ВТУ "Св. Св. Кирил и Методий"
Факултет "Математика и Информатика"

КУРСОВА РАБОТА
ПО ДИСЦИПЛИНАТА
Мобилни приложения за iOS
На тема:
ОРГАНИЗАТОР НА АНИМЕ И МАНГА АКТИВНОСТ

Изготвил:
Пламенна Викторова Петрова
Специалност:
Софтуерно инженерство
Факултетен № 1909011132

Проверил:
гл. ас. д-р
Георги Светлинов
Шипковенски

Велико Търново
2021

Изисквания към разработване на приложение

Да се разработи приложение от вида:

1. Приложение съдържащо текстови полета.
2. Приложение съдържащо табличен изглед.
3. Приложение съдържащо връзка към Core Data.
4. Приложение съдържащо WebView.
5. Приложение съдържащо връзка към приложен програмен интерфейс (API).
6. Приложение съдържащо работа със сензори или използване на услуги за местоположение.

Разработеното приложение задължително да съдържа имплементиране на навигация с връзка към друг view controller.

При невъзможност за разработка на приложение, да се направи проучване за новата работна рамка SwiftUI на Apple, за изграждане на потребителски интерфейси за iOS, tvOS, macOS и watchOS.

Обемът на документацията е необходимо да бъде не по-малък от 4 000 символа, като размерът на шрифта и разредката се определят по избор от студента.

Съдържание

1. Цел и изпълнение на разработката

Цел на разработката е да се създаде SwiftUI базирано приложение, което използва Firebase Authentication с Email/Password Sign-in Provider за съхранение на потребители както и вградената Core Data база данни за запис на Entity обекти. Активността на всеки един потребител е разграничена. Той може да добавя нови обекти от тип Anime, Manga, TVShow или Movie и да вижда списък от направените от него записи заедно с най-важната информация, която ги репрезентира. Редактирането се постига динамично във View-то за детайли за всеки обект, а при навигация обратно към списъка промените биват отразени. За изтриването пък се избира конкретен обект от списъка, върху който да бъде извършена операцията. Съществува и изглед, който извежда последните три обекта от съответния тип, сортирани по датата им на създаване. Общи характеристики на обектите са : id, име, епизоди или глави, ако има такива, рейтинг, кратко ревю, статут (завършен, четен/гледан в момента, планиран и други), id на потребителя и снимка, като за последната е писан къстъм ImagePicker, чрез който се достъпва библиотеката за снимки на устройство и може да се посочи дадено изображение. В приложението са създадени View-та, в които се fetch-ват резултати от Open Source API-то Jikan, което представлява неофициалното API на известния сайт за японски аниме и манга продукции MyAnimeList. Извличат се данни за най-добрите 50 анимета и манга спрямо дадения рейтинг. Освен това App-ът следи дали потребителят е логнат или не – ако последното действие от негова страна, преди да затвори приложението, е да се log out-не, при следващото му влизане той ще бъде пренасочен към логин View-то и обратно, когато логването е уловено, на потребителя ще се покаже главният изглед.

2. Основни етапи на разработката

2.1. Връзка към Firebase Authentication

За да се добавят dependency-тата на Firebase първо трябва да се инсталират cocoapods за проекта с командата :

sudo gem install cocoapods

После Pods библиотеките се инициализират с :

pod init

Отваря се Podfile-а -> open Podfile, където се описват dependency-тата, в случая небходими са :

pod ‘Firebase/Core’

pod ‘Firebase/Auth’

С pod install те биват инсталирани. Отваря се xcode workspace-а на приложението, защото в xcode project новоинсталираните dependency-та не могат да бъдат достъпени. Междувременно се създава и нов проект във Firebase конзолата, за Authentication се избира Email/Password и в Project Overview app-ът се регистрира за iOS. Изтегля се Google-Service-info.plist файла, който трябва да се принесе и в Xcode. С това проектът бива свързан към услугите на Firebase.

2.2. Създаване на изгледи за регистрация и логин и имплементиране на логика за аутентификация

В ContentView е разписан клас, който наследява ObservableObject и съдържа функции са регистрация, логин и логаут на потребителите, служейки си с предоставените възможности на Firebase. Една @Published променлива се променя автоматично спрямо това дали auth функционалността засича потребители.

```
class AppViewModel : ObservableObject {

   let auth = Auth.auth()

   @Published var signedIn = false

   var isSignedIn: Bool {
      return auth.currentUser != nil
   }

   func signIn(email: String, password: String) {
      auth.signIn(withEmail: email, password: password) { [weak self] result,
error in
         guard result != nil, error == nil else {
            return
```

```
            }

            DispatchQueue.main.async {
                // Success
                self?.signedIn = true
            }
        }
    }

    func signUp(email: String, password: String) {
        auth.createUser(withEmail: email, password: password) { [weak self]
result, error in
            guard result != nil, error == nil else {
                return
            }

            DispatchQueue.main.async {
                // Success
                self?.signedIn = true
            }
        }
    }

    func signOut() {
        try? auth.signOut()

        self.signedIn = false
    }

}
```

По-нататък в SignIn и SignUp View-тата потребителят подава на текстово и secure поле имейла и паролата си, а в в action-ните на бутоните се извикват функциите за регистрация и логин от view модела и се извършва проста валидация за резултата. В SignIn View-то има навигационен линк към SignUp View-то.

```
struct SignInView: View {
    @State var email = ""
    @State var password = ""
```

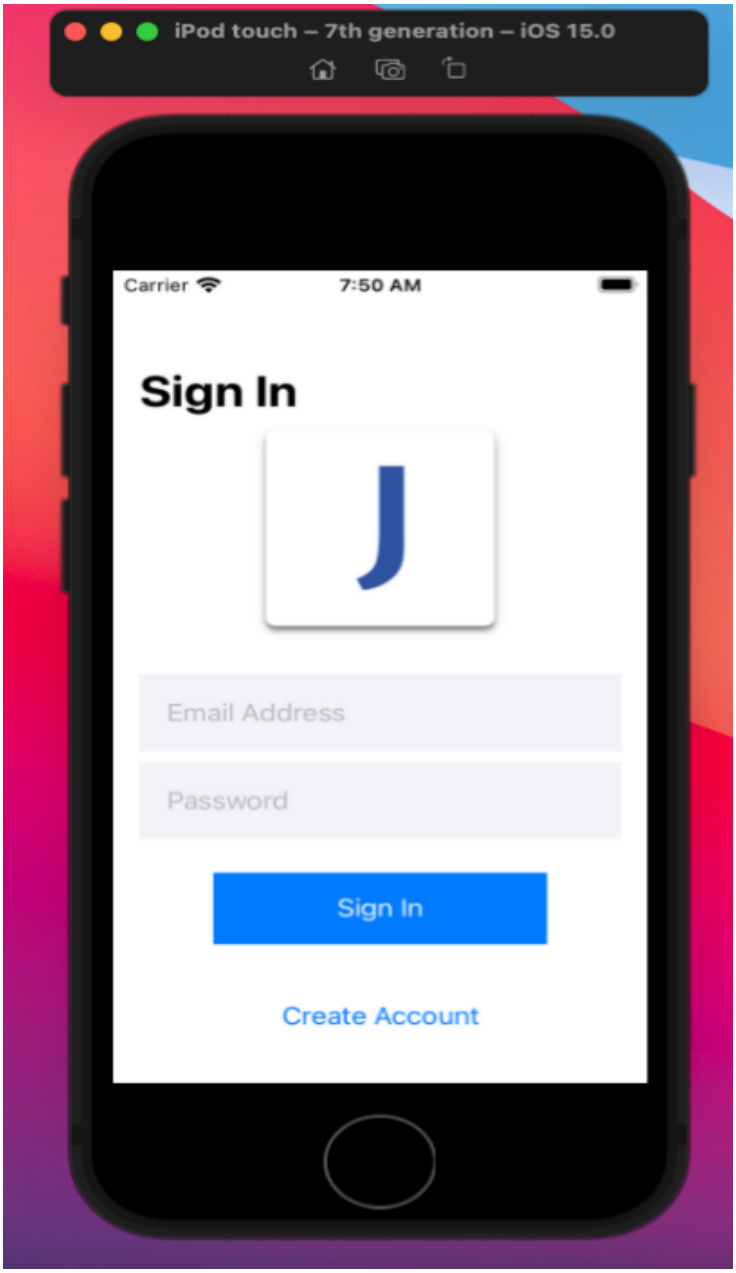
```
@EnvironmentObject var viewModel: AppViewModel

var body: some View {
    VStack {
        Image("logo")
            .resizable()
            .scaledToFit()
            .frame(width: 150, height: 150)

        VStack {
            TextField("Email Address", text: $email)
                .disableAutocorrection(true)
                .autocapitalization(.none)
                .padding()
                .background(Color(.secondarySystemBackground))

            SecureField("Password", text: $password)
                .disableAutocorrection(true)
                .autocapitalization(.none)
                .padding()
                .background(Color(.secondarySystemBackground))

            Button(action: {

                guard !email.isEmpty, !password.isEmpty else {
                    return
                }

                viewModel.signIn(email: email, password: password)

            }, label: {
                Text("Sign In")
                    .foregroundColor(Color.white)
                    .frame(width: 200, height: 50)
                    .cornerRadius(8)
                    .background(Color.blue)
            })
            .padding()
```

```
                NavigationLink("Create Account", destination: SignUpView())
                    .padding()
            }
            .padding()

            Spacer()
        }
        .navigationTitle("Sign In")
    }
}
```

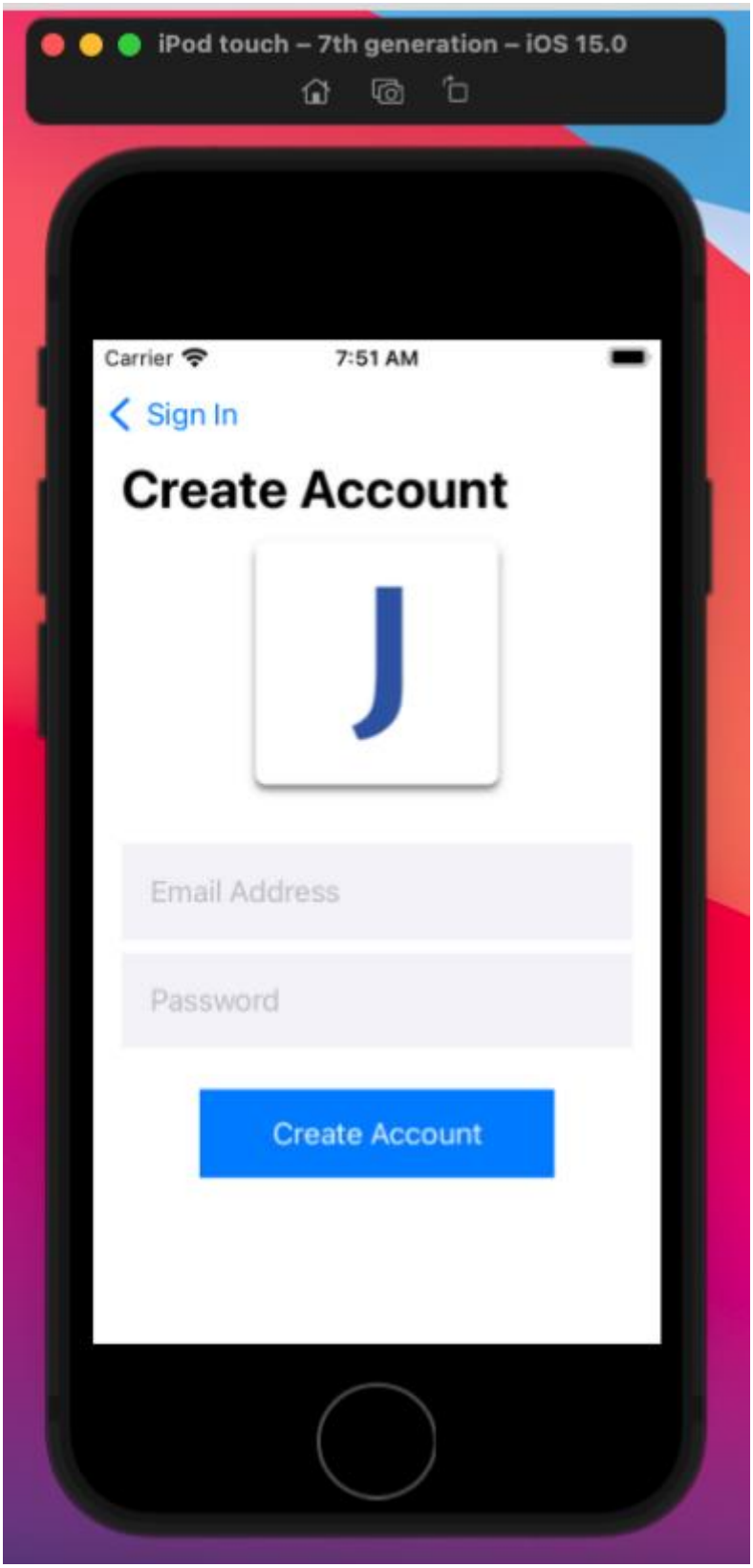
В main класа на приложението се декларира променилива от тип @UIApplicationDelegateAdaptor, сочеща AppDelegate класа, където се конфигурира FirebaseApp. Отделно на ContentView се закача инстанция на AppViewModel като environmentObject. Тези действия осъществяват изпълнението на Firebase функциите.

```
@UIApplicationDelegateAdaptor(AppDelegate.self) var appDelegate

class AppDelegate: NSObject, UIApplicationDelegate {
    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey : Any]? = nil) -> Bool {

        FirebaseApp.configure()

        return true
    }
}

var body: some Scene {
        WindowGroup {
            let viewModel = AppViewModel()
            ContentView()
                .environmentObject(viewModel)
        }
    }
```

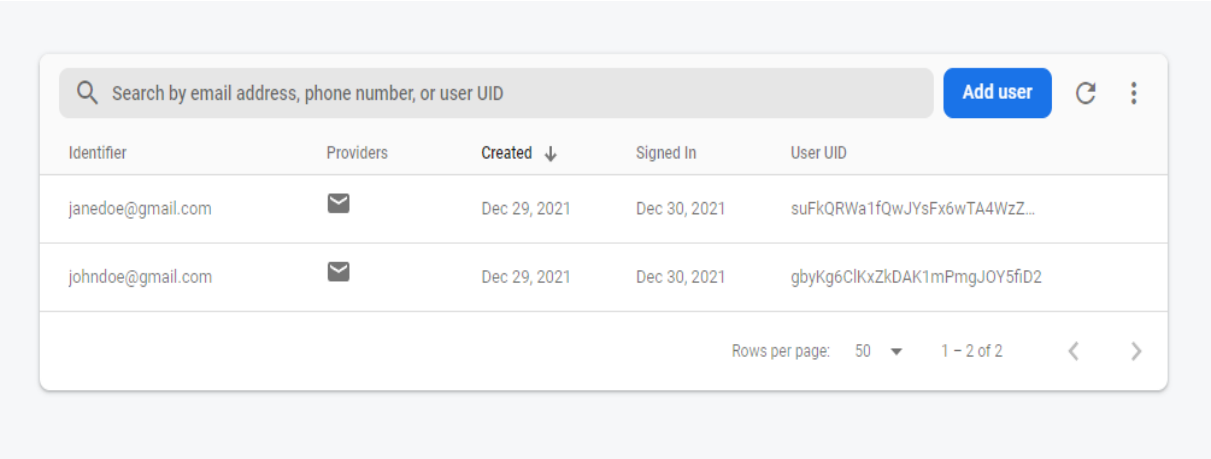
SignInView :


iPod touch – 7th generation – iOS 15.0
Carrier
7:50 AM
Sign In
J
Email Address
Password
Sign In
Create Account

SignUpView :

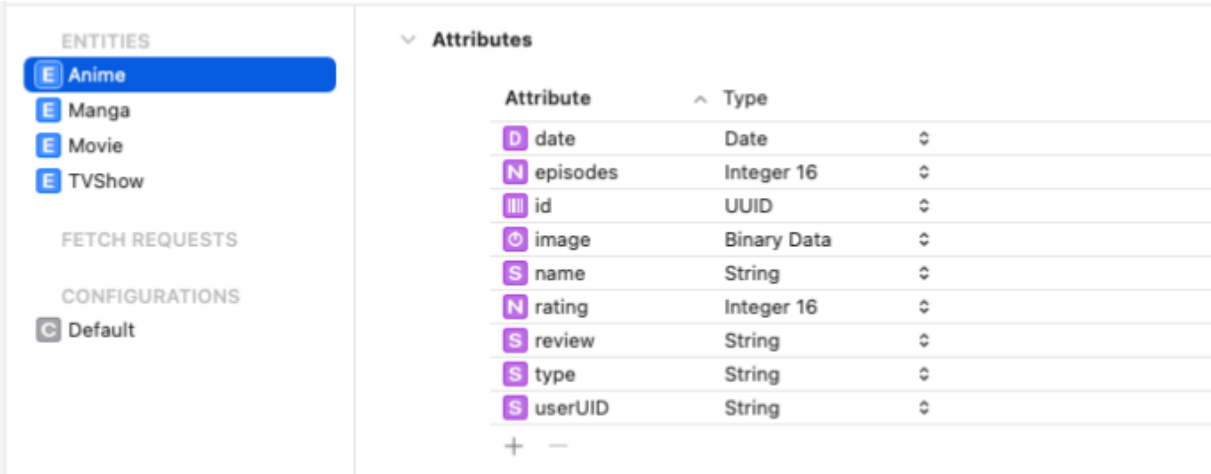


Примерни регистрирани потребители във Firebase :



2.3. Работа с CoreData

В CoreData са добавени Entity-тата Anime, Manga, Movie и TVShow, а за атрибутите им са посочени типове на данните. Така id-та са от тип UUID – уникален идентификатор, имената са стрингове, епизодите – integer-и, датите – Date, а снимките – BinaryData.



2.4. Създаване на Add, Details и List View-та за обектите

1) AddView-та

Вземаме за пример аниметата :

```
import SwiftUI
import FirebaseAuth

struct AddAnimeView: View {

    @Environment(\.managedObjectContext) var moc
```

```
    @Environment(\.presentationMode) var presentationMode

    @State private var name = ""
    @State private var episodes = 0
    @State private var rating = 0
    @State private var review = ""
    @State private var type = ""

    @State private var image : Data? = .init(count: 0)

    @State private var showImage = false

    let types = ["Watching", "On hold", "Dropped", "Completed", "ReWatching",
"Plan to Watch"]

    let currentUserUID : String = (Auth.auth().currentUser?.uid)!

    var body: some View {

        let jikanImage = UIImage(named: "logo")
        let jikanImagePngData = jikanImage?.pngData()

        NavigationView{
            Form {
                TextField("Anime name", text: $name)
                    .disableAutocorrection(true)
                Section {
                    HStack {
                        Text("Upload image")
                        Spacer()
                        if self.image?.count != 0 {
                            Button(action: {
                                self.showImage.toggle()
                            }) {
                                Image(uiImage: UIImage(data: self.image!)!)
                                    .renderingMode(.original)
                                    .resizable()
                                    .frame(width: 75, height: 75, alignment: .leading)
                            }
```

```
        } else {
            Button(action: {
                self.showImage.toggle()
            }) {
                Image("logo")
                    .resizable()
                    .scaledToFit()
                    .frame(width: 75, height: 75)
            }
        }
    }
}
Section {
    Picker("Select", selection: $type) {
        ForEach(types, id: \.self) {
            Text($0)
        }
    }
}
Section {
    HStack {
        Text("Episodes")
            .font(.headline)
        Spacer()
        Picker("Episodes", selection: $episodes) {
            ForEach(0..<2000){
                Text("\($0)")
            }
        }
        .pickerStyle(WheelPickerStyle())
        .frame(width: 100, height: 40, alignment: .center)
        .clipped()
        .compositingGroup()
    }
}
Section {
    Picker("Rating", selection: $rating) {
        ForEach(0..<11) {i in
            HStack {
                Image(systemName: "star.fill")
```

```
                        .foregroundColor(.yellow)
                    Text(" \(i)")
                  }
                }
              }
            }
            Section {
              Text("Review")
              TextEditor(text: $review)
                .frame(height: 170)
                .disableAutocorrection(true)
            }
          }
          .navigationBarItems(trailing:
           Button("Save") {
            let newAnime = Anime(context: self.moc)
            newAnime.name = self.name
            newAnime.image = self.image ?? jikanImagePngData
            newAnime.type = self.type
            newAnime.episodes = Int16(self.episodes)
            newAnime.rating = Int16(self.rating)
            newAnime.review = self.review
            newAnime.date = Date()
            newAnime.userUID = currentUserUID

            try? self.moc.save()

            self.presentationMode.wrappedValue.dismiss()
          })
          .navigationBarTitleDisplayMode(.inline)
          .sheet(isPresented: self.$showImage, content: {
            ImagePicker(show: self.$showImage, image: self.$image)
          })
      }
    }
}

struct AddAnimeView_Previews: PreviewProvider {
    static var previews: some View {
        AddAnimeView()
```

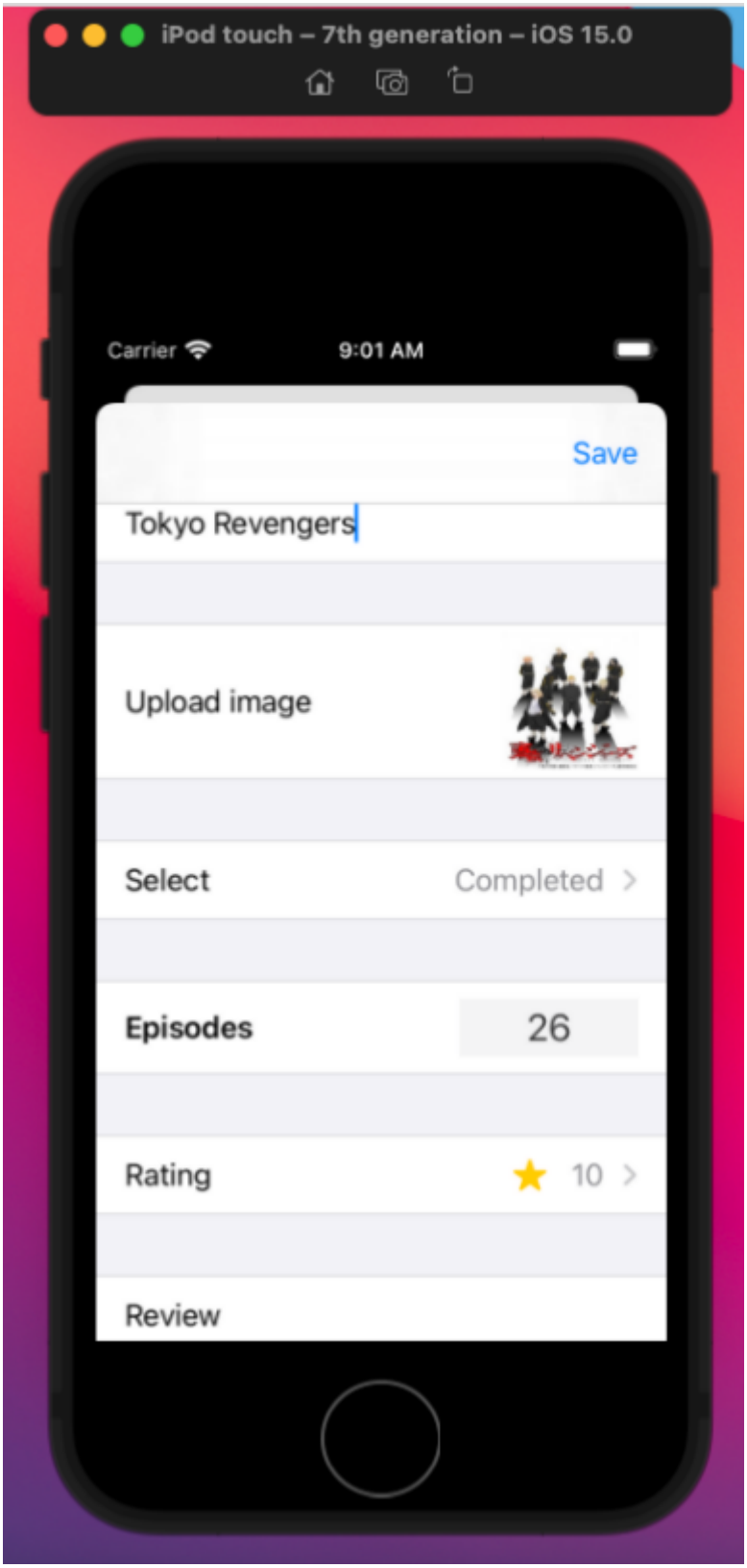
```
    }
}
```

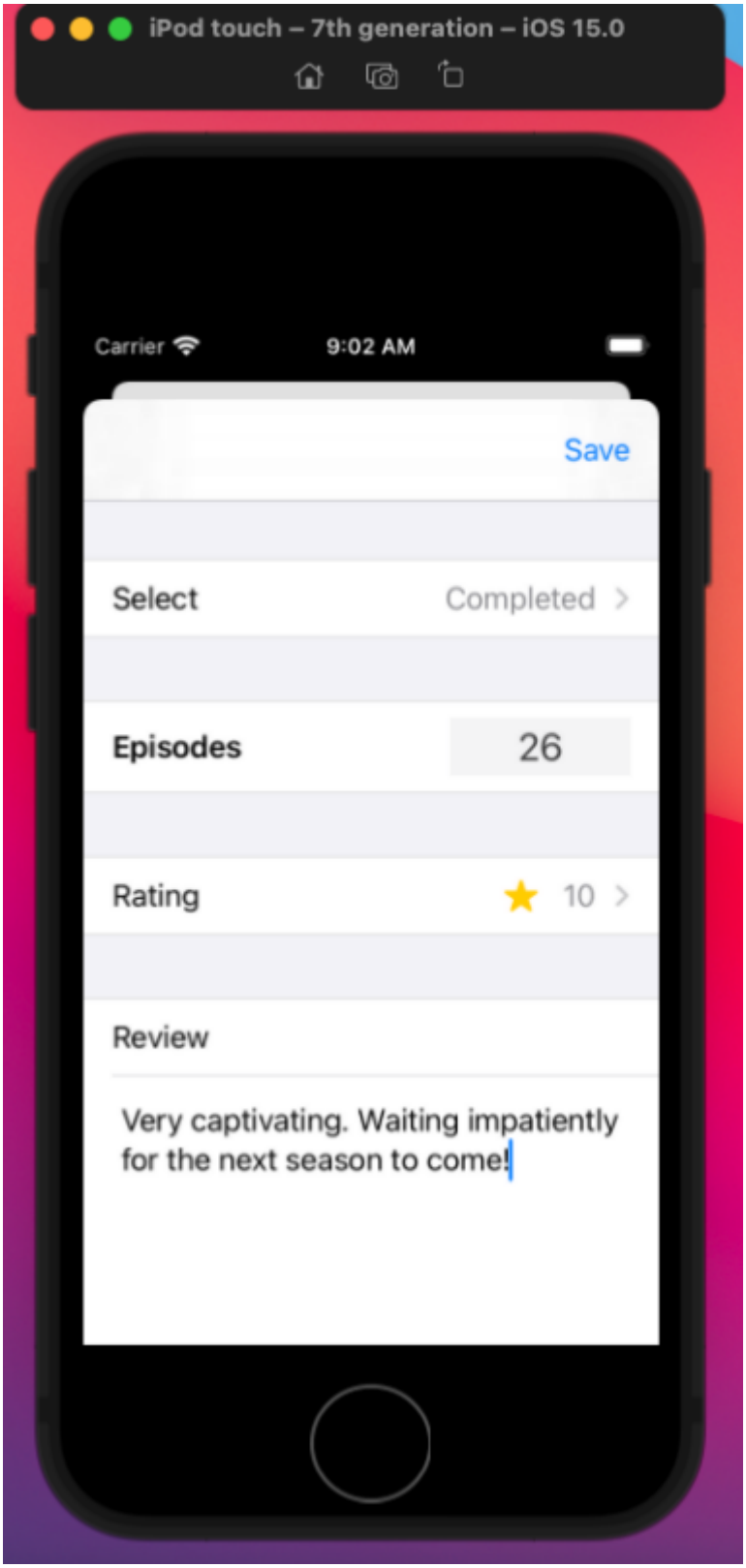
В структурата се декларира managedObjectContext променлива за менажиране на Entity обектите от CoreData. @State променливите поддържат модификатори както за четене така и за писане. Image променливата е от опционален Data тип, защото ако потребителят реши да не избере снимка при запис на нов обект, да се изпълни алтернативата за добавяне на снимка по подразбиране от Assets хранилището. Статутите на гледане се съхраняват в хардкоднат масив, а чрез showImage променливата се следи дали потребителят е задействал ImagePicker-а. Текстово поле пази референция към наименованието на анимето, в TextEditor се пише ревюто, а с обикновени Picker-и се посочват статут, рейтинг и брой на епизодите. Елементите на View-то са организирани във форма и са разделени в секции. На някои места те са nest-нати един в друг. Снимките се подменят, като се запазва оригиналния им вид, въпреки че се преоразмеряват. В координиращ клас в ImagePicker структурата е написана и логиката за компресиране и отразяване на същото качество при upload. За дата се регистрира моментният timestamp.

```
@Binding var show : Bool
 @Binding var image : Data?

    func makeCoordinator() -> ImagePicker.Coordinator {
        return ImagePicker.Coordinator(firstChild: self)
    }

    func makeUIViewController(context:
UIViewControllerRepresentableContext<ImagePicker>) ->
UIImagePickerController {
        let picker = UIImagePickerController()
```

```
        picker.sourceType = .photoLibrary
        picker.delegate = context.coordinator
        return picker
    }


class Coordinator : NSObject, UIImagePickerControllerDelegate,
UINavigationControllerDelegate {

        var child : ImagePicker

        init (firstChild : ImagePicker) {
            child = firstChild
        }

        func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {
            self.child.show.toggle()
        }

        func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {

            let image = info[.originalImage]as! UIImage

            let data = image.jpegData(compressionQuality: 0.45)

            self.child.image = data!
            self.child.show.toggle()
        }
    }
```

При натискане на бутона Save се създава обект от тип Anime съобразно избраните от потребителя стойности за атрибутите, userUID-то се взема от detector-а на Firebase Auth.


iPod touch – 7th generation – iOS 15.0
Carrier
9:01 AM
Save
Tokyo Revengers
Upload image
Select
Completed
Episodes
26
Rating
10
Review


iPod touch – 7th generation – iOS 15.0
Carrier
9:02 AM
Save
Select
Completed
Episodes
26
Rating
10
Review
Very captivating. Waiting impatiently for the next season to come!

2) ListView-та

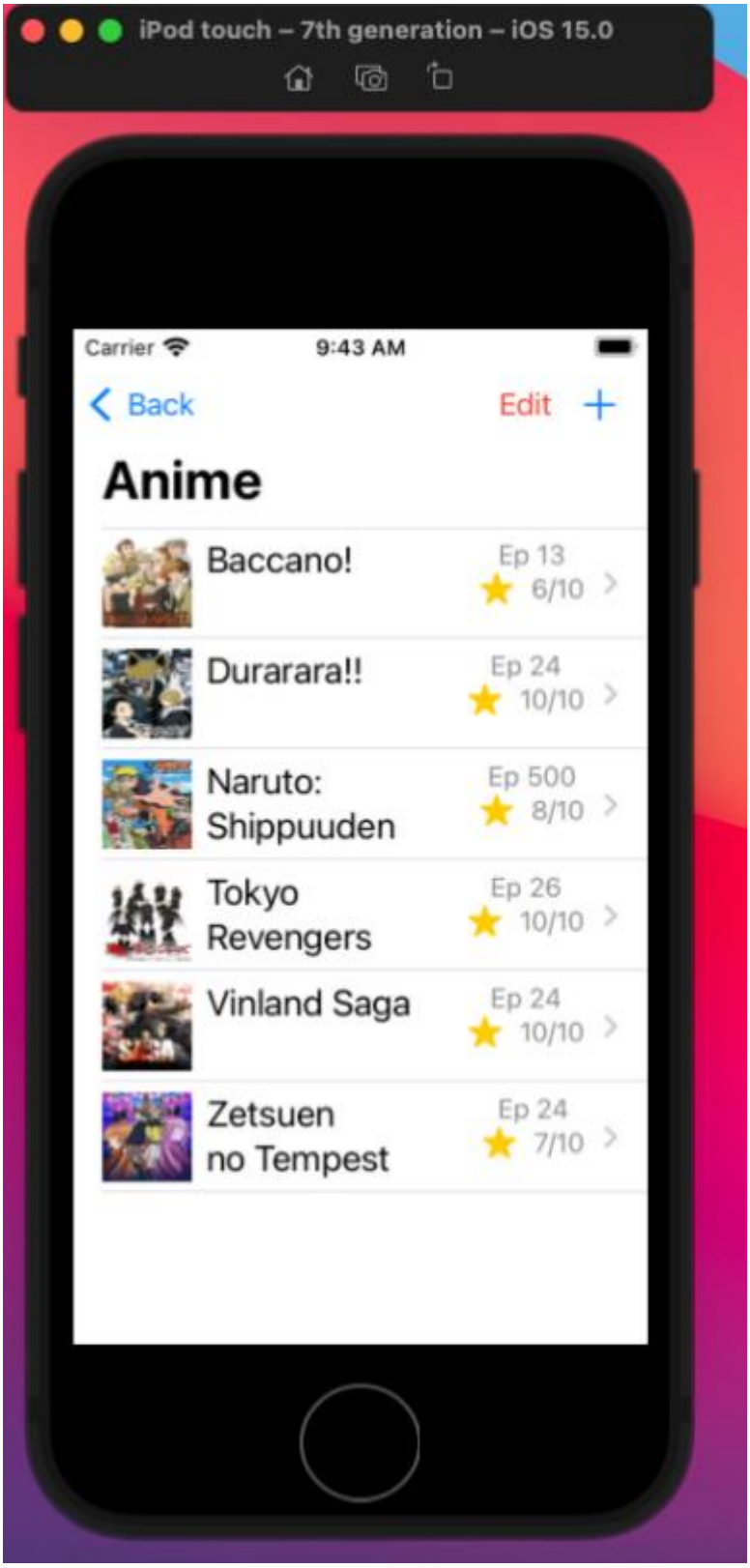
Представено е AnimeListView-то :

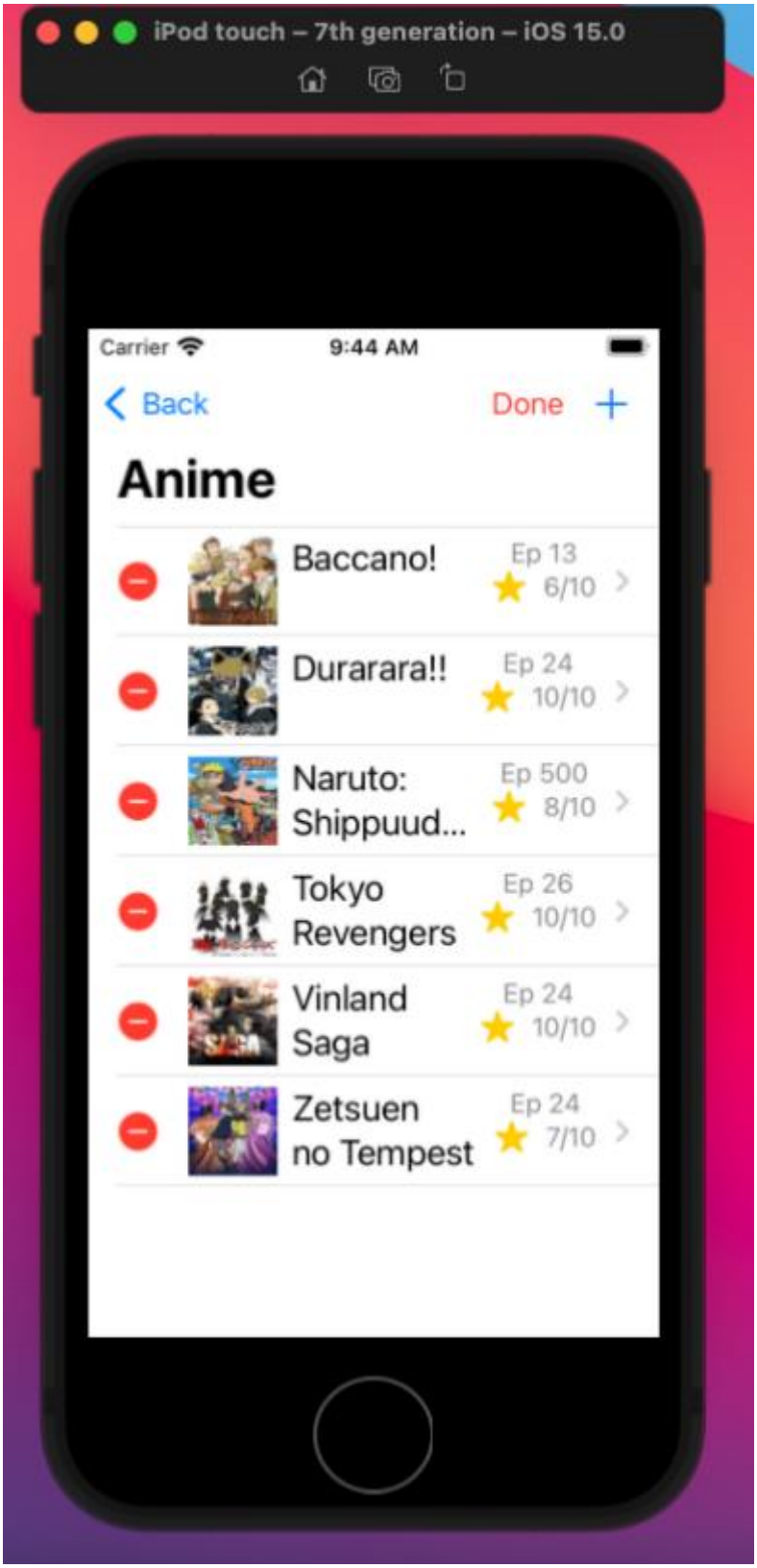
```
import SwiftUI
import Firebase

struct AnimeListView: View {

   @Environment(\.managedObjectContext) var moc

   @FetchRequest(entity: Anime.entity(), sortDescriptors: [
      NSSortDescriptor(keyPath: \Anime.name, ascending: true),
      NSSortDescriptor(keyPath: \Anime.episodes, ascending: true)
   ]) var anime: FetchedResults<Anime>

   @State private var showingAddScreen = false

   var body: some View {

      let jikanImage = UIImage(named: "logo")
      let jikanImagePngData = jikanImage?.pngData()

      List{
         ForEach(anime,id: \.self){ anime in
            if (Auth.auth().currentUser?.uid == anime.userUID) {
               NavigationLink(destination:
                  AnimeDetailsView(anime: anime)) {
                  VStack(alignment: .leading) {
                     HStack(alignment: .top) {
                        Image(uiImage: UIImage(data: anime.image ??
jikanImagePngData!)!)
                           .resizable()
```

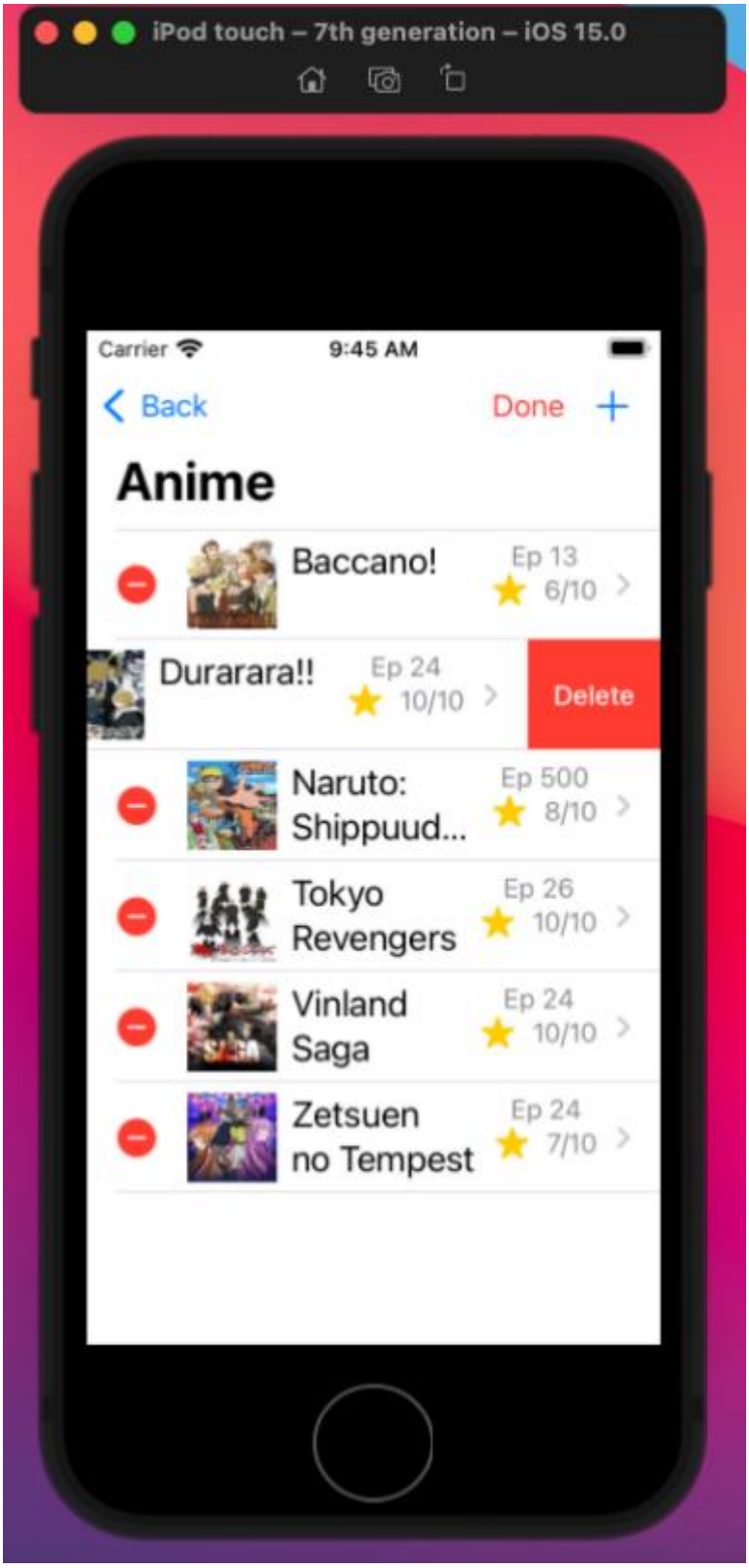
```
                        .frame(width: 50, alignment: .center)
                    Text(anime.name ?? "Unknown")
                        .font(.title2)
                    Spacer()
                    VStack{
                        Text("Ep \(anime.episodes)")
                            .font(.subheadline)
                            .multilineTextAlignment(.trailing)
                            .foregroundColor(.secondary)
                        HStack{
                            Image(systemName: "star.fill").foregroundColor(.yellow)
                            Text("\(anime.rating)/10")
                                .font(.subheadline)
                                .foregroundColor(.secondary)
                        }
                    }
                }
            }
            .frame(height: 50)
        }          }
    }
    .onDelete(perform: removeAnime)
  }
  .navigationBarTitle("Anime")
  .listStyle(PlainListStyle())
  .navigationBarItems(trailing: HStack{
    EditButton().accentColor(.red); Button(action: {
      self.showingAddScreen.toggle()
    }) {
      Image(systemName: "plus")
    }
  })
  .sheet(isPresented: $showingAddScreen) {
    AddAnimeView().environment(\.managedObjectContext, self.moc)
  }
}

func removeAnime(at offsets: IndexSet) {
  for offset in offsets{
    let singleAnime = anime[offset]
```

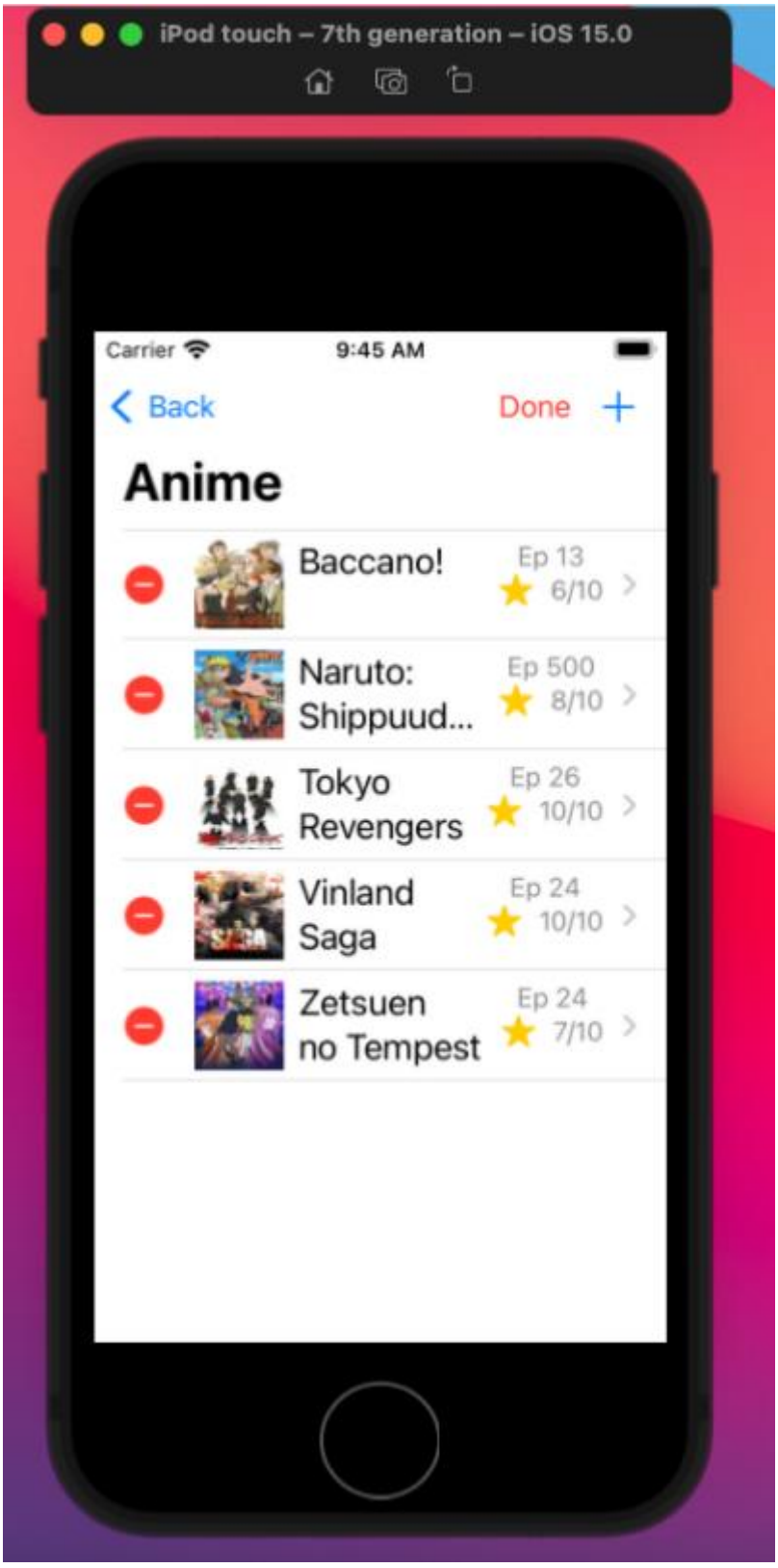
```
            moc.delete(singleAnime)
        }
        try? moc.save()
    }
}

struct AnimeListView_Previews: PreviewProvider {
    static var previews: some View {
        AnimeListView()
    }
}
```

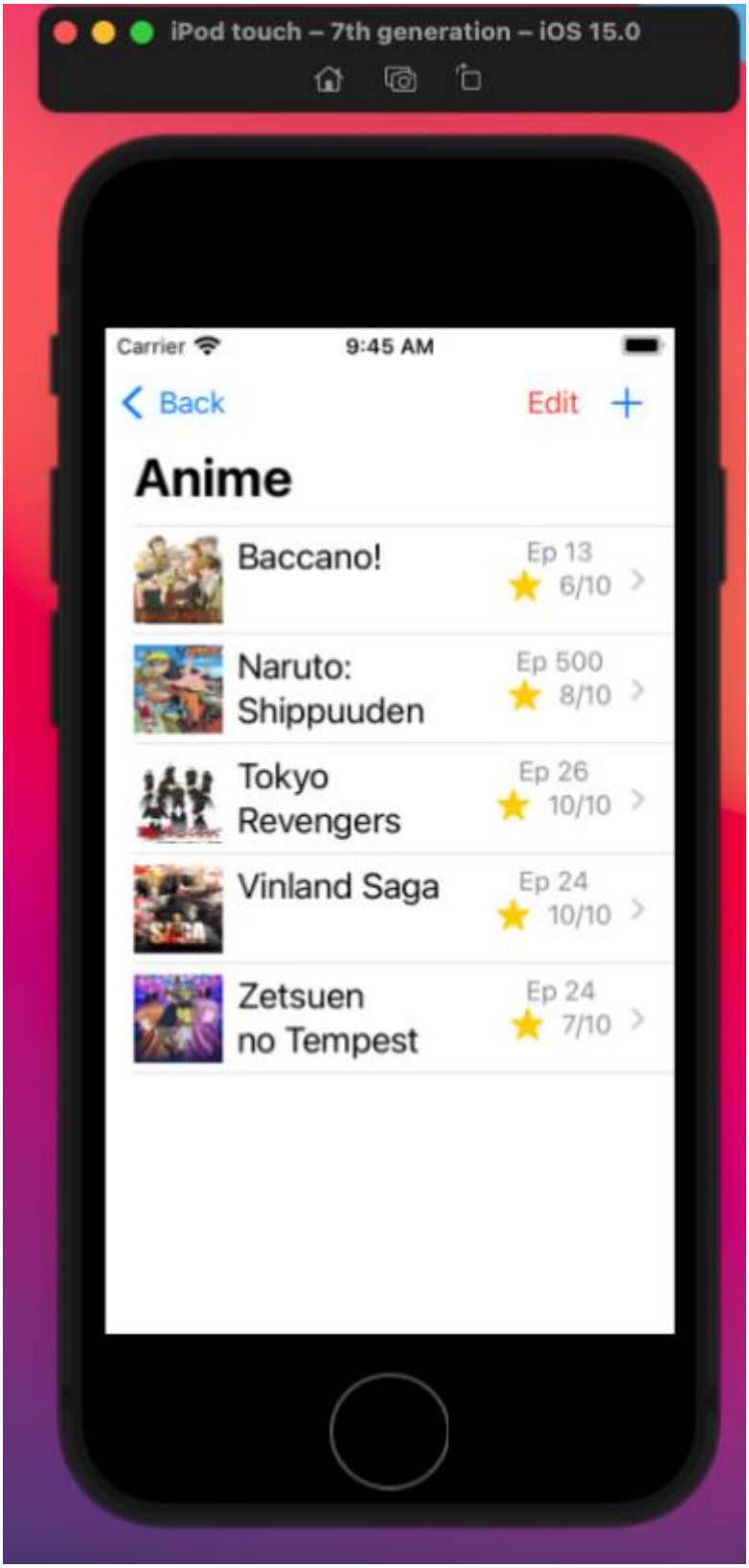
С @FetchRequest се означават entity обектите от тип Anime, те се сортират по име и епизоди в намаляващ ред чрез NSSortDescriptor-и. Fetch-натите резултати се записват в променливата anime. Във ForEach-а на List-а аниметата се филтрират спрямо това дали техния userUID съвпада с UID-то на текущия потребител. Всеки елемент на List-а представлява навигационен линк към прилежащия му AnimeDetailsView. Показва се следната информация : снимка, наименование, епизоди, рейтинг. Кликването на Edit бутона в горния край привежда в режим, при който ако се натисне червения минус до даден запис, той се изтрива от базата. Селекцията кой обект да бъде премахнат се случва във функцията removeAnime, която се изпълнява при onDelete. При натискане на плюса потребителят отваря AddAnimeView-то.


iPod touch – 7th generation – iOS 15.0
Carrier
9:43 AM
Back
Edit
Anime
Baccano!
Ep 13
6/10
Durarara!!
Ep 24
10/10
Naruto:
Shippuuden
Ep 500
8/10
Tokyo
Revengers
Ep 26
10/10
Vinland Saga
Ep 24
10/10
Zetsuen
no Tempest
Ep 24
7/10


iPod touch – 7th generation – iOS 15.0
Carrier
9:44 AM
Back
Done
Anime
Baccano!
Ep 13
6/10
Durarara!!
Ep 24
10/10
Naruto: Shippuud...
Ep 500
8/10
Tokyo Revengers
Ep 26
10/10
Vinland Saga
Ep 24
10/10
Zetsuen no Tempest
Ep 24
7/10


iPod touch – 7th generation – iOS 15.0
Carrier
9:45 AM
Back
Done
Anime
Baccano!
Ep 13
6/10
Durarara!!
Ep 24
10/10
Delete
Naruto: Shippuud...
Ep 500
8/10
Tokyo Revengers
Ep 26
10/10
Vinland Saga
Ep 24
10/10
Zetsuen no Tempest
Ep 24
7/10


iPod touch – 7th generation – iOS 15.0
Carrier
9:45 AM
Back
Done
Anime
Baccano!
Ep 13
6/10
Naruto: Shippuud...
Ep 500
8/10
Tokyo Revengers
Ep 26
10/10
Vinland Saga
Ep 24
10/10
Zetsuen no Tempest
Ep 24
7/10


iPod touch – 7th generation – iOS 15.0
Carrier
9:45 AM
Back
Edit
Anime
Baccano!
Ep 13
6/10
Naruto:
Shippuuden
Ep 500
8/10
Tokyo
Revengers
Ep 26
10/10
Vinland Saga
Ep 24
10/10
Zetsuen
no Tempest
Ep 24
7/10

3) DetailsView-та

За аниметата :

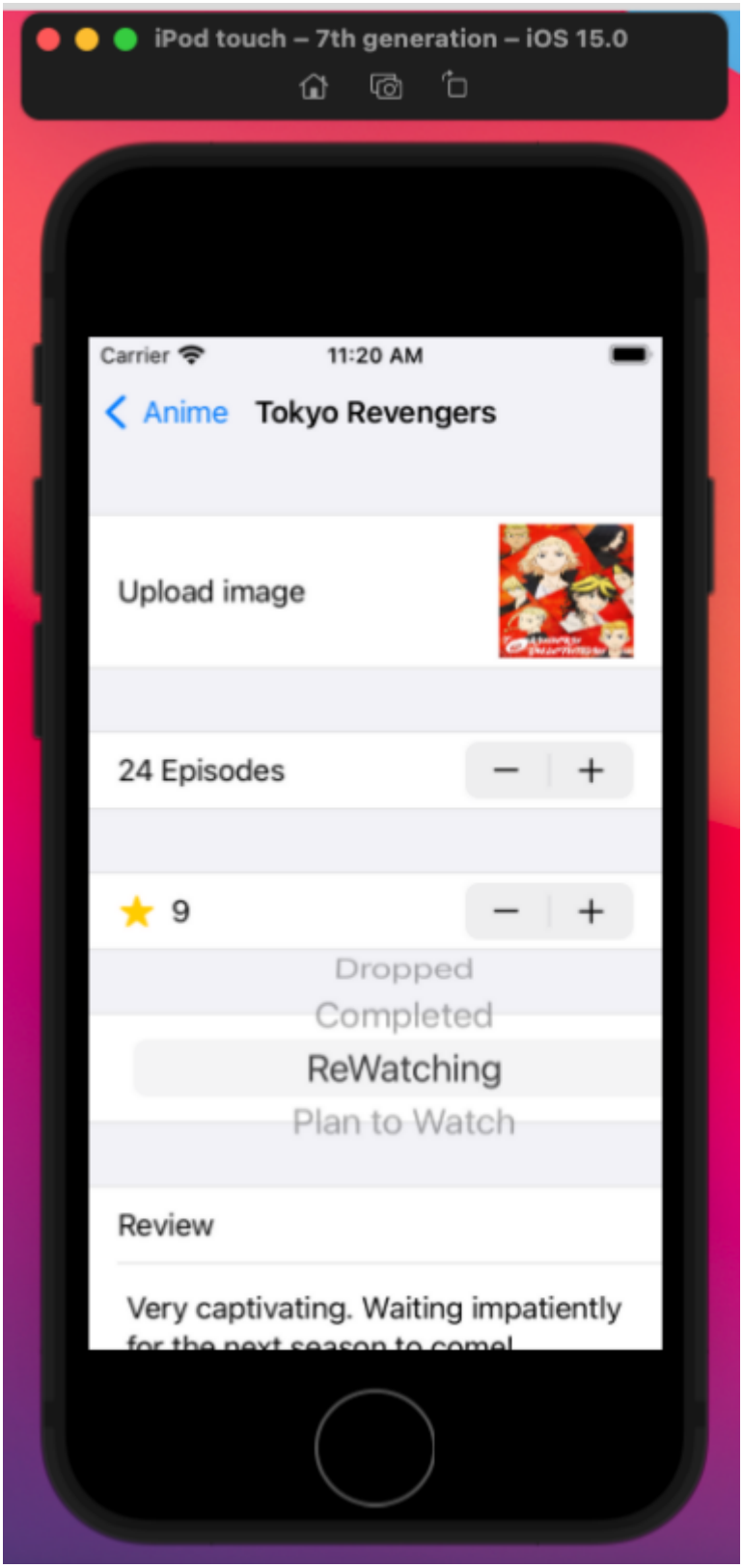
```
import SwiftUI
import CoreData

struct AnimeDetailsView: View {

    @ObservedObject var anime: Anime

    @Environment(\.managedObjectContext) var moc

    @Environment(\.presentationMode) var presentationMode

    @State private var episodes: Int16 = 0
    @State private var rating: Int16 = 0
    @State private var review = ""
    @State private var type = ""

    @State private var image : Data? = .init(count: 0)

    @State private var showImage = false

    let types = ["Watching", "On hold", "Dropped", "Completed", "ReWatching",
"Plan to Watch"]

    var body: some View {

        let jikanImage = UIImage(named: "logo")
        let jikanImagePngData = jikanImage?.pngData()
```

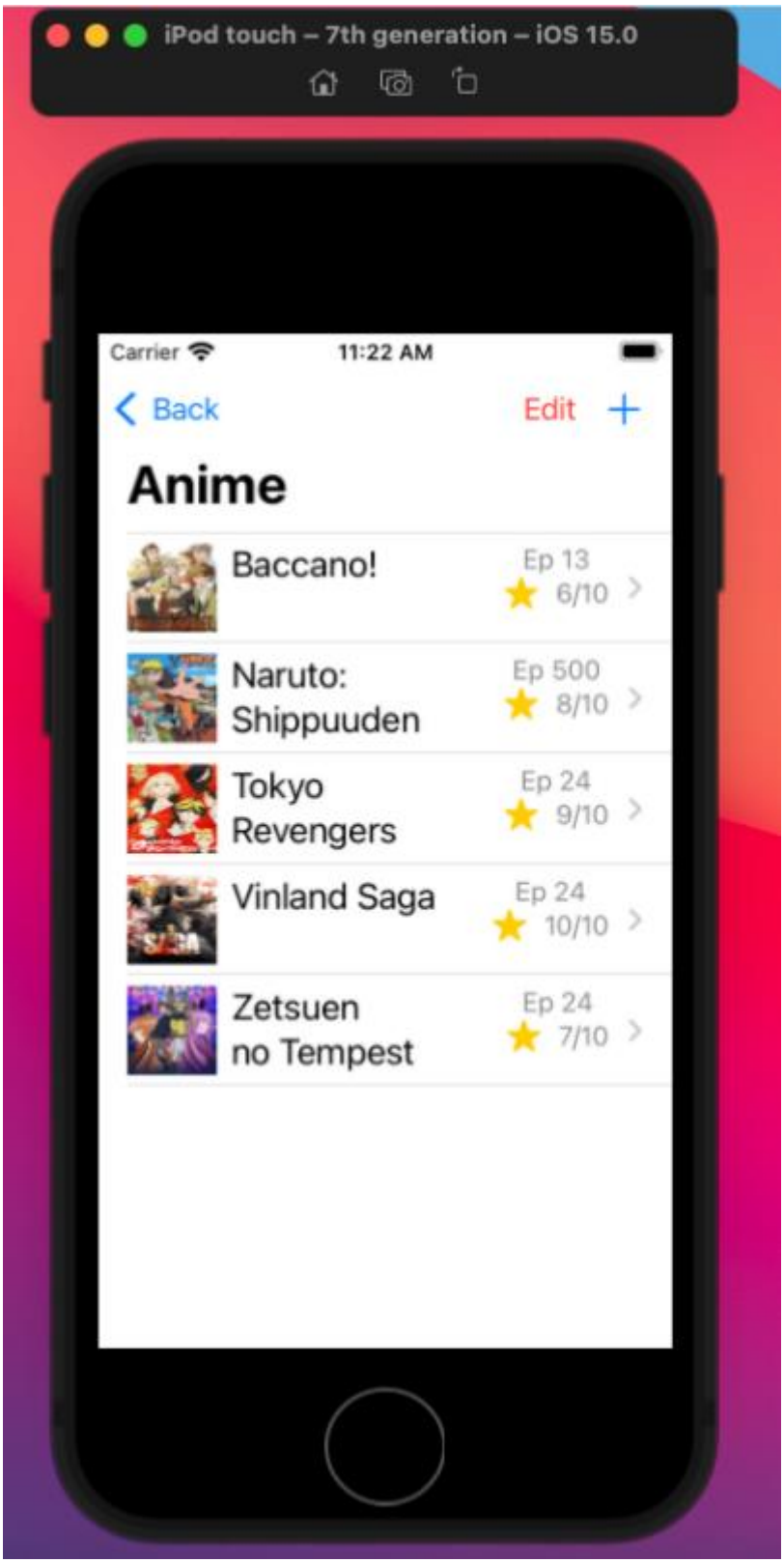
```
VStack{
   Form{
      Section {
         HStack {
            Text("Upload image")
            Spacer()
            if self.image?.count != 0 {
               Button(action: {
                  self.showImage.toggle()
               }) {
                  Image(uiImage: UIImage(data: self.image!)!)
                     .renderingMode(.original)
                     .resizable()
                     .frame(width: 75, height: 75, alignment: .leading)
               }
            } else {
               Button(action: {
                  self.showImage.toggle()
               }) {
                  Image("logo")
                     .resizable()
                     .scaledToFit()
                     .frame(width: 75, height: 75)
               }
            }
         }
      }
      Stepper(value: $episodes, step: 1) {
         Text("\(episodes) Episodes")
      }
      Section{
         Stepper(value: $rating, in: 0...10, step: 1) {
            HStack{
               Image(systemName: "star.fill")
                  .foregroundColor(.yellow)
               Text("\(rating)")
            }
         }
      }
```

```
            Picker("Select", selection: $type) {
                ForEach(types, id: \.self) {
                    Text($0)
                }
            }
            .pickerStyle(WheelPickerStyle())
            .frame(height: 50, alignment: .center)
            Section{
                Text("Review")
                TextEditor(text: $review)
                    .frame(height: 170)
                    .disableAutocorrection(true)
            }
        }
        .sheet(isPresented: self.$showImage, content: {
            ImagePicker(show: self.$showImage, image: self.$image)
        })
    }
    .onAppear{
        self.episodes = self.anime.episodes
        self.rating = self.anime.rating
        self.type = self.anime.type ?? "-"
        self.review = self.anime.review ?? "-"
        self.image = self.anime.image ?? jikanImagePngData!
    }
    .onDisappear(perform: saveChanges)
    .navigationBarTitle("\(anime.name ?? "-")", displayMode: .inline)
}

func saveChanges() {
    anime.episodes = episodes
    anime.rating = rating
    anime.review = review
    anime.type = type
    anime.image = image
    try? self.moc.save()
    DispatchQueue.main.asyncAfter(deadline: .now() + 0.5){
        self.presentationMode.wrappedValue.dismiss()
    }
}
```

```
}

struct AnimeDetailsView_Previews: PreviewProvider {
   static let moc = NSManagedObjectContext(concurrencyType:
.mainQueueConcurrencyType)

   static var previews: some View {
      let anime = Anime(context: moc)
      anime.name = "Anime"
      anime.episodes = 0
      anime.rating = 0
      anime.type = "Watching"
      anime.review = "ok"

      return NavigationView {
         AnimeDetailsView(anime: anime)
      }
   }
}
```

DetailsView-тата служат както за преглед така и за редактиране на вече въведената информация за Entity атрибутите. Както при View-тата за Create операцията @State променливи играят ролята на read/write модификатори. Прикачен е и същият ImagePicker. Промените се изпълняват при навигация обратно в списъка с обектите – извикването на функцията saveChanges в onDisappear. Новите данни заместват старите при асинхронно dispatch-ване на опашката в срок от 0.5 секунди.


iPod touch – 7th generation – iOS 15.0
Carrier
11:20 AM
Anime
Tokyo Revengers
Upload image
24 Episodes
9
Dropped
Completed
ReWatching
Plan to Watch
Review
Very captivating. Waiting impatiently


iPod touch – 7th generation – iOS 15.0
Carrier
11:22 AM
Back
Edit
Anime
Baccano!
Ep 13
6/10
Naruto:
Shippuuden
Ep 500
8/10
Tokyo
Revengers
Ep 24
9/10
Vinland Saga
Ep 24
10/10
Zetsuen
no Tempest
Ep 24
7/10

2.5. RecentlyAddedView

В RecentlyAddedView се визуализират последните три обекта от всички видове – аниме, манга, филми и сериали, като към всеки един от тях е закачен навигационен линк към съответстващото му DetailsView. По подразбиране резултатите са подредени според датата си на създаване :

```
@Environment(\.managedObjectContext) var moc

@FetchRequest(entity: Manga.entity(), sortDescriptors: [
    NSSortDescriptor(keyPath: \Manga.date, ascending: false)
]) var manga: FetchedResults<Manga>
@FetchRequest(entity: Anime.entity(), sortDescriptors: [
    NSSortDescriptor(keyPath: \Anime.date, ascending: false)
]) var anime: FetchedResults<Anime>
@FetchRequest(entity: TVShow.entity(), sortDescriptors: [
    NSSortDescriptor(keyPath: \TVShow.date, ascending: false)
]) var tvShows: FetchedResults<TVShow>
@FetchRequest(entity: Movie.entity(), sortDescriptors: [
    NSSortDescriptor(keyPath: \Movie.date, ascending: false)
]) var movies: FetchedResults<Movie>
```

След това в body-то колекциите от обекти се филтрират спрямо условието дали техния userUID е равен на uid-то на Firebase auth current user-а :

```
let currentUserUID = Auth.auth().currentUser?.uid

let filteredManga = manga.filter({ $0.userUID == currentUserUID })
let filteredAnime = anime.filter({ $0.userUID == currentUserUID })
let filteredTVShows = tvShows.filter({ $0.userUID == currentUserUID })
let filteredMovies = movies.filter({ $0.userUID == currentUserUID })
```

Последните добавени обекти са организирани в един обикновен List. Разделени са на отделни секции с вложени елементи за снимка, наименование, рейтинг, епизоди или чаптъри. Кодовата репрезентация на една такава секция изглежда по следния начин :

```
Section(header:Text("Movies")){
        if filteredMovies.count > 3 {
```

```
    ForEach(filteredMovies[0..<3],id: \.self){movie e
        NavigationLink(destination: MovieDetailsView(movie: movie)){
            VStack(alignment: .leading){
                HStack {
                    Image(uiImage: UIImage(data: movie.image ?? Data())!)
                        .resizable()
                        .frame(width: 50, alignment: .center)
                    Text(movie.name ?? "Unknown")
                        .font(.title2)
                    Spacer()
                    VStack {
                        Text("\(movie.rating)/10")
                            .font(.subheadline)
                            .foregroundColor(.secondary)
                        HStack(alignment: .center){
                            Image(systemName: "star.fill")
                                .foregroundColor(.yellow)

                        }
                    }
                }
            }
            .frame(height: 50)
        }
    }
}
else{
    ForEach(filteredMovies,id: \.self){movie in
        NavigationLink(destination: MovieDetailsView(movie: movie)){
            VStack(alignment: .leading){
                HStack {
                    Image(uiImage: UIImage(data: movie.image ?? Data())!)
                        .resizable()
                        .frame(width: 50, alignment: .center)
                    Text(movie.name ?? "Unknown")
                        .font(.title2)
                    Spacer()
                    VStack {
                        Text("\(movie.rating)/10")
                            .font(.subheadline)
```

```
                        .foregroundColor(.secondary)
                    HStack(alignment: .center){
                        Image(systemName: "star.fill")
                            .foregroundColor(.yellow)

                    }
                }
            }
        }
        .frame(height: 50)
    }
  }
 }
}
```

2.6. TopManga- и TopAnimeListView

Това са View-та, в които се fetch-ва API response-ът от Jikan url-ите https://api.jikan.moe/v3/top/manga/1 и респективно https://api.jikan.moe/v3/top/anime/1 . Те съдържат json-и с топ 50 рейтнатите манга и аниме в MyAnimeList.

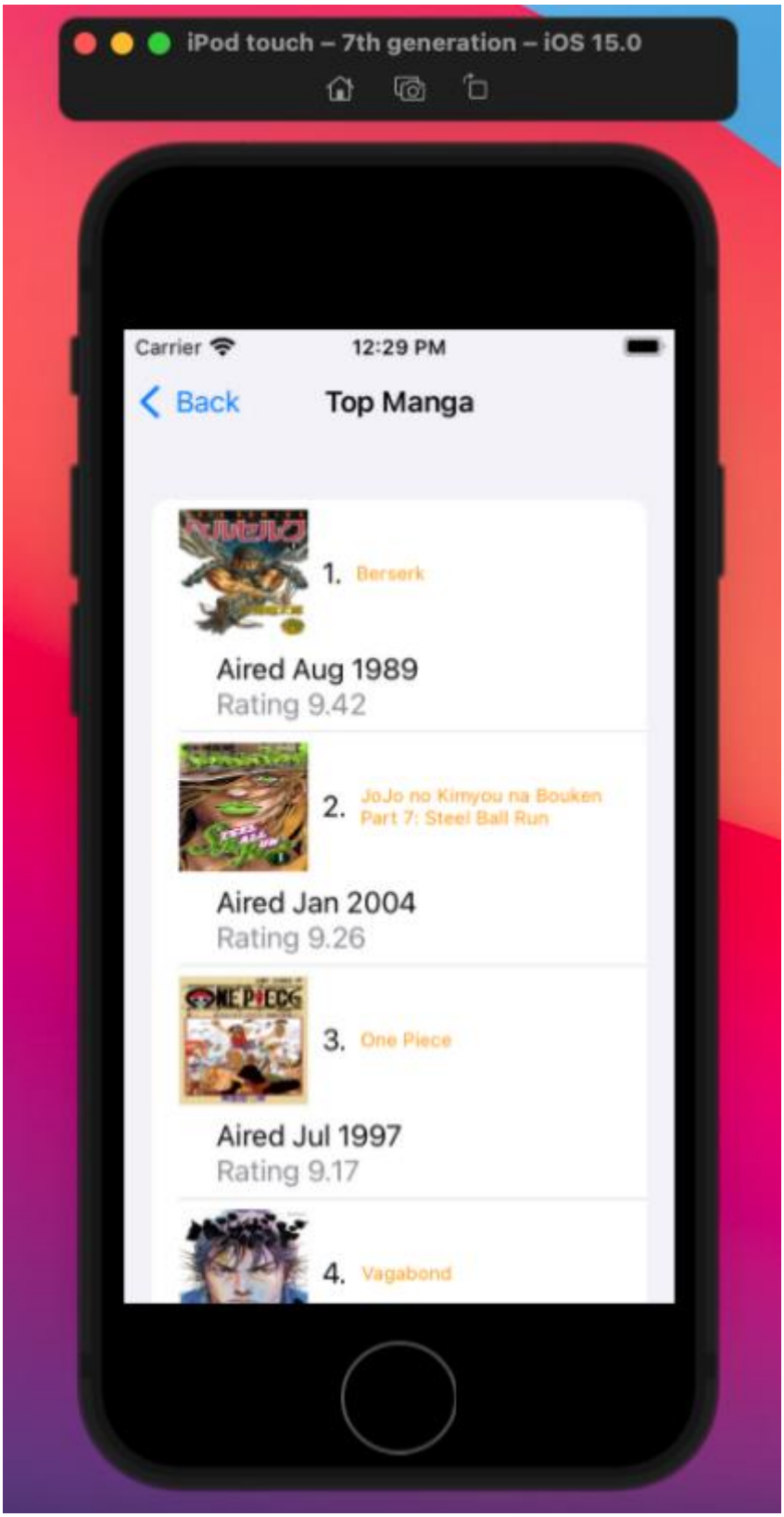
TopMangaListView :

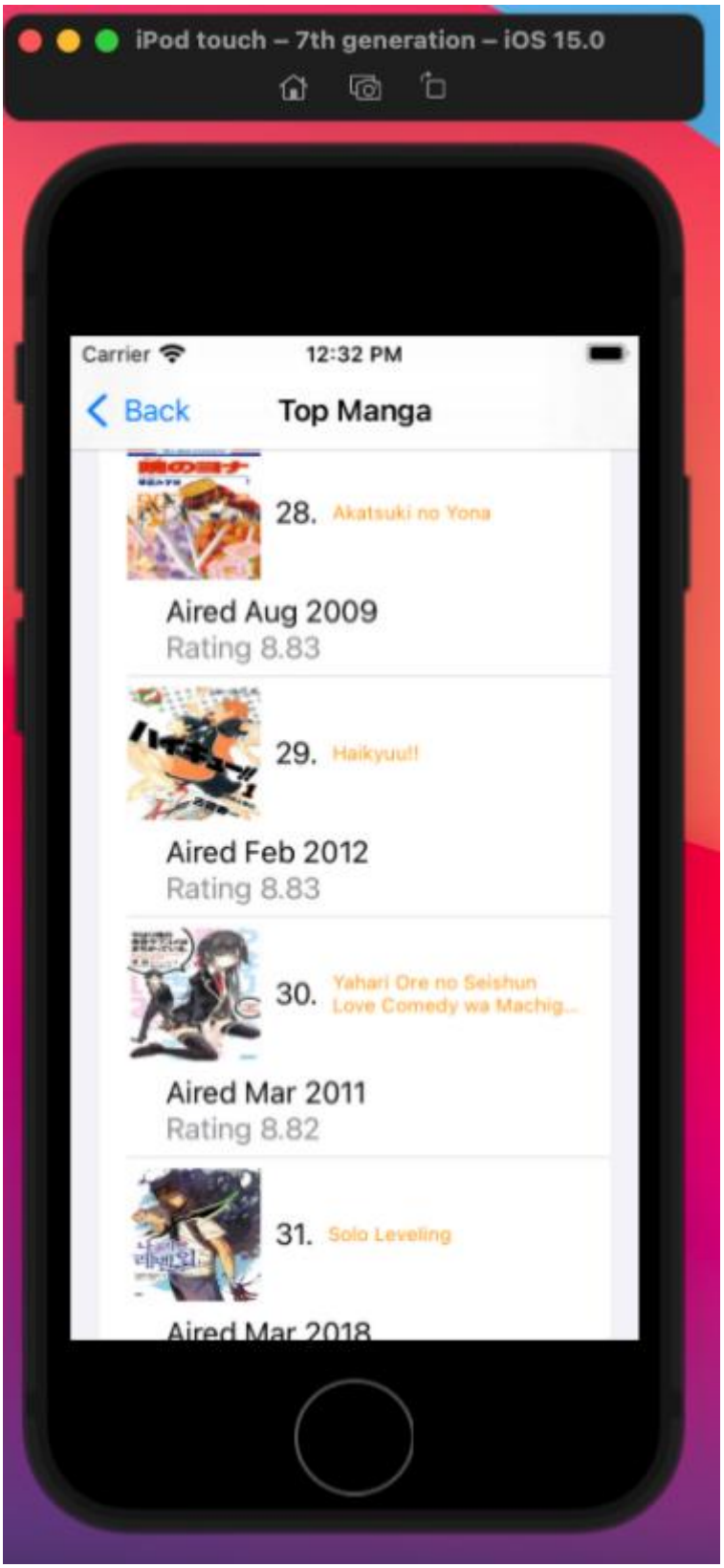
```
import SwiftUI
import Kingfisher

struct TopMangaData: Codable {

   var top: [TopManga]
```

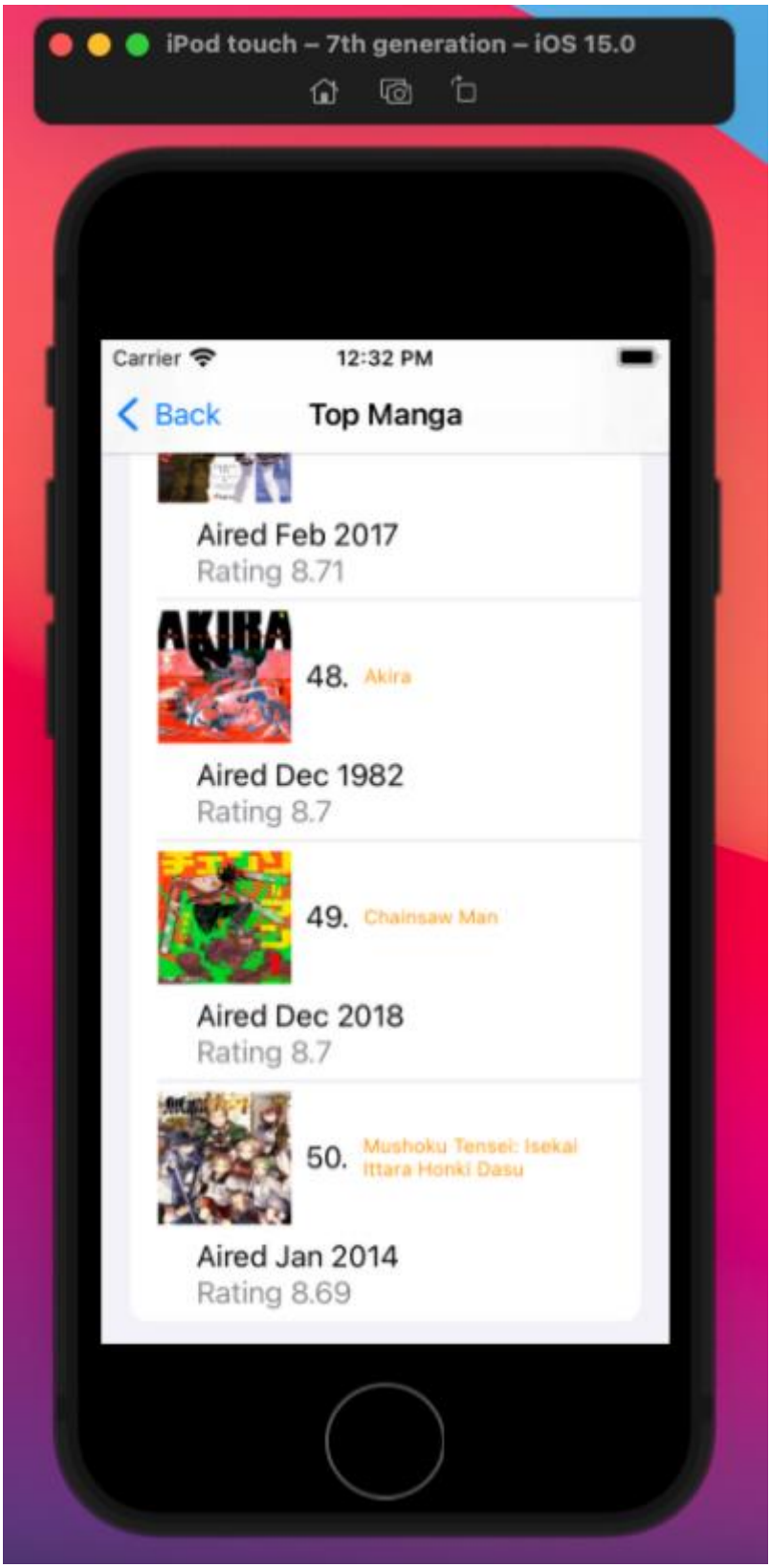
```
}
struct TopManga: Codable {
    var title: String
    var rank: Int
    var score: Double
    var start_date: String
    var image_url: String
}

struct TopMangaListView: View {
    @State var isNavigationBarHidden: Bool = true
    @State private var results = [TopManga]()
    var body: some View {

        List(results, id: \.rank) { item in
            VStack(alignment: .leading){

                HStack {
                    KFImage(URL(string: item.image_url)!)
                        .resizable()
                        .frame(width: 75, height: 75)
                    Text("\(item.rank).")
                    Text("\(item.title)")
                        .font(.caption2)
                        .foregroundColor(.orange)
                }
                Text("    Aired \(item.start_date)")
                Text("    Rating \(String(item.score))")
                    .foregroundColor(.secondary)


            }.navigationBarTitle("Top Manga",displayMode: .inline)
        }
        .onAppear(perform: topManga)
    }
    func topManga(){
        guard let url = URL(string: "https://api.jikan.moe/v3/top/manga/1") else {
            print("error")
            return
        }
```

```
        URLSession.shared.dataTask(with: url) { (data, response, error) in
            if let data = data{
                if let decodedResponse = try?
JSONDecoder().decode(TopMangaData.self, from: data){
                    DispatchQueue.main.async {
                        self.results = decodedResponse.top

                    }
                }
                return
            }
            print("Fetch failed: \(error?.localizedDescription ?? "Unknown error")")

        }.resume()
    }
}

struct TopMangaListView_Previews: PreviewProvider {
    static var previews: some View {
        TopMangaListView()
    }
}
```

Работи се с Codable структури, където се декларират променливи от определен тип. Във функцията topManga се извлича json response-ът. В последствие той се decode-ва чрез URLSession към колекция от обекти от TopManga структурата. Важно е имената на променливите да съответстват на ключовете от json-а. В противен случай ще се локализира грешка. Резултатите се извеждат в списък, където за конвертирането на image url-а към UIImage е инсталиран външният пакет KingFisher. Всъщност от iOS 15 това не е необходимо, защото може да се постигне с вградения за SwiftU AsyncImage. Освен снимката се появяват името, ранка, рейтинга и датата на излизане на мангата/аниметата.

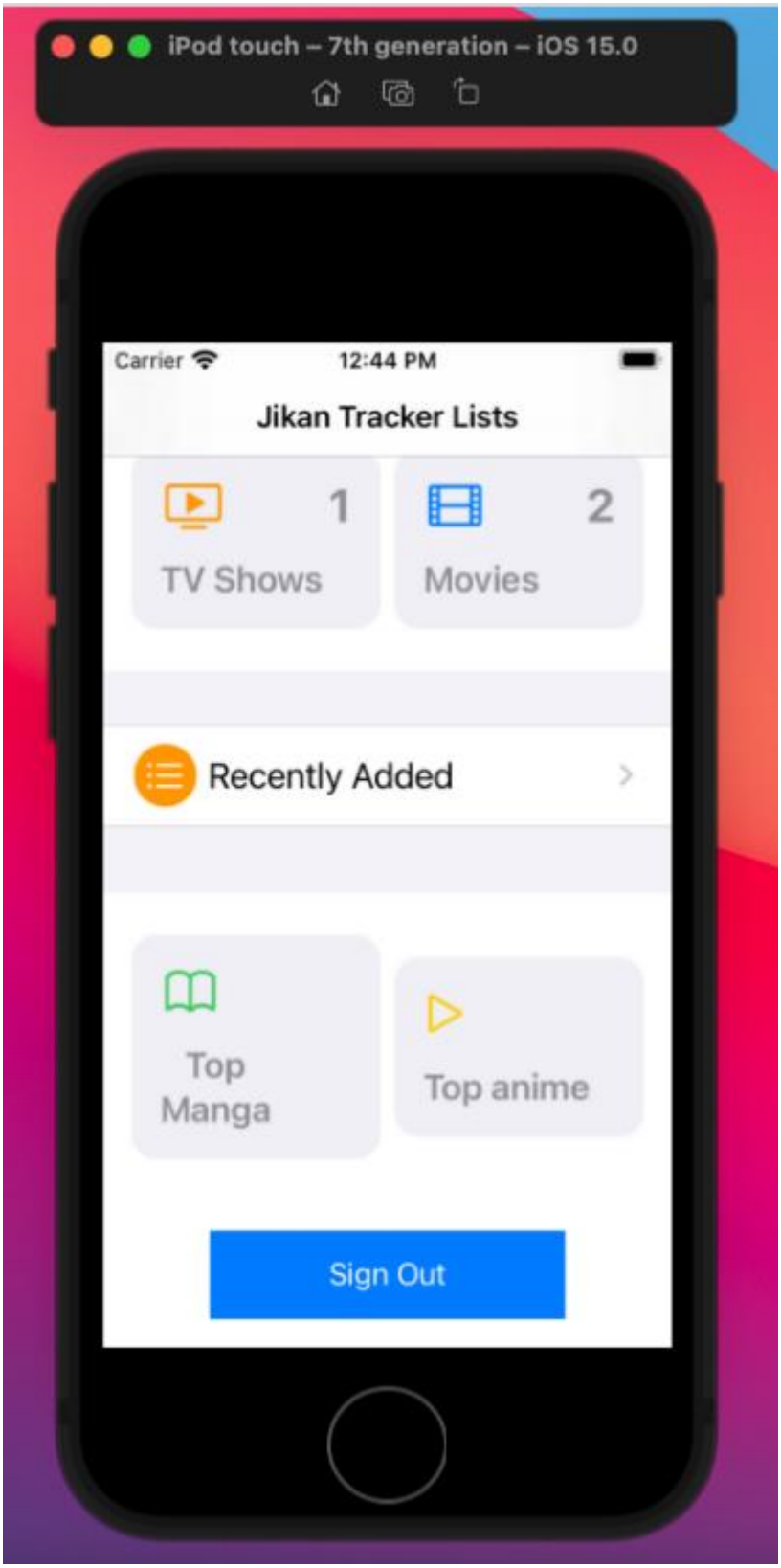

iPod touch – 7th generation – iOS 15.0
Carrier
12:29 PM
Back
Top Manga
1. Berserk
Aired Aug 1989
Rating 9.42
2. JoJo no Kimyou na Bouken Part 7: Steel Ball Run
Aired Jan 2004
Rating 9.26
3. One Piece
Aired Jul 1997
Rating 9.17
4. Vagabond


iPod touch – 7th generation – iOS 15.0
Carrier
12:32 PM
Back
Top Manga
28. Akatsuki no Yona
Aired Aug 2009
Rating 8.83
29. Haikyuu!!
Aired Feb 2012
Rating 8.83
30. Yahari Ore no Seishun Love Comedy wa Machig...
Aired Mar 2011
Rating 8.82
31. Solo Leveling
Aired Mar 2018


iPod touch – 7th generation – iOS 15.0
Carrier
12:32 PM
Back
Top Manga
Aired Feb 2017
Rating 8.71
48. Akira
Aired Dec 1982
Rating 8.7
49. Chainsaw Man
Aired Dec 2018
Rating 8.7
50. Mushoku Tensei: Isekai Ittara Honki Dasu
Aired Jan 2014
Rating 8.69

2.7. ContentView

ContentView е главният изглед на приложението. В него потребителят бива приветстван по имейл и приканен да разгледа и обогати списъците си с анимета, манга, филми и сериали, от които се интересува, да се информира за последните си записи както и да провери как се класират най-добрите 50 манга и анимета в реално време. Той може да влиза/излиза в/от системата по свое желание.


iPod touch – 7th generation – iOS 15.0
Carrier
12:44 PM
Jikan Tracker Lists
1
TV Shows
2
Movies
Recently Added
Top Manga
Top anime
Sign Out

Redirect-ите към ListView-тата са подредени в LazyVGrid, в който участва CustomGroup. CustomGroup-ът представлява View шаблон със системна иконка и Text елементи за брой и label. Той се преформатира спрямо предназначението на ListView-тата.

```
struct CustomGroup:View {
   var img = ""
   var count = ""
   var color:Color
   var label = ""

   var body: some View{
      VStack{
         GroupBox(label:
         HStack{
            Text("\(Image(systemName: img))")
               .foregroundColor(color)
               .font(.title)
            Spacer()
            Text(count)
               .foregroundColor(.gray)
               .font(.title)
               .fontWeight(.bold)
         }){
            VStack{
               Text("")
               HStack{
                  Text(label)
                     .font(.title2)
                     .foregroundColor(.gray)
                     .fontWeight(.semibold)
                  Spacer()
               }
            }
         }.cornerRadius(15)
      }
   }
}
```

```
LazyVGrid(columns: [.init(), .init()]){
                NavigationLink(destination: MangaListView()){
                  CustomGroup(img: "books.vertical", count:
"\(filteredManga.count)", color: Color.green, label: "Manga")
                }
                NavigationLink(destination: AnimeListView()){
                  CustomGroup(img: "tv", count: "\(filteredAnime.count)", color:
Color.red, label: "Anime")
                }
                NavigationLink(destination: TVShowsListView()){
                  CustomGroup(img: "play.tv", count:
"\(filteredTVShows.count)", color: Color.orange, label: "TV Shows")
                }
                NavigationLink(destination:
                  MoviesListView()) {
                  CustomGroup(img: "film", count: "\(filteredMovies.count)",
color: Color.blue, label: "Movies")
                }
              }.padding()
```

Същото се отнася и за TopAnime- и TopMangaListView-тата.

```
VStack{
                LazyVGrid(columns: [.init(), .init()]) {
                  NavigationLink(destination: TopMangaListView()) {
                    CustomGroup(img: "book", color: Color.green, label: "Top
Manga")
                  }
                  NavigationLink(destination: TopAnimeListView()) {
                    CustomGroup(img: "play", color: Color.yellow, label: "Top
anime")
                  }
                }.padding()
              }
```

А навигационния линк към RecentlyAddedView е във Form :

```
Form{
                HStack{
                  NavigationLink(destination: RecentlyAddedView()){
```

```
                Label(
                    title: {
                        Text("Recently Added")
                            .font(.title2)
                    },
                    iOS: {
                        Image(systemName: "list.bullet")
                            .padding(9)
                            .font(.title2)
                            .background(Color.orange)
                            .foregroundColor(.white)
                            .clipShape(Circle())
                    }
                )
            }
        }.padding(5)
    }.frame(height: 125)
```

При натискане на Sign Out бутона потребителят се log out-ва :

```
Button(action: {
            viewModel.signOut()
        }, label: {
            Text("Sign Out")
                .frame(width: 200, height: 50)
                .background(Color.blue)
                .foregroundColor(Color.white)
                .padding()
        })
```

В самата структура ContentView се извикват отделените View-та. Осъществява се навигация между тях спрямо signed in статута на потребителя.

```
struct ContentView: View {
    @EnvironmentObject var viewModel: AppViewModel

    var body: some View {
        NavigationView {
```

```
            if viewModel.signedIn {
                JikanTrackerView()
            } else {
                SignInView()
            }
        }
        .onAppear{
            viewModel.signedIn = viewModel.isSignedIn
        }
    }
}
```

За да се работи с CoreData обектите в main App структурата на ContentView се подава managedObjectContext-а за environment с viewContext-а на споделения PersistenceController.

```
let persistenceController = PersistenceController.shared

ContentView()
            .environment(\.managedObjectContext,
persistenceController.container.viewContext)
```

3. Заключение

Създадено е CRUD приложение с Image Upload и API fetching и decoding функционалности, базирано на SwiftUI фреймуърка. Аутентификацията се имплементира с помощта на Firebase. В CoreData се пази референция на потребителския идентификатор като Entity атрибут. Потребителите могат да засичат активността си за анимета, манга, филми и японски игрални сериали. Приложението е с лесен и удобен за ползване интерфейс. Промените по записите се извършват на момента, без излишни забавяния, а Open Source API услугите предоставят най-важната информация за любимите анимета и манга на MyAnimeList потребителите.

Линк към github repository-то на проекта : https://github.com/plamenna-petrova/JikanTracker