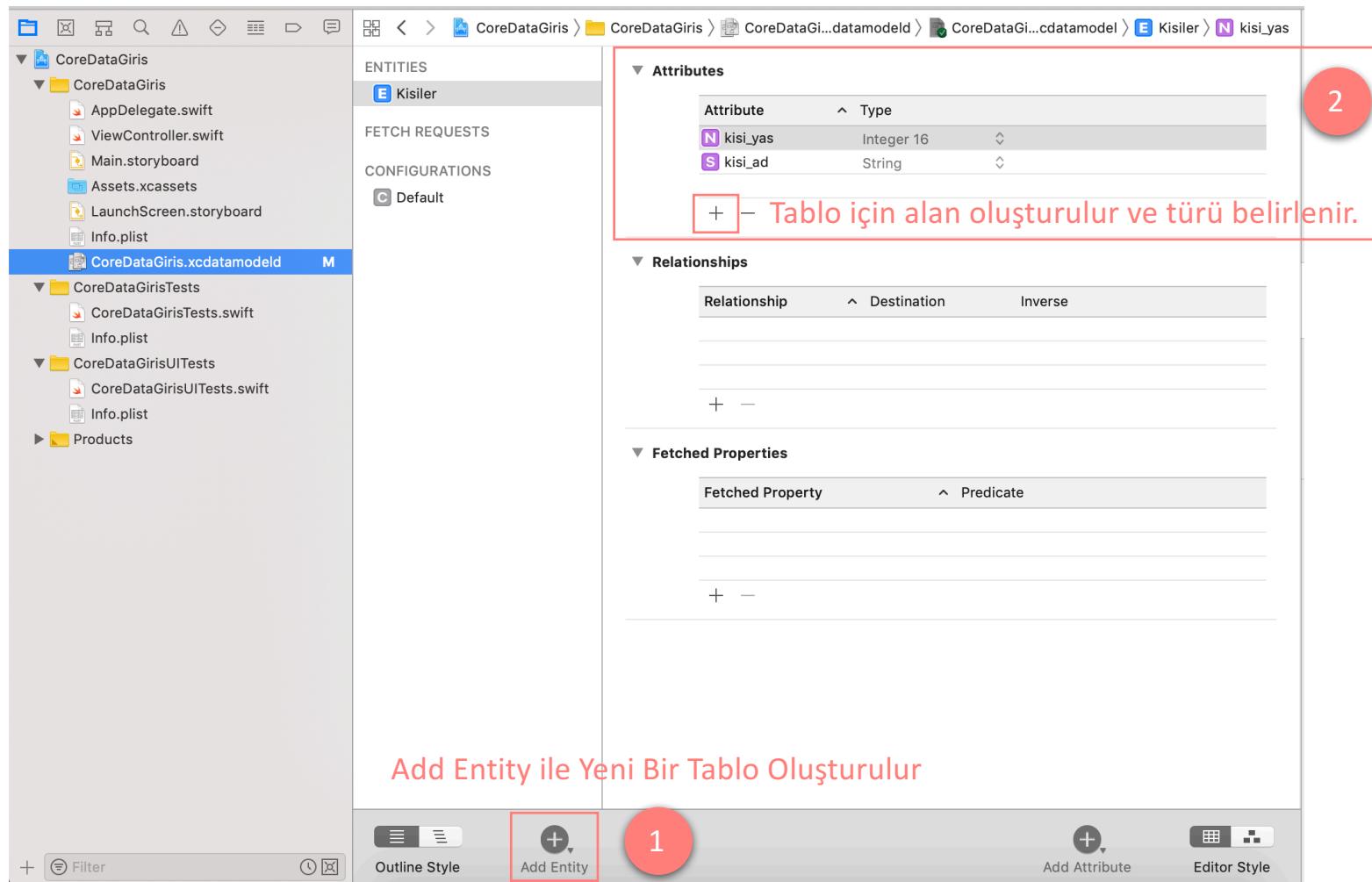
Kurulum



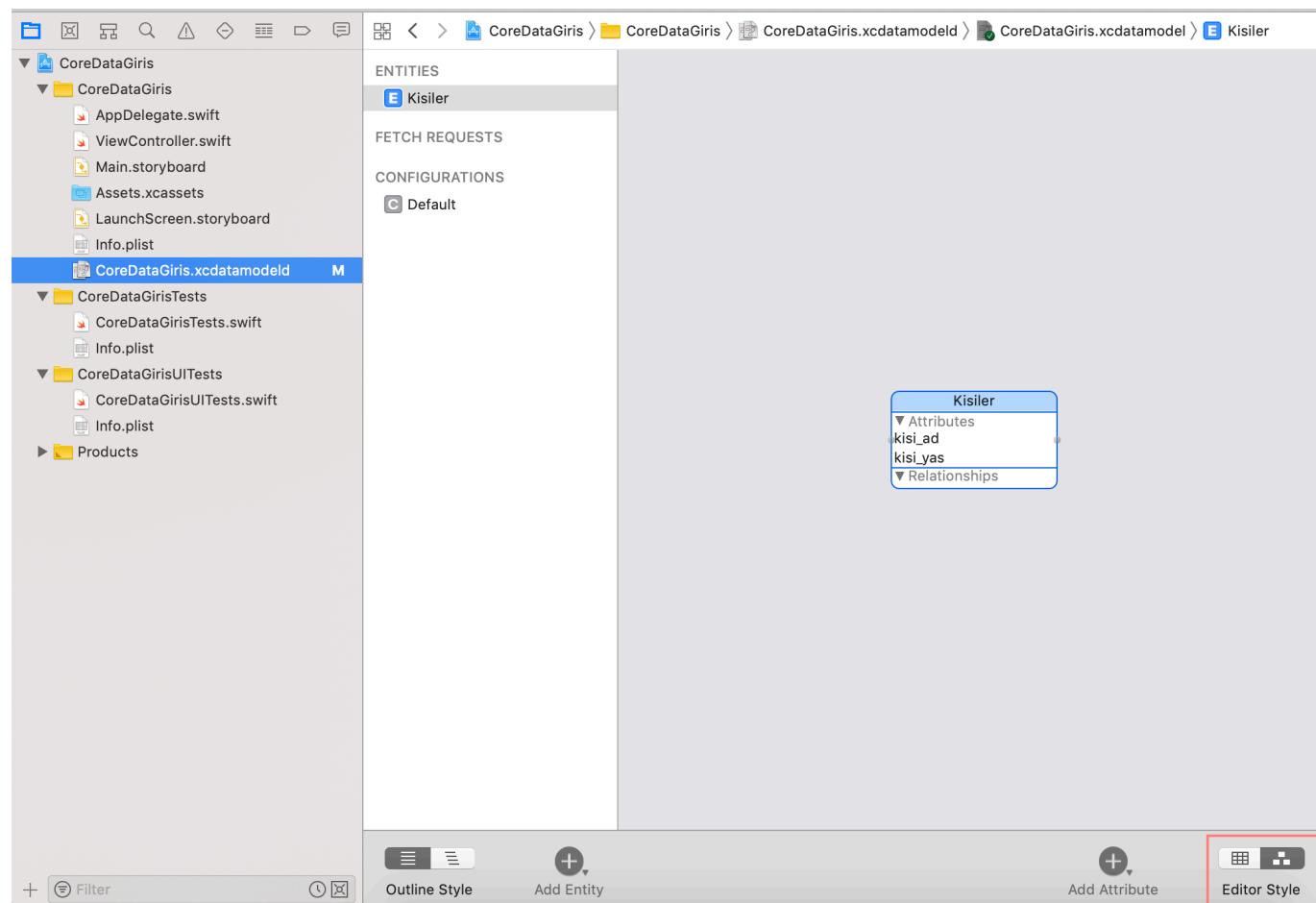
Core Data Editörün Açılmasi



Tablo Oluşturma

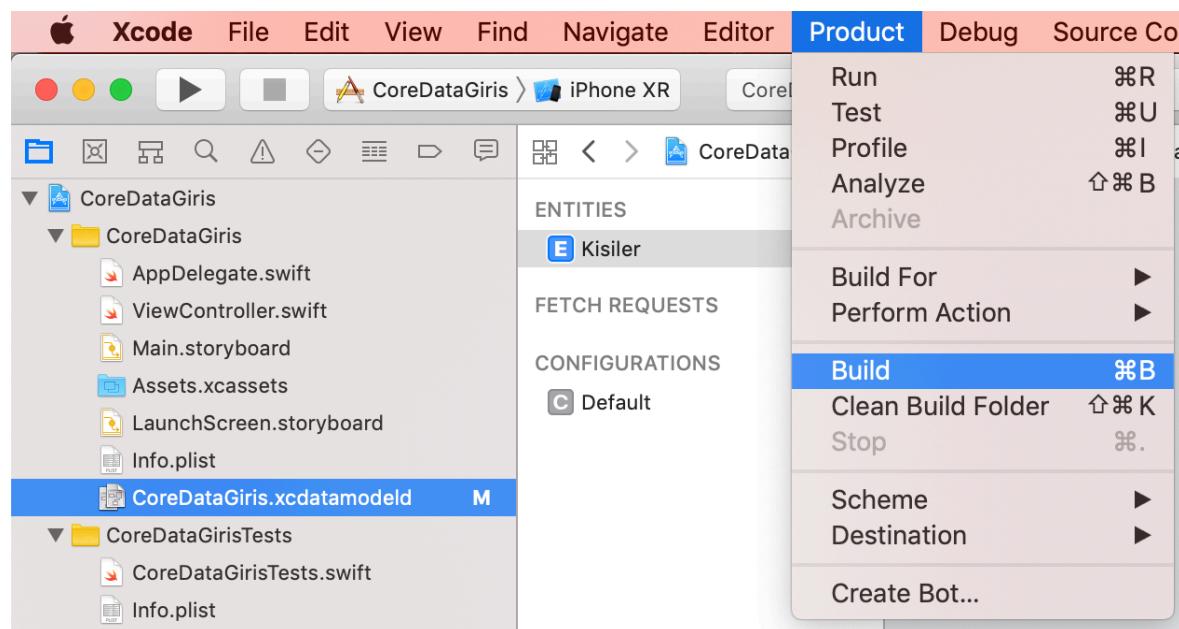


Tabloların Görüntülenme Şeklinin Değiştirilmesi



Veritabanının Xcode içinde aktif edilmesi

- Yapılan değişikler sonucunda projemizi build etmemiz gereklidir aksi halde gerekli kodlara erişemeyiz.



Kodal Kurulum

```
import UIKit
```

```
import CoreData
```

Kütüphane import edilmelidir.

Her işlem için kullanılmamaktadır ama gerekli olabileceği için eklenir.

```
let appDelegate = UIApplication.shared.delegate as! AppDelegate  
//AppDelegate nesnesi gereklidir.
```

```
class ViewController: UIViewController {
```

```
    let context = appDelegate.persistentContainer.viewContext  
    //context ile işlem yapılabilir.
```

Veri Kaydı

```
let kisi = Kisiler(context: context)
//Kisiler sınıfı oluşturduğumu tabloyu temsil eder.
//Kişiler sınıfın alanlarına veriler eklenir.Yani tablo alanlarına
kisi.kisi_ad = "Ahmet"
kisi.kisi_yas = 16

// Veri kaydı yapılır.
appDelegate.saveContext()
```

Veri Okuma

```
let appDelegate = UIApplication.shared.delegate as! AppDelegate  
//AppDelegate nesnesi gereklidir.  
  
class ViewController: UIViewController {  
  
    let context = appDelegate.persistentContainer.viewContext  
    //context ile işlem yapılabilir.  
  
    var kisilerListe = [Kisiler]()  
    //Tablo alanlarını temsil eden sınıfından oluşan bir liste.  
    //Kayıtlar bu liste içinde tutulacak
```

```
do {  
    kisilerListe = try context.fetch(Kisiler.fetchRequest())  
    //İlgili tablodan bütün veriler alınır ve listeye aktarılır.  
} catch {  
    print("Fetching Failed")  
}  
  
//Verileri gösterme  
for k in kisilerListe {  
    print("Ad : \(k.kisi_ad!) - Yaş : \(k.kisi_yas)")  
}
```

Veri Silme

```
let kisi = kisilerListe[1]
//Belirtilen indeksteki kisi nesnesi alınır.
self.context.delete(kisi)
//Silme işlemi gerçekleşir.
appDelegate.saveContext()
```

Veri Güncelleme

```
let kisi = kisilerListe[1]
//Belirtilen indeksteki kisi nesnesi alınır.
kisi.kisi_ad = "Güncel Ad"
kisi.kisi_yas = 99
//Nesneye yeni veriler eklenir.
appDelegate.saveContext()
```

Veri Sıralama (Sort)

```
import CoreData

//Bu işlemi yapmak için coredata import edilmelidir.
let fetchRequest: NSFetchedRequest<Kisiler> = Kisiler.fetchRequest()

//Hangi alan üzerinde sıralama yapacağımızı belirtiyoruz.
//Kisiler tablosunun kisi_yas alanında sıralama yapılacak.
//ascending: true -> küçükten büyüğe
//ascending: false -> büyükten küçüğe
let sort = NSSortDescriptor(key: #keyPath(Kisiler.kisi_yas), ascending: true)

fetchRequest.sortDescriptors = [sort] //Sıralama fetchRequest nesnesine eklenir.

do {
    kisilerListe = try context.fetch(fetchRequest)
    //Yeni fetchRequest nesnesi ile veri alırken sıralama değişir.
} catch {
    print("Cannot fetch Expenses")
}

//Verileri gösterme
for k in kisilerListe {
    print("Ad : \(k.kisi_ad!) - Yaş : \(k.kisi_yas)")
}
```

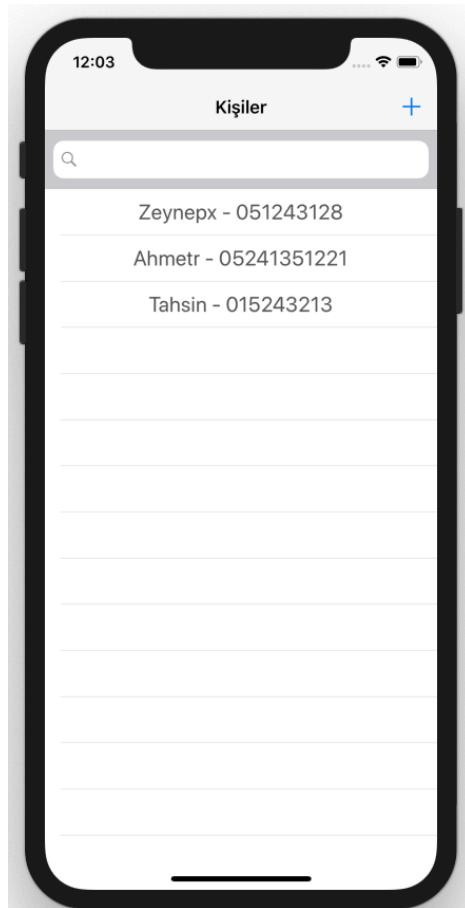
Veri filtreleme (Predicate) : Sorgu

```
//@ = String i = integer Temsil eder.  
let fetchRequest: NSFetchedRequest<Kisiler> = Kisiler.fetchRequest()  
  
//Sorgu oluşturulur.kisi yaşı 34'e eşit olan veriler almak için.  
fetchRequest.predicate = NSPredicate(format: "kisi_yas == %i", 34)  
  
do {  
    kisilerListe = try context.fetch(fetchRequest)  
    //Yeni fetchRequest nesnesi ile sorgu sonucu veriler alınır.  
} catch {  
    print("Cannot fetch Expenses")  
}  
  
//Verileri gösterme  
for k in kisilerListe {  
    print("Ad : \(k.kisi_ad!) - Yaş : \(k.kisi_yas)")  
}
```

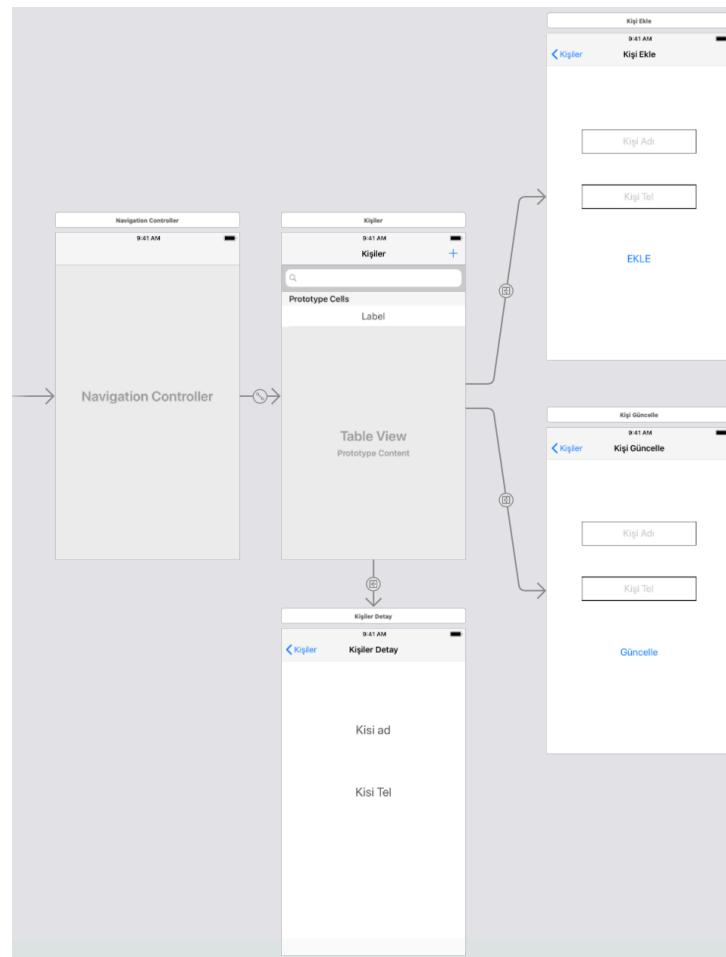
Çeşitli Sorgu Yapıları

```
//@ = String  i = integer
let fetchRequest: NSFetchedRequest<Kisiler> = Kisiler.fetchRequest()
//sorgu oluşturulur.kisi yaşı 26 eşitse
fetchRequest.predicate = NSPredicate(format: "kisi_yas == %i", 26)
//kişinin adı ahmet'e eşitse
fetchRequest.predicate = NSPredicate(format: "kisi_ad == %@", "ahmet")
//kişinin yaşı 18 den büyükse
fetchRequest.predicate = NSPredicate(format: "kisi_yas > %i", 18)
//kişinin adı ahmet'e eşitse
fetchRequest.predicate = NSPredicate(format: "%K == %i", "kisi_ad", "ahmet")
//kişinin adı a harfini içeriyorsa
fetchRequest.predicate = NSPredicate(format: "kisi_ad CONTAINS %@", "a")
//iki şart kullanılabilir.
fetchRequest.predicate = NSPredicate(format: "kisi_yas == %i and kisi_ad == %@", 26, "ahmet")
//ilişkisi olduğu tablodan sorgu olabilir.
//composition yaparak erişim sağlanır.
fetchRequest.predicate = NSPredicate(format: "kisiler_is.is_ad == %@", "spor")
```

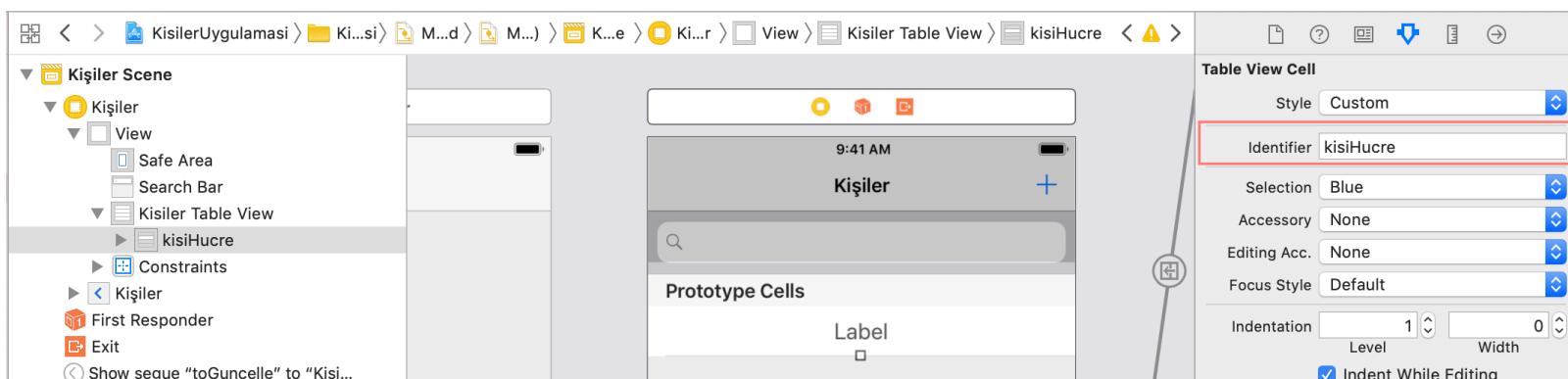
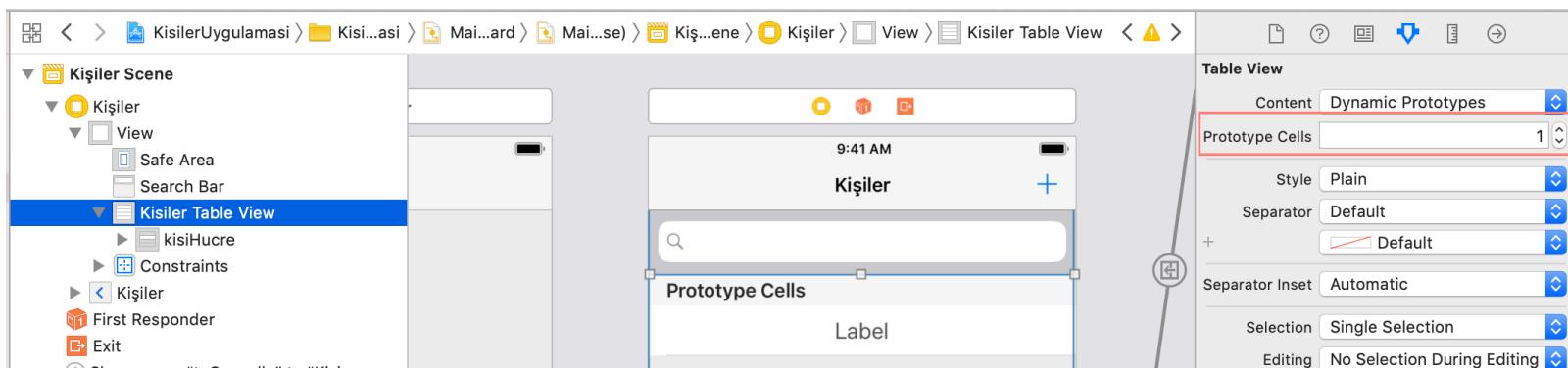
Uygulama : Kişiler Uygulaması



Genel Tasarım Yapısı



Custom TableView Cell İşlemleri



Custom TableView Cell Sınıfı

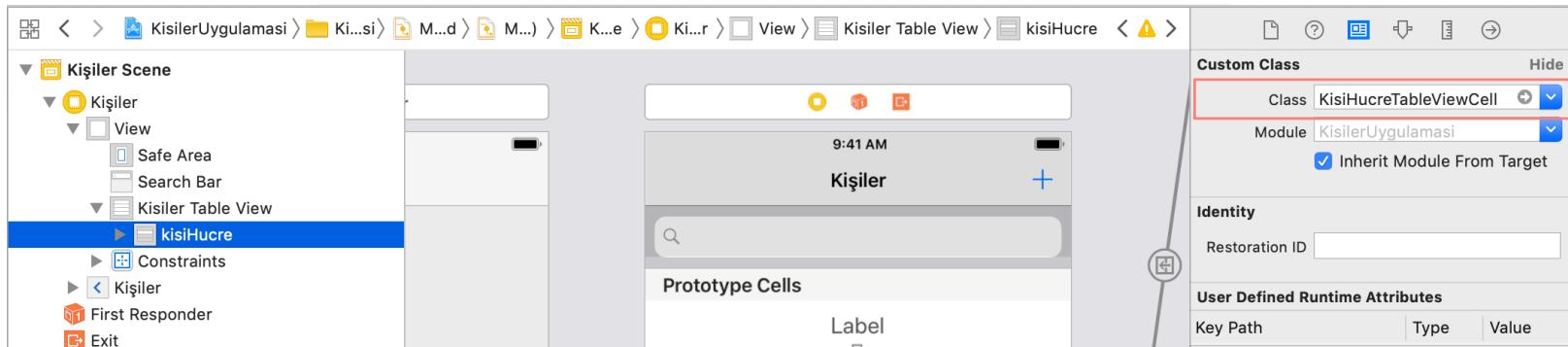
```
import UIKit

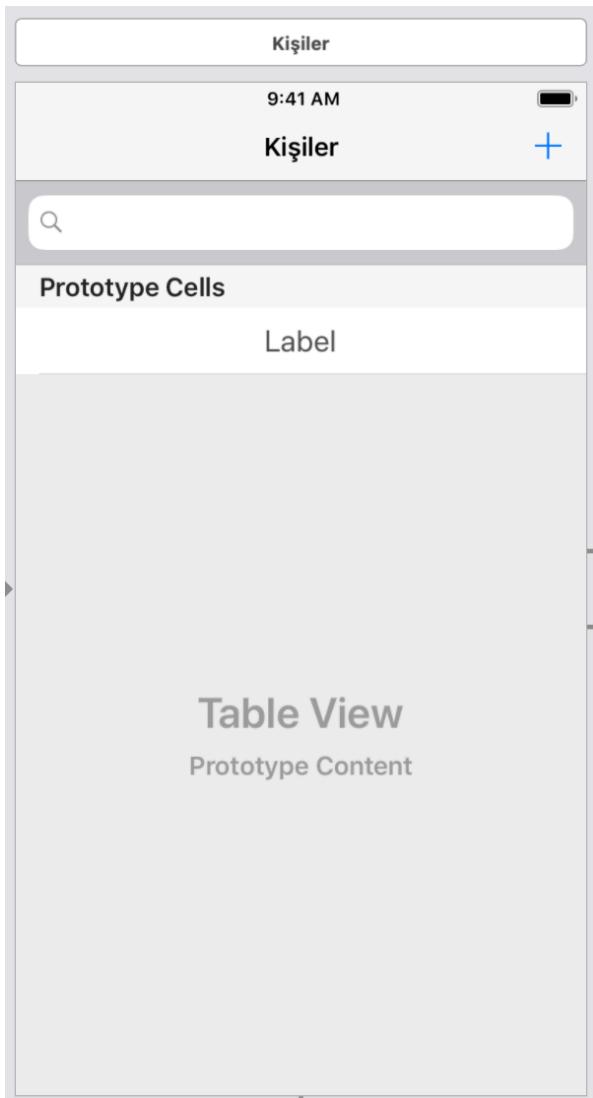
class KisihucreTableViewCell: UITableViewCell {

    @IBOutlet weak var kisiYaziLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
    }
}
```





```
import UIKit
import CoreData

let appDelegate = UIApplication.shared.delegate as! AppDelegate
//AppDelegate nesnesi gereklidir. Globaldir her viewcontrollerden erisilebilir.

class ViewController: UIViewController {

    let context = appDelegate.persistentContainer.viewContext
    //context ile işlem yapılabilir.
    @IBOutlet weak var searchBar: UISearchBar!
    @IBOutlet weak var kisilerTableView: UITableView!

    var kisilerListe = [Kisiler]()//Gelecek veri için liste oluşturulur.
    var aramaYapiliyorMu = false//Bu metod çalıştırısa arama yapılıyor demektir.
    var aramaKelimesi:String?//Arama yapılan kelime

    override func viewDidLoad() {
        super.viewDidLoad()
        kisilerTableView.delegate = self//Protocol ile tableview bağlanır.
        kisilerTableView.dataSource = self//Protocol metodlarının çalışması için bu bağlantı önemlidir.
        searchBar.delegate = self
        //Arama protocolune gorsel nesne bağlanır.
    }
    override func viewDidAppear(_ animated: Bool) {
        //Başka bir sayfadan geri gelindiğinde bu metod çalışır.
        //Arama yapılarak geçiş yapıldıysa ona göre geri dönüş sağlanır.
        //Arama kelimesi ve ona göre sonucu geri dönüldüğünde orda olmalıdır.
        if aramaYapiliyorMu {
            aramaYap(kisi_ad: aramaKelimesi!)
        }else{
            tumKisilerAl()
        }
        kisilerTableView.reloadData()//Veri alındıktan sonra veriyi table içine aktarır.
    }
    func tumKisilerAl() {
        do {
            kisilerListe = try context.fetch(Kisiler.fetchRequest())
        } catch {print("Fetching Failed")}
    }
}
```

```
func aramaYap(kisi_ad:String) {
    // @ = String i = integer Temsil eder.
    let fetchRequest: NSFetchedResultsController<Kisiler> = Kisiler.fetchRequest()

    fetchRequest.predicate = NSPredicate(format: "kisi_ad CONTAINS %@", kisi_ad.lowercased())

    do {
        self.kisilerListe = try context.fetch(fetchRequest)
        // Yeni fetchRequest nesnesi ile sorgu sonucu veriler alınır.
    } catch {print("Cannot fetch Expenses")}
}

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    let indeks = sender as? Int

    if segue.identifier == "toGuncelle" {
        let gidilecekVC = segue.destination as! KisiGuncelleViewController
        gidilecekVC.kisi = self.kisilerListe[indeks!]

    }

    if segue.identifier == "toDetay" {
        let gidilecekVC = segue.destination as! KisiDetayViewController
        gidilecekVC.kisi = self.kisilerListe[indeks!]
    }
}
```

```
//TableView Protocol Metodları
extension ViewController:UITableViewDelegate,UITableViewDataSource{

    func numberOfSections(in tableView: UITableView) -> Int {
        return 1 //Tableview içindeki bölüm sayısı
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return kisilerListe.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
        UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "kisiHucre", for: indexPath) as!
            KisiHucreTableViewCell

        var kisi = Kisiler()
        kisi =  kisilerListe[indexPath.row]

        cell.kisiYaziLabel.text = "\(kisi.kisi_ad!) - \(kisi.kisi_tel!)"

        return cell
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        self.performSegue(withIdentifier: "toDetay", sender: indexPath.row)
        //Detay ekranına geçiş yaparken seçilen kisinin index numarasında prepare metoduna
        //gönderir.
        //Çünkü veri transferi yapılacaktır.
    }
}
```

```
func tableView(_ tableView: UITableView, editActionsForRowAt indexPath: IndexPath) -> [UITableViewCellAction] {
    let silAction = UITableViewRowAction(style: .default, title: "Sil" , handler: { (action:UITableViewRowAction, indexPath: IndexPath) -> Void in
        var kisi = Kisiler()
        kisi = self.kisilerListe[indexPath.row]
        self.context.delete(kisi)
        //Silme işlemi gerçekleşir.
        appDelegate.saveContext()
        //Silme yapıldıktan sonra silme işlemi arama yapıldıktan sonra ise ona göre liste gösterilmelidir.
        if self.aramaYapiliyorMu {
            self.aramaYap(kisi_ad: self.aramaKelimesi!)
        }else{
            self.tumKisilerAl()
        }
        tableView.reloadData()
        //Kişiler silindiğten sonra verileri tekrar almalıyız.
        //Aksi takdirde verilerin en son halini göremeyiz
    })
    let duzenleAction = UITableViewRowAction(style: .normal, title: "Güncelle" , handler: { (action:UITableViewRowAction, indexPath:IndexPath) -> Void in
        self.performSegue(withIdentifier: "toGuncelle", sender: indexPath.row)
        //Güncelle ekrانına geçiş yaparken seçilen kişinin index numarasında prepare metoduna gönderir.
        //Çünkü veri transferi yapılacaktır.
    })
    //Oluşturulan action'lar tableview'e eklenir.
    return [silAction,duzenleAction]
}
```

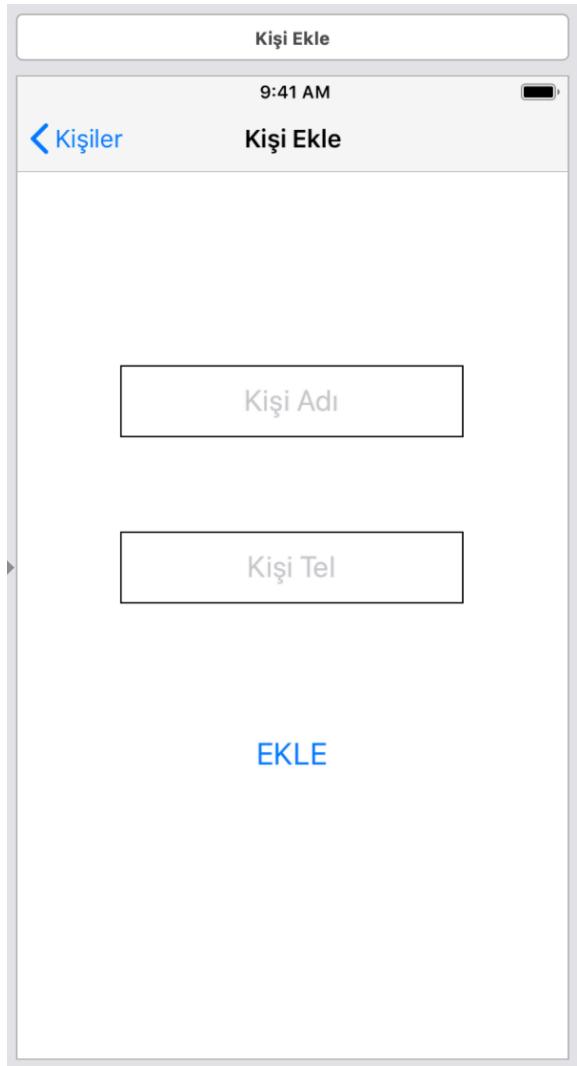
```
//SearchBar Protocol Metodları
extension ViewController: UISearchBarDelegate {

    func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {
        //Arama işlemi bu metodu herf harf girildiğinde çalıştırır.
        print("Arama Sonucu : \(searchText)")

        aramaKelimesi = searchText//Arama kelimesine arama key aktarılır.

        if searchText == "" {//Arama alanı boş ise arama yapılmıyor.
            aramaYapiliyorMu = false
            self.tumKisilerAl()
        }else{
            aramaYapiliyorMu = true
            self.aramaYap(kisi_ad: searchText)
        }

        kisilerTableView.reloadData()
        //Arama her yapıldığında tableview içindeki veri tekrar yüklenirki anlık veriler
        //değişsin.
    }
}
```



```
import UIKit

class KisiEkleViewController: UIViewController {

    let context = appDelegate.persistentContainer.viewContext
    //context ile işlem yapılabilir.

    @IBOutlet weak var kisiAdTextfield: UITextField!

    @IBOutlet weak var kisiTelTextfield: UITextField!

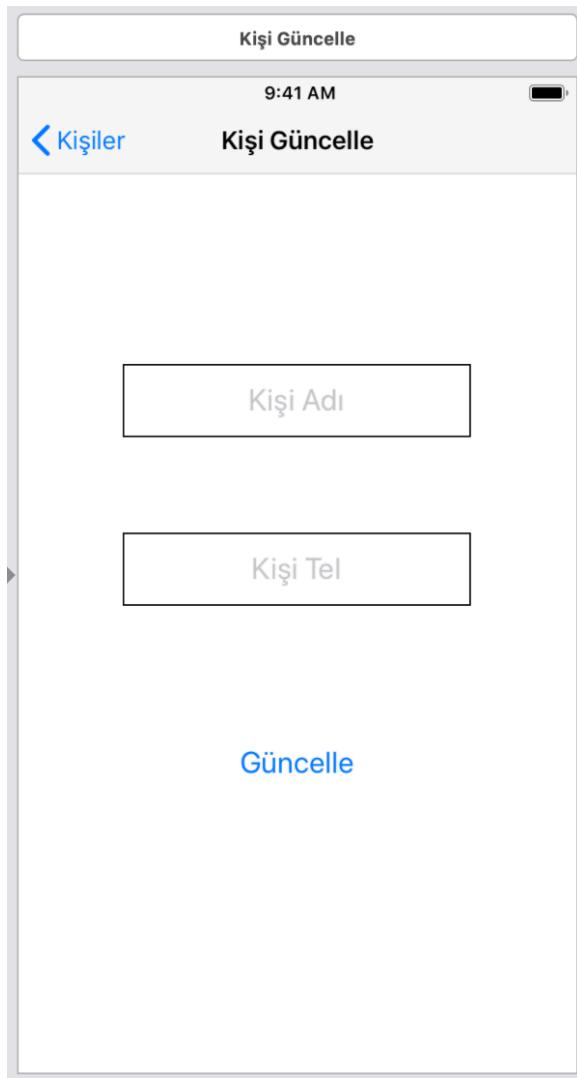
    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }
    @IBAction func ekle(_ sender: Any) {

        let kisi = Kisiler(context: context)

        kisi.kisi_ad = kisiAdTextfield.text
        kisi.kisi_tel = kisiTelTextfield.text
        // Veri kaydı yapılır.
        appDelegate.saveContext()

    }
}
```



```
import UIKit

class KisiGuncelleViewController: UIViewController {

    @IBOutlet weak var kisiAdTextfield: UITextField!
    @IBOutlet weak var kisiTelTextfield: UITextField!

    var kisi:Kisiler?

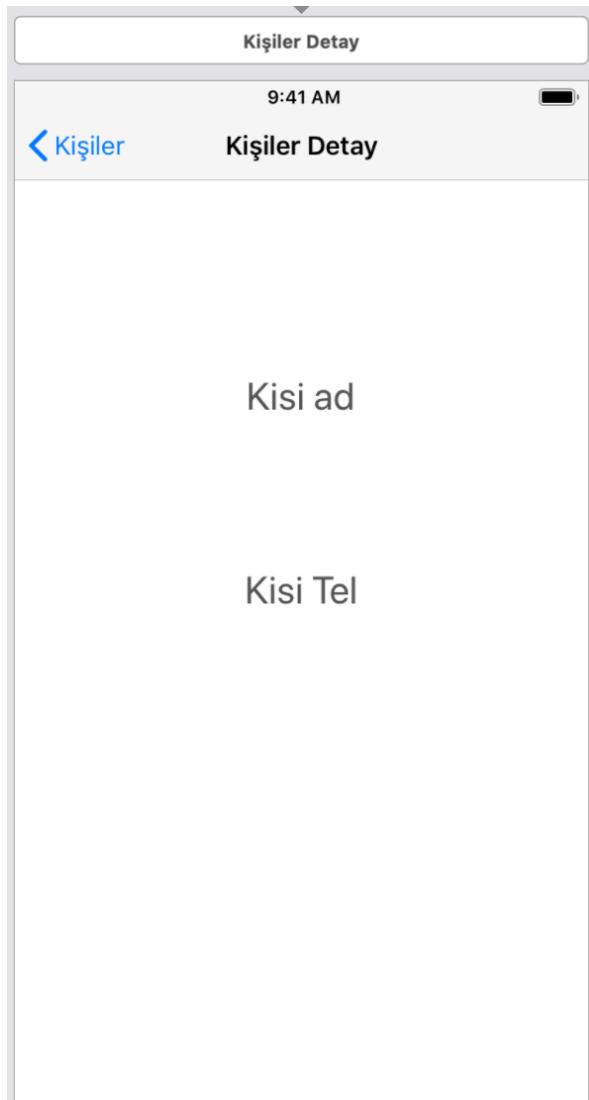
    override func viewDidLoad() {
        super.viewDidLoad()

        //Güncellemeyi kolaylaştırmak için textfield içine veriler eklenir
        kisiAdTextfield.text = kisi?.kisi_ad
        kisiTelTextfield.text = kisi?.kisi_tel
    }

    @IBAction func guncelle(_ sender: Any) {

        //Gönderilmiş kişi nesnesini güncelliyoruz.
        self.kisi!.kisi_ad = kisiAdTextfield.text
        self.kisi!.kisi_tel = kisiTelTextfield.text
        //Nesneye yeni veriler eklenir.
        appDelegate.saveContext()
    }

}
```



```
import UIKit

class KisiDetayViewController: UIViewController {

    @IBOutlet weak var kisiAdLabel: UILabel!
    @IBOutlet weak var kisiTelLabel: UILabel!

    var kisi:Kisiler?

    override func viewDidLoad() {
        super.viewDidLoad()

        kisiAdLabel.text = kisi?.kisi_ad
        kisiTelLabel.text = kisi?.kisi_tel
    }
}
```

SQLite

SQLite Tablo Oluşturma

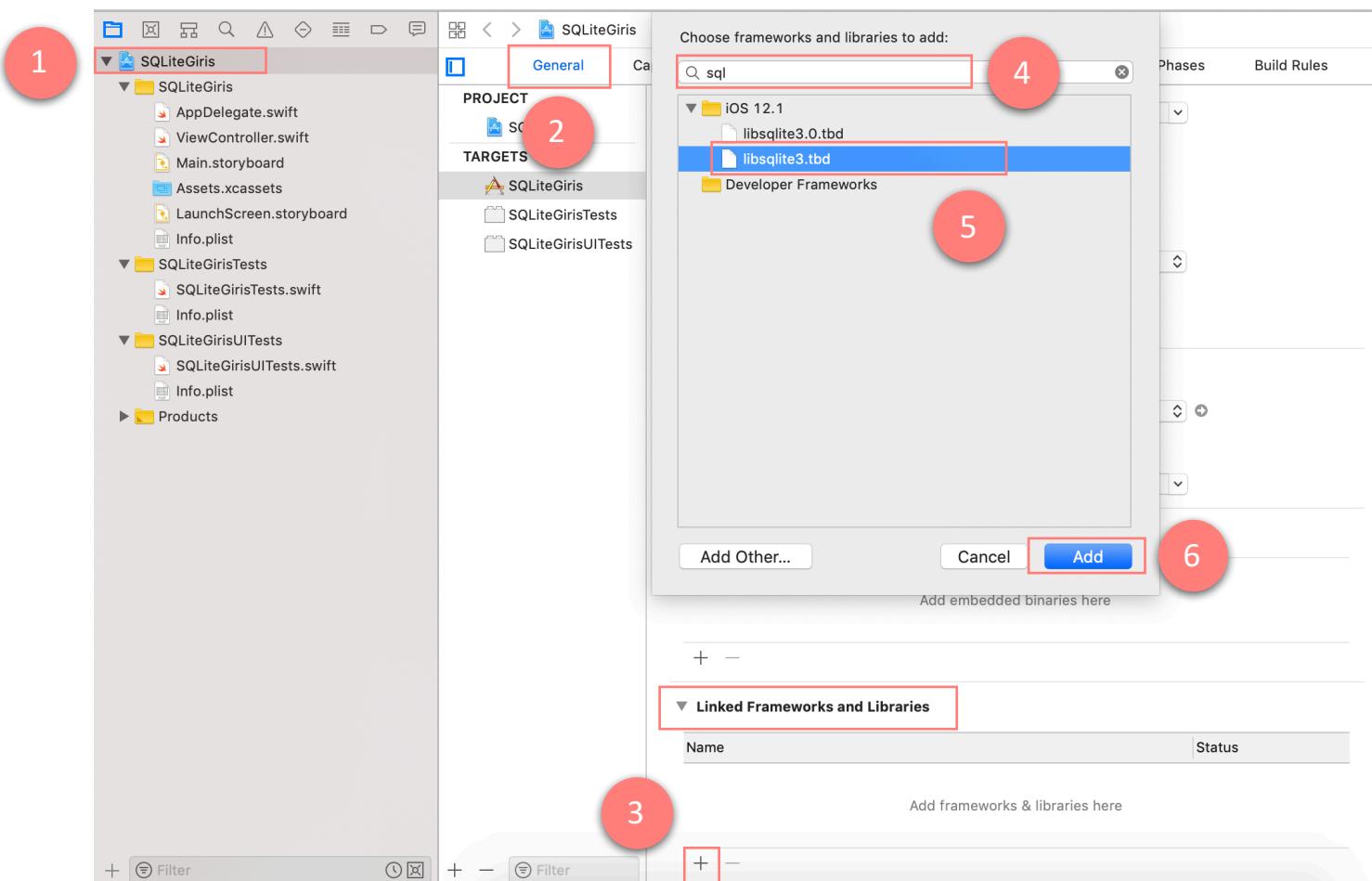
```
CREATE TABLE yonetmenler(  
    yonetmen_id INTEGER PRIMARY KEY AUTOINCREMENT,  
    yonetmen_ad TEXT  
);  
  
CREATE TABLE kategoriler(  
    kategori_id INTEGER PRIMARY KEY AUTOINCREMENT,  
    kategori_ad TEXT  
);  
  
CREATE TABLE filmler (  
    film_id INTEGER PRIMARY KEY AUTOINCREMENT,  
    film_ad TEXT,  
    film_yil TEXT,  
    film_resim TEXT,  
    kategori_id INTEGER,  
    yonetmen_id INTEGER,  
    FOREIGN KEY (kategori_id) REFERENCES kategoriler(kategori_id),  
    FOREIGN KEY (yonetmen_id) REFERENCES yonetmenler(yonetmen_id)  
);
```

SQLite İlişki İçeren Tablodan Veri Alma

```
SELECT * FROM filmler, yonetmenler, kategoriler  
WHERE filmler.yonetmen_id = yonetmenler.yonetmen_id  
AND filmler.kategori_id = kategoriler.kategori_id
```

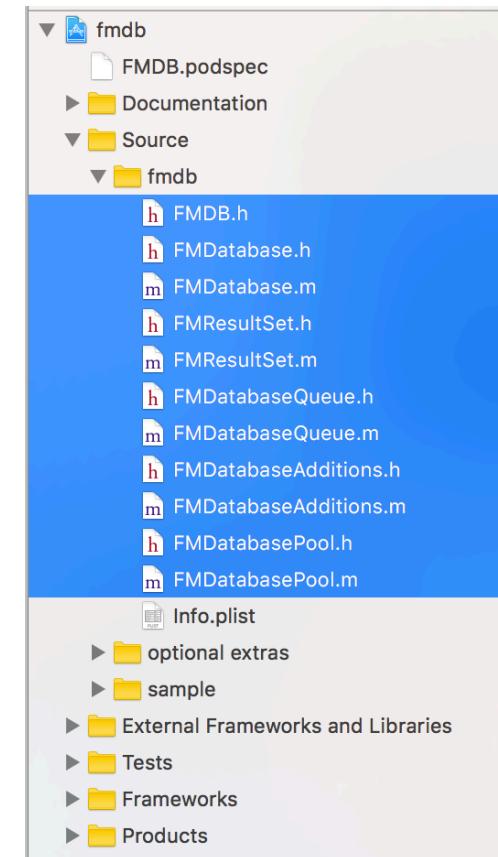
SQLite Kurulumu

Sqlite Kütüphanesinin Projeye Eklenmesi

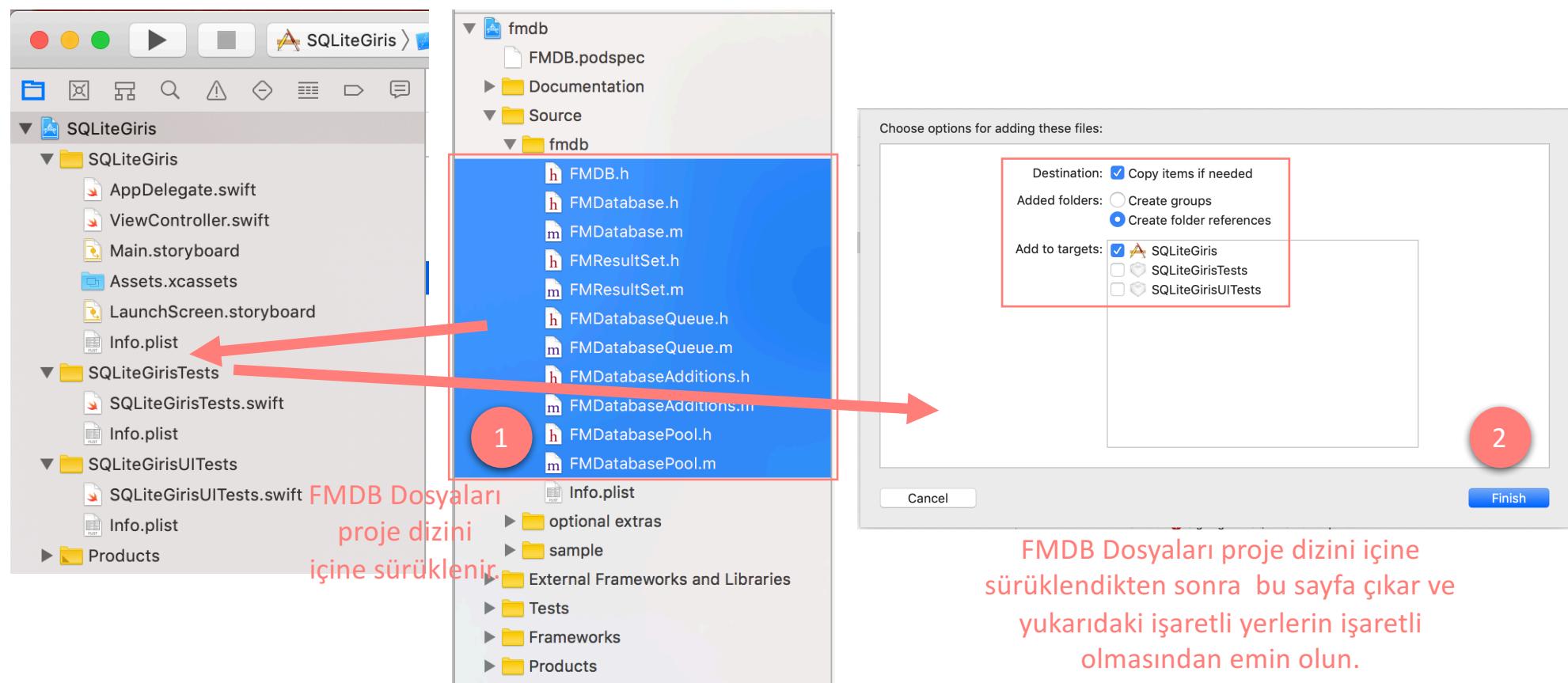


FMDB Dosyaları

- FMDB dosyaları <https://github.com/ccgus/fmdb.git> üzerinden indirilebilir.
- Bu dosyalar Objective – c dili ile yazılmıştır.
- Dosyalar sol tarafta mavi seçili olan dosyalardır.
- Bu dosyalar resimde olduğu gibi seçilir ve projeye eklenir.
- Eklenirken **if needed copy items** seçeneği seçili olmalıdır.

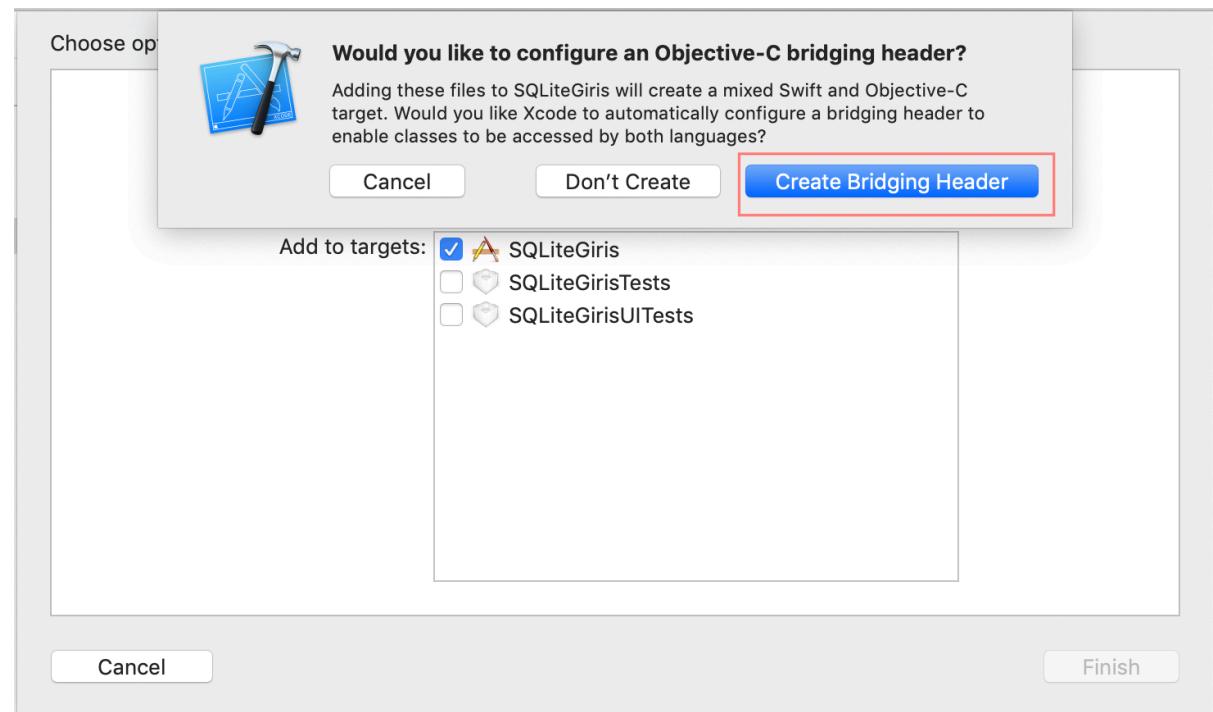


FMDB Dosyalarının Xcode Projesine Eklenmesi



FMDB Dosyaları ile Xcode Arasında Köprü Oluşturma

- FMDB Dosyaları proje eklendikten sonra Xcode üzerinde uyarı çıkar.
- Çıkan uyarıda Create Bridging Header seçersek FMDB Dosyaları ile Xcode arasında köprü oluşturur.
- Bu köprünün amacı Objective C ile Swift arasındaki uyumu sağlar.



Köprü Dosyasının Kodlanması

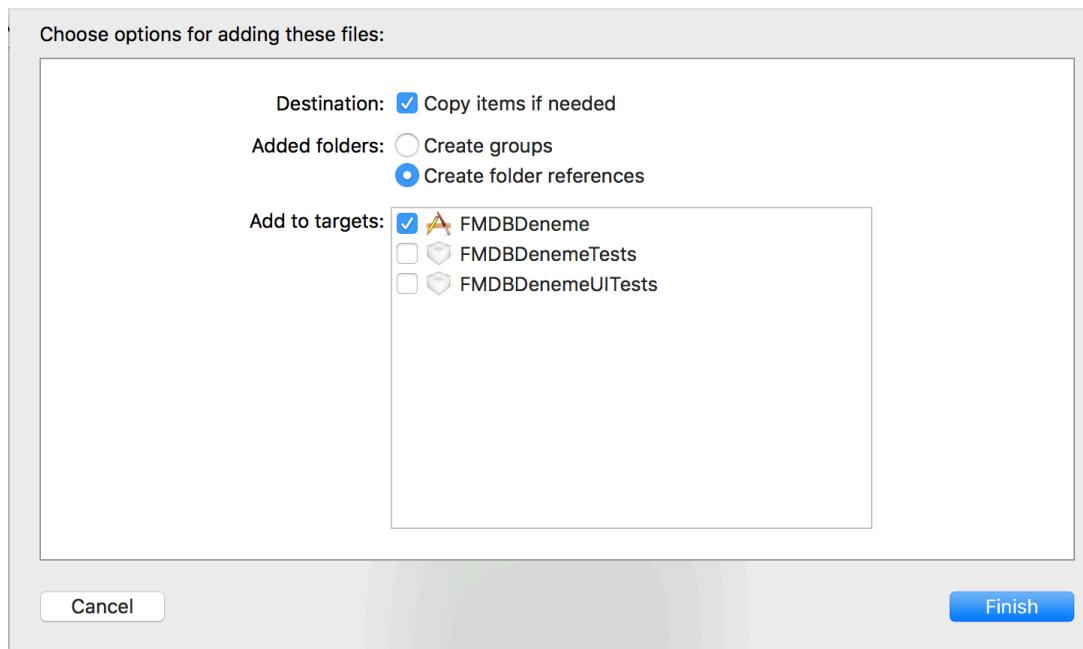
- Köprü dosyasının içine `#import "FMDB.h"` ekliyor ve Xcode projesini kaydediyoruz.

The screenshot shows the Xcode interface with the project navigation bar at the top. Below it, the project structure is displayed in a tree view. Under the 'SQLiteGiris' folder, there is a 'SQLiteGiris' group containing several Swift files: AppDelegate.swift, ViewController.swift, Main.storyboard, Assets.xcassets, LaunchScreen.storyboard, and Info.plist. There are also Objective-C header files: FMDatabase.h, FMDatabase.m, FMDatabaseAdditions.h, FMDatabaseAdditions.m, FMDatabasePool.h, FMDatabasePool.m, FMDatabaseQueue.h, FMDatabaseQueue.m, FMDB.h, FMResultSet.h, and FMResultSet.m. At the bottom of the tree view, the 'SQLiteGiris-Bridging-Header.h' file is selected and highlighted with a blue background and a red border around its name. In the main editor area to the right, the content of this file is shown:

```
1 //  
2 // Use this file to import your target's public headers that you would  
3 //  
4 //  
5 #import "FMDB.h"  
6
```

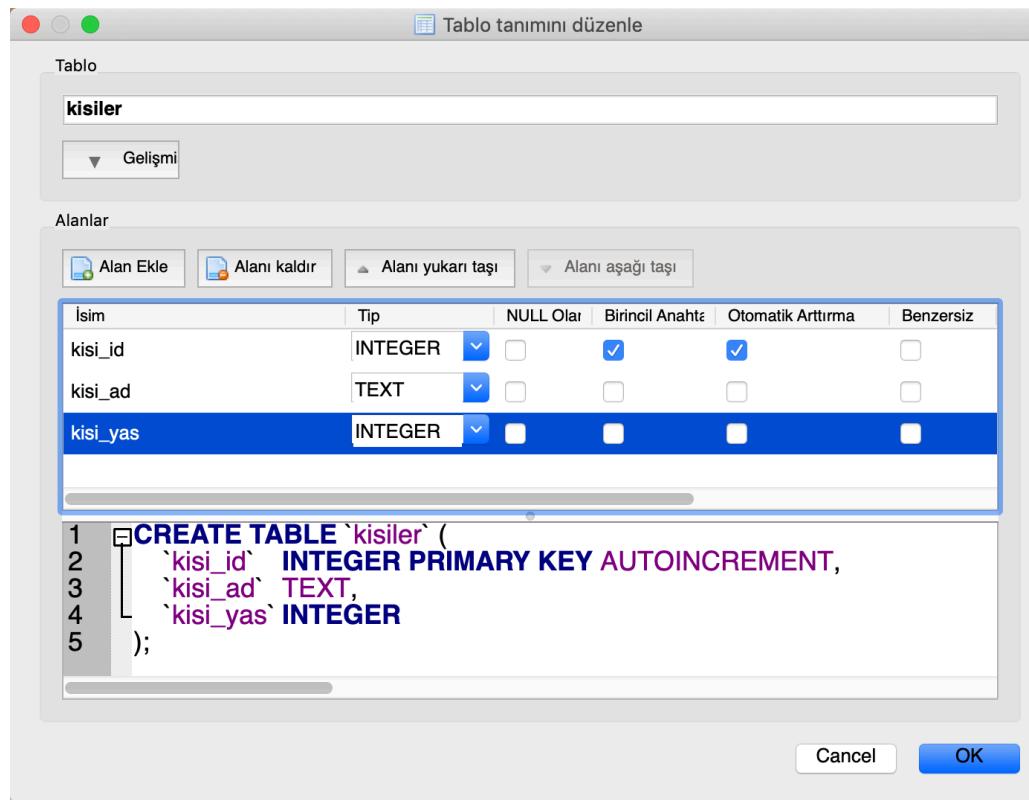
SQLite Veri tabanının Projemize Eklenmesi

- Bu işlemi yapmadan önce veri tabanımızı oluşturmalıyız.
- Oluşturduğumuz Sqlite veri tabanını Xcode projemizin içine sürüklüyoruz.
- Kopyalama işlemi unutulmamalıdır bunun için resimdeki yerler seçili olmalıdır.



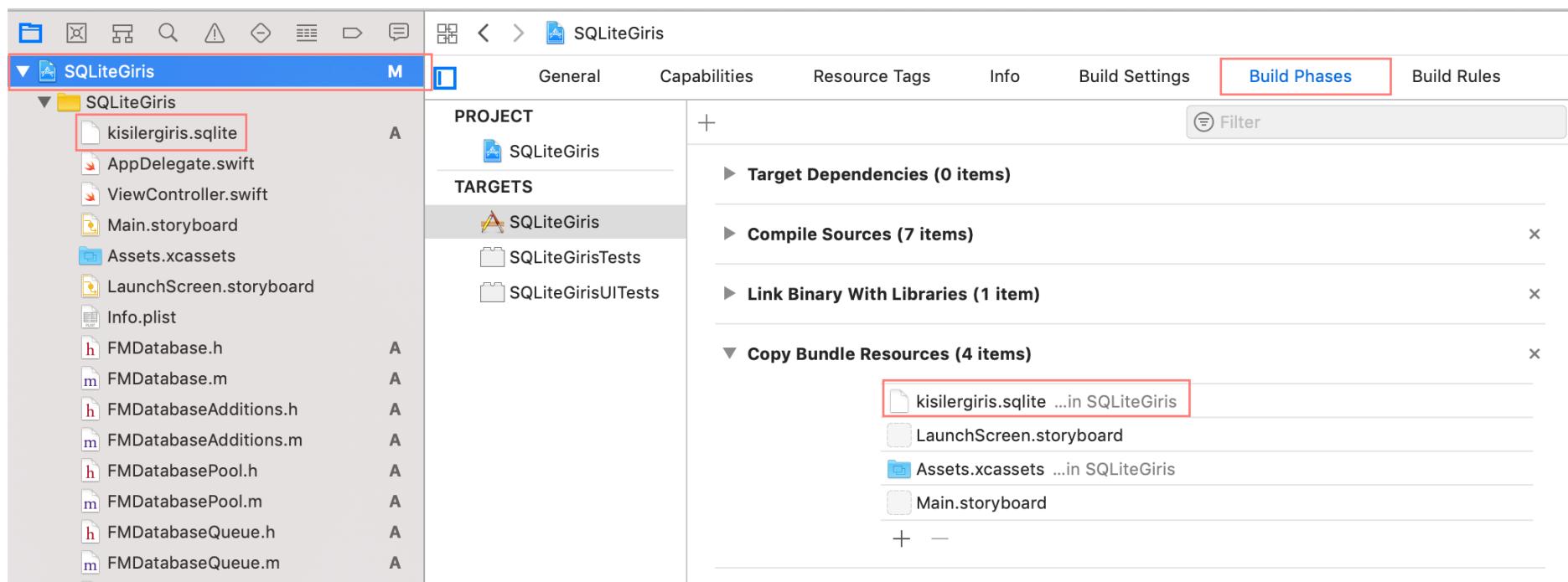
Veri tabanı Oluşturma

- DB Browser for SQLite programı üzerinde veri tabanı ve tablolar oluşturulmalıdır.



Veri tabanının Xcode projesi **Bundle** içinde olduğunun kontrol edilmesi

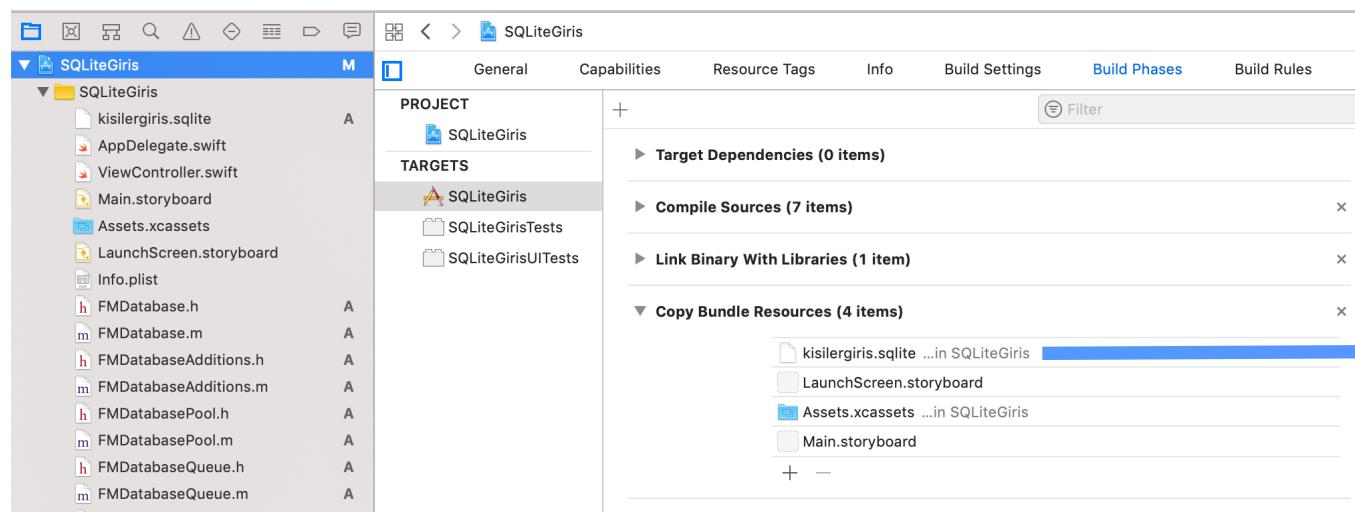
- Bundle Xcode projemiz içinde yer alan medya, veri tabanı vb dosyaların bulunduğu alandır.



Yazılımsal Kurulum

Veri Tabanının Kopyalanması

- Veri tabanı bundle içinde yer alır.
- Uygulama ilk çalıştırıldığında bundle'dan cihazın içine kopyalanmalıdır.
- Bir kere kopyalandıktan sonra kopyalandığı yerden veri tabanı üzerinde işlemler yapılabilir.
- Kopyalama işlemi uygulamanın ilk sayfası açıldığında çalıştırılmalıdır.



Veri Tabanı Kopyalama Kodlaması

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        veritabaniKopyala()
    }

    func veritabaniKopyala(){
        //Veritabanının bundle üzerindeki yeri
        let bundleYolu = Bundle.main.path(forResource: "kisilergiris", ofType: ".sqlite")

        let hedefYol = NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).first!

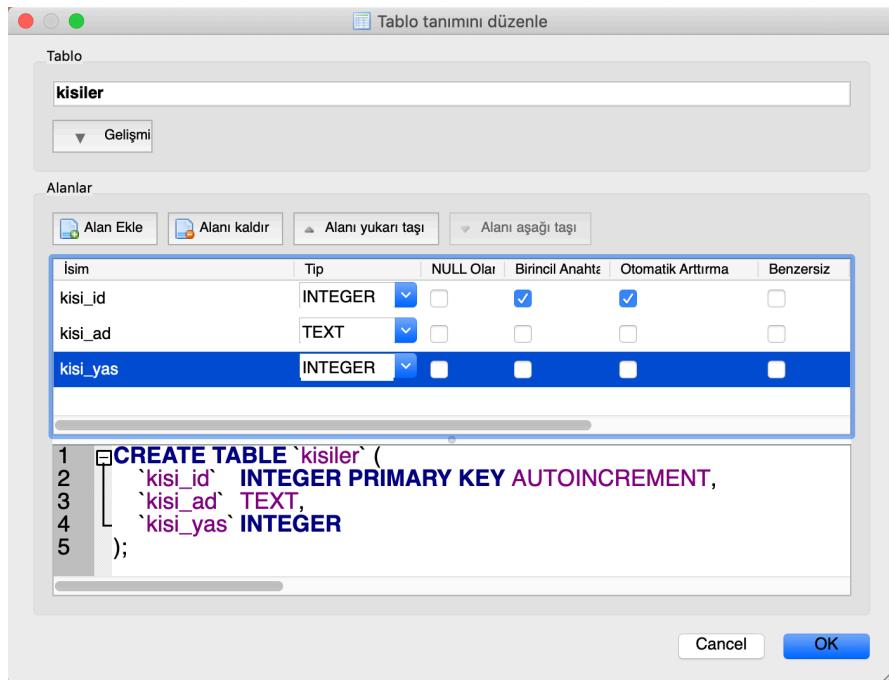
        let fileManager = FileManager.default //Kopyalama işlemi için FileManager gereklidir.

        //Kopyalama yapılacak yer
        let kopyalanacakYer = URL(fileURLWithPath: hedefYol).appendingPathComponent("kisilergiris.sqlite")

        if fileManager.fileExists(atPath: kopyalanacakYer.path) {
            //Kopyalama yaparken veritabanının daha önce kopyalanıp kopyalanmadığı sorulur.
            print("Veritabanı zaten var.Kopyalamaya Gerek Yok.")
        }else{
            //Kopyalanmadıysa kopyalama işlemi yapılır.
            do {
                try fileManager.copyItem(atPath: bundleYolu!, toPath: kopyalanacakYer.path)
                //Kopyalama işlemi
            }catch{
                print(error)
            }
        }
    }
}
```

Veri Tabanı Modeli için Swift Sınıfı

- Her bir veri tabanı tablosunu temsil eden bir swift sınıfı oluşturulmalıdır.



```
import Foundation

class Kisiler {
    var kisi_id:Int?
    var kisi_ad:String?
    var kisi_yas:Int?

    init() {}

    init(kisi_id:Int,kisi_ad:String,kisi_yas:Int) {
        self.kisi_id = kisi_id
        self.kisi_ad = kisi_ad
        self.kisi_yas = kisi_yas
    }
}
```

Veri Tabanı İşlemleri için Swift Sınıfları

- Ayrıca her bir veri tabanı tablosu üzerinde yapılacak işlemleri temsil eden sınıf oluşturulur.
- Bu sınıfı dao (database access object) sınıfı denir.

```
import Foundation

class Kisilerdao {
    let db:FMDatabase?

    init() {
        //Kopyalanmış veritabanını kopyalandığı yerden alarak kullanılmaya hazır hale getirilir.
        let hedefYol = NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).first!
        let veritabaniURL = URL(fileURLWithPath: hedefYol).appendingPathComponent("kisilergiris.sqlite")

        db = FMDatabase(path: veritabaniURL.path)//Veritabanına bağlanmak için nesne
    }

    func tumKisileriAl() -> [Kisiler]{
        var liste = [Kisiler]()
        db?.open()//Veritabanı bağlantısı açılır.

        do {
            let rs = try db!.executeQuery("SELECT * FROM kisiler", values: nil)

            while rs.next() {
                let kisi = Kisiler(kisi_id: Int(rs.string(forColumn: "kisi_id"))!
                                    , kisi_ad: rs.string(forColumn: "kisi_ad")!
                                    , kisi_yas: Int(rs.string(forColumn: "kisi_yas"))!)

                liste.append(kisi)
            }
        }catch{
            print(error.localizedDescription)
        }

        db?.close()//Veritabanı bağlantısı kapanır.

        return liste
    }
}
```

Kopyalanmış veri tabanına ismi ile her seferinde erişmeliyiz.

Veri Tabanına Erişime Yakından Bakış

```
import Foundation

class Kisilerdao {

    let db:FMDatabase?

    init() {
        //Kopyalanmış veritabanını kopyalandığı yerden alarak kullanılmaya hazır hale getirilir.
        let hedefYol = NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).first!
        let veritabaniURL = URL(fileURLWithPath: hedefYol).appendingPathComponent("kisilergiris.sqlite")

        db = FMDatabase(path: veritabaniURL.path)//Veritabanına bağlanmak için nesne
    }
}
```

SQLite Veri tabanı İşlemleri

Select : Veri Tabanından verinin alınması

```
func tumKisileriAl() -> [Kisiler]{

    var liste = [Kisiler]()

    db?.open()//Veritabanı bağlantısı açılır.

    do {
        let rs = try db!.executeQuery("SELECT * FROM kisiler", values: nil)

        while rs.next() {
            let kisi = Kisiler(kisi_id: Int(rs.string(forColumn: "kisi_id"))!
                , kisi_ad: rs.string(forColumn: "kisi_ad")!
                , kisi_yas: Int( rs.string(forColumn: "kisi_yas"))!)

            liste.append(kisi)
        }
    }catch{
        print(error.localizedDescription)
    }

    db?.close()//Veritabanı bağlantısı kapanır.

    return liste
}
```

Insert : Veri kaydı

```
func kisiEkle(kisi_ad:String,kisi_yas:Int){  
    db?.open()  
  
    do {  
  
        try db!.executeUpdate("INSERT INTO kisiler (kisi_ad,kisi_yas) VALUES (?,?)", values: [kisi_ad,kisi_yas])  
  
    } catch {  
        print(error.localizedDescription)  
    }  
  
    db?.close()  
}
```

? işaretleri sırayla values dizisi içindeki verileri temsil eder.

values dizisi çeşitli türde verileri bir arada tutabilir.

Update : Güncelleme İşlemi

```
func kisiGuncelle(kisi_id:Int,kisi_ad:String,kisi_yas:Int){
    db?.open()

    do {
        try db!.executeUpdate("UPDATE kisiler SET kisi_ad = ?,kisi_yas=? WHERE kisi_id = ?", values: [kisi_ad,kisi_yas,kisi_id])

    } catch {
        print(error.localizedDescription)
    }

    db?.close()
}
```

Delete : Silme İşlemi

```
func kisiSil(kisi_id:Int){  
    db?.open()  
  
    do {  
  
        try db!.executeUpdate("DELETE FROM kisiler WHERE kisi_id = ?", values: [kisi_id])  
  
    } catch {  
        print(error.localizedDescription)  
    }  
  
    db?.close()  
}
```

Kayıt Kontrol

```
func kisiKontrol(kisi_ad:String) -> Int{
    var sonuc = 0
    db?.open()
    do {

        let rs = try db!.executeQuery("SELECT count(*) as sonuc FROM kisiler WHERE kisi_ad = ?", values: [kisi_ad])

        while rs.next() {
            sonuc = Int(rs.string(forColumn: "sonuc"))!
        }

    } catch {
        print(error.localizedDescription)
    }
    db?.close()

    return sonuc
}
```

Arama Yapma

```
func aramaYap(kisi_ad:String) -> [Kisiler]{

    var liste = [Kisiler]()

    db?.open()//Veritabanı bağlantısı açılır.

    do {
        let rs = try db!.executeQuery("SELECT * FROM kisiler WHERE kisi_ad like '%\($kisi_ad)%'", values: nil)

        while rs.next() {
            let kisi = Kisiler(kisi_id: Int(rs.string(forColumn: "kisi_id"))!
                , kisi_ad: rs.string(forColumn: "kisi_ad")!
                , kisi_yas: Int(rs.string(forColumn: "kisi_yas"))!)

            liste.append(kisi)
        }
    }catch{
        print(error.localizedDescription)
    }

    db?.close()//Veritabanı bağlantısı kapanır.

    return liste
}
```

Tek Bir Veri Getir

```
func kisiGetir(kisi_id:Int) -> Kisiler {  
  
    var kisi = Kisiler()  
  
    db?.open()//Veritabanı bağlantısı açılır.  
  
    do {  
        let rs = try db!.executeQuery("SELECT * FROM kisiler WHERE kisi_id = ?", values: [kisi_id])  
  
        while rs.next() {  
            kisi = Kisiler(kisi_id: Int(rs.string(forColumn: "kisi_id"))!  
                           , kisi_ad: rs.string(forColumn: "kisi_ad")!  
                           , kisi_yas: Int( rs.string(forColumn: "kisi_yas"))!)  
        }  
    }catch{  
        print(error.localizedDescription)  
    }  
  
    db?.close()//Veritabanı bağlantısı kapanır.  
  
    return kisi  
}
```

LIMIT : Sınırlı Veri Alma

```
func kisileriAl2() -> [Kisiler]{

    var liste = [Kisiler]()

    db?.open()//Veritabanı bağlantısı açılır.

    do {
        let rs = try db!.executeQuery("SELECT * FROM kisiler LIMIT 2", values: nil)

        while rs.next() {
            let kisi = Kisiler(kisi_id: Int(rs.string(forColumn: "kisi_id"))!
                , kisi_ad: rs.string(forColumn: "kisi_ad")!
                , kisi_yas: Int( rs.string(forColumn: "kisi_yas"))!)

            liste.append(kisi)
        }
    }catch{
        print(error.localizedDescription)
    }

    db?.close()//Veritabanı bağlantısı kapanır.

    return liste
}
```

Rasgele 2 Veri Getirme

```
func rasgele2Kisi() -> [Kisiler]{

    var liste = [Kisiler]()

    db?.open()//Veritabanı bağlantısı açılır.

    do {
        let rs = try db!.executeQuery("SELECT * FROM kisiler ORDER BY RANDOM() LIMIT 2", values: nil)

        while rs.next() {
            let kisi = Kisiler(kisi_id: Int(rs.string(forColumn: "kisi_id"))!
                                 , kisi_ad: rs.string(forColumn: "kisi_ad")!
                                 , kisi_yas: Int( rs.string(forColumn: "kisi_yas"))!)

            liste.append(kisi)
        }
    }catch{
        print(error.localizedDescription)
    }

    db?.close()//Veritabanı bağlantısı kapanır.

    return liste
}
```

Teşekkürler...



kasim-adalan



kasimadalan@gmail.com



kasimadalan