**import** SwiftUI

**import** FirebaseFirestore

**struct** ContactsView: View {

@EnvironmentObject **var** authManager: AuthManager

@State **private** **var** clients: [Client] = []

@State **private** **var** funktionäre: [Funktionär] = []

@State **private** **var** clubs: [Club] = []

@State **private** **var** searchText = ""

@State **private** **var** filterType: ContactFilterType = .all

@State **private** **var** filterClub: String? = **nil**

@State **private** **var** errorMessage = ""

@State **private** **var** imageCache: [String: UIImage] = [:]

@State **private** **var** showingAddContactSheet = **false**

@State **private** **var** favorites: Set<String> = [] // Für Favoriten

@State **private** **var** groupBy: GroupByOption = .none // Für Kategorisierung

**enum** ContactFilterType: String, CaseIterable {

**case** all = "Alle"

**case** clients = "Klienten"

**case** funktionäre = "Funktionäre"

}

**enum** GroupByOption: String, CaseIterable {

**case** none = "Keine"

**case** club = "Verein"

**case** type = "Typ"

}

// Gefilterte Kontakte

**var** filteredContacts: [ContactItem] {

**let** allContacts = clients.map { ContactItem(client: $0, isFavorite: favorites.contains($0.id ?? "")) } +

funktionäre.map { ContactItem(funktionär: $0, isFavorite: favorites.contains($0.id ?? "")) }

**return** allContacts.filter { contact **in**

**let** matchesSearch = searchText.isEmpty ||

contact.name.lowercased().contains(searchText.lowercased()) ||

(contact.clubName?.lowercased().contains(searchText.lowercased()) ?? **false**) // Suche nach Verein

**let** matchesType = filterType == .all ||

(filterType == .clients && contact.isClient) ||

(filterType == .funktionäre && !contact.isClient)

**let** matchesClub = filterClub == **nil** || contact.clubName == filterClub

**return** matchesSearch && matchesType && matchesClub

}.sorted {

**if** favorites.contains($0.id) && !favorites.contains($1.id) { **return** **true** }

**if** !favorites.contains($0.id) && favorites.contains($1.id) { **return** **false** }

**return** $0.name < $1.name

} // Favoriten zuerst, dann alphabetisch

}

// Gruppierte Kontakte

**var** groupedContacts: [(key: String, items: [ContactItem])] {

**switch** groupBy {

**case** .none:

**return** [("Alle Kontakte", filteredContacts)]

**case** .club:

**let** grouped = Dictionary(grouping: filteredContacts) { $0.clubName ?? "Ohne Verein" }

**return** grouped.map { (key: $0.key, items: $0.value) }.sorted { $0.key < $1.key }

**case** .type:

**let** grouped = Dictionary(grouping: filteredContacts) { $0.isClient ? "Klienten" : "Funktionäre" }

**return** grouped.map { (key: $0.key, items: $0.value) }.sorted { $0.key < $1.key }

}

}

**var** clubOptions: [String] {

**let** allClubs = (clients.compactMap { $0.vereinID } + funktionäre.compactMap { $0.vereinID }).uniqued()

**return** ["Alle"] + allClubs.sorted()

}

**var** body: **some** View {

NavigationStack {

VStack(spacing: 0) {

// Filter- und Suchleiste

VStack(spacing: 10) {

TextField("Suche nach Name oder Verein...", text: $searchText)

.textFieldStyle(RoundedBorderTextFieldStyle())

.padding(.horizontal)

HStack(spacing: 15) {

Picker("Typ", selection: $filterType) {

ForEach(ContactFilterType.allCases, id: \.**self**) { type **in**

Text(type.rawValue).tag(type)

}

}

.pickerStyle(SegmentedPickerStyle())

Picker("Verein", selection: $filterClub) {

ForEach(clubOptions, id: \.**self**) { club **in**

Text(club).tag(club == "Alle" ? String?.none : String?.some(club))

}

}

.pickerStyle(MenuPickerStyle())

Picker("Gruppieren", selection: $groupBy) {

ForEach(GroupByOption.allCases, id: \.**self**) { option **in**

Text(option.rawValue).tag(option)

}

}

.pickerStyle(MenuPickerStyle())

}

.padding(.horizontal)

}

.padding(.vertical, 10)

.background(Color(.systemGray6))

// Kontaktliste

List {

ForEach(groupedContacts, id: \.key) { group **in**

Section(header: Text(group.key)) {

ForEach(group.items) { contact **in**

NavigationLink(destination: contactDetailView(for: contact)) {

contactRow(for: contact)

}

.swipeActions(edge: .trailing) {

Button(role: .destructive) {

deleteContact(contact)

} label: {

Label("Löschen", systemImage: "trash")

}

Button {

editContact(contact)

} label: {

Label("Bearbeiten", systemImage: "pencil")

}

.tint(.blue)

}

.swipeActions(edge: .leading) {

Button {

toggleFavorite(contact)

} label: {

Label(favorites.contains(contact.id) ? "Entfernen" : "Favorit", systemImage: favorites.contains(contact.id) ? "star.slash" : "star")

}

.tint(.yellow)

}

}

}

}

}

.listStyle(InsetGroupedListStyle())

}

.navigationTitle("Kontakte")

.toolbar {

ToolbarItem(placement: .navigationBarTrailing) {

Menu {

Button(action: { showingAddContactSheet = **true** }) {

Label("Neuer Kontakt", systemImage: "plus")

}

Button(action: { exportToCSV() }) {

Label("Als CSV exportieren", systemImage: "square.and.arrow.up")

}

} label: {

Image(systemName: "ellipsis.circle")

}

}

}

.sheet(isPresented: $showingAddContactSheet) {

AddContactView(

onSave: { newContact **in**

**if** newContact.isClient {

FirestoreManager.shared.createClient(client: newContact.client!) { result **in**

handleSaveResult(result, type: "Klient")

}

} **else** {

FirestoreManager.shared.createFunktionär(funktionär: newContact.funktionär!) { result **in**

handleSaveResult(result, type: "Funktionär")

}

}

},

onCancel: { showingAddContactSheet = **false** }

)

}

.alert(isPresented: .constant(!errorMessage.isEmpty)) {

Alert(title: Text("Fehler"), message: Text(errorMessage), dismissButton: .default(Text("OK")) {

errorMessage = ""

})

}

.onAppear {

loadContacts()

loadClubs()

loadFavorites()

}

}

}

// Hilfsstruktur für einheitliche Kontaktanzeige

**struct** ContactItem: Identifiable {

**let** id: String

**let** name: String

**let** type: String

**let** clubName: String?

**let** profileImageURL: String?

**let** phone: String?

**let** email: String?

**let** isClient: Bool

**let** client: Client?

**let** funktionär: Funktionär?

**let** isFavorite: Bool

**init**(client: Client, isFavorite: Bool) {

**self**.id = client.id ?? UUID().uuidString

**self**.name = "\(client.vorname) \(client.name)"

**self**.type = client.typ

**self**.clubName = client.vereinID

**self**.profileImageURL = client.profilbildURL

**self**.phone = client.kontaktTelefon

**self**.email = client.kontaktEmail

**self**.isClient = **true**

**self**.client = client

**self**.funktionär = **nil**

**self**.isFavorite = isFavorite

}

**init**(funktionär: Funktionär, isFavorite: Bool) {

**self**.id = funktionär.id ?? UUID().uuidString

**self**.name = "\(funktionär.vorname) \(funktionär.name)"

**self**.type = "Funktionär"

**self**.clubName = funktionär.vereinID

**self**.profileImageURL = funktionär.profilbildURL

**self**.phone = funktionär.kontaktTelefon

**self**.email = funktionär.kontaktEmail

**self**.isClient = **false**

**self**.client = **nil**

**self**.funktionär = funktionär

**self**.isFavorite = isFavorite

}

}

// Kontaktzeile

@ViewBuilder

**private** **func** contactRow(for contact: ContactItem) -> **some** View {

HStack(spacing: 10) {

// Profilbild

**if** **let** url = contact.profileImageURL, **let** cachedImage = imageCache[url] {

Image(uiImage: cachedImage)

.resizable()

.scaledToFill()

.frame(width: 40, height: 40)

.clipShape(Circle())

.overlay(Circle().stroke(Color.gray, lineWidth: 1))

} **else** {

Image(systemName: "person.circle.fill")

.resizable()

.scaledToFill()

.frame(width: 40, height: 40)

.foregroundColor(.gray)

.clipShape(Circle())

.overlay(Circle().stroke(Color.gray, lineWidth: 1))

.onAppear { loadImage(url: contact.profileImageURL, key: contact.profileImageURL ?? "") }

}

// Name und Details

VStack(alignment: .leading, spacing: 2) {

HStack {

Text(contact.name)

.font(.headline)

**if** contact.isFavorite {

Image(systemName: "star.fill")

.foregroundColor(.yellow)

.font(.system(size: 12))

}

}

Text(contact.type)

.font(.subheadline)

.foregroundColor(.gray)

**if** **let** clubName = contact.clubName {

Text(clubName)

.font(.caption)

.foregroundColor(.secondary)

}

}

Spacer()

// Aktionsbuttons

HStack(spacing: 15) {

**if** **let** phone = contact.phone {

Button(action: { openURL("tel:\(phone)") }) {

Image(systemName: "phone.fill")

.foregroundColor(.blue)

}

}

**if** **let** email = contact.email {

Button(action: { openURL("mailto:\(email)") }) {

Image(systemName: "envelope.fill")

.foregroundColor(.blue)

}

}

}

}

.padding(.vertical, 5)

}

// Daten laden

**private** **func** loadContacts() {

FirestoreManager.shared.getClients { result **in**

**switch** result {

**case** .success(**let** loadedClients):

clients = loadedClients

loadedClients.forEach { loadImage(url: $0.profilbildURL, key: $0.profilbildURL ?? "") }

**case** .failure(**let** error):

errorMessage = "Fehler beim Laden der Klienten: \(error.localizedDescription)"

}

}

FirestoreManager.shared.getFunktionäre { result **in**

**switch** result {

**case** .success(**let** loadedFunktionäre):

funktionäre = loadedFunktionäre

loadedFunktionäre.forEach { loadImage(url: $0.profilbildURL, key: $0.profilbildURL ?? "") }

**case** .failure(**let** error):

errorMessage = "Fehler beim Laden der Funktionäre: \(error.localizedDescription)"

}

}

}

**private** **func** loadClubs() {

FirestoreManager.shared.getClubs { result **in**

**switch** result {

**case** .success(**let** loadedClubs):

clubs = loadedClubs

**case** .failure(**let** error):

errorMessage = "Fehler beim Laden der Vereine: \(error.localizedDescription)"

}

}

}

**private** **func** loadImage(url: String?, key: String) {

**guard** **let** urlString = url, **let** url = URL(string: urlString), imageCache[key] == **nil** **else** { **return** }

URLSession.shared.dataTask(with: url) { data, \_, \_ **in**

**if** **let** data = data, **let** image = UIImage(data: data) {

DispatchQueue.main.async {

imageCache[key] = image

}

}

}.resume()

}

// Favoriten laden (z. B. aus UserDefaults oder Firestore)

**private** **func** loadFavorites() {

**if** **let** userID = authManager.userID, **let** data = UserDefaults.standard.data(forKey: "favorites\_\(userID)"),

**let** savedFavorites = **try**? JSONDecoder().decode(Set<String>.**self**, from: data) {

favorites = savedFavorites

}

}

**private** **func** saveFavorites() {

**if** **let** userID = authManager.userID, **let** data = **try**? JSONEncoder().encode(favorites) {

UserDefaults.standard.set(data, forKey: "favorites\_\(userID)")

}

}

// Detailansicht

@ViewBuilder

**private** **func** contactDetailView(for contact: ContactItem) -> **some** View {

**if** contact.isClient, **let** client = contact.client {

// Erstelle ein Binding für den Client aus der clients-Liste

**if** **let** index = clients.firstIndex(where: { $0.id == client.id }) {

ClientView(client: $clients[index]) // Verwende Binding zum Array-Element

} **else** {

Text("Client nicht in der Liste gefunden")

}

} **else** **if** **let** funktionär = contact.funktionär {

FunktionärView(funktionär: .constant(funktionär))

} **else** {

Text("Kontakt nicht verfügbar")

}

}

// Aktionen

**private** **func** openURL(\_ urlString: String) {

**if** **let** url = URL(string: urlString), UIApplication.shared.canOpenURL(url) {

UIApplication.shared.open(url)

}

}

**private** **func** toggleFavorite(\_ contact: ContactItem) {

**if** favorites.contains(contact.id) {

favorites.remove(contact.id)

} **else** {

favorites.insert(contact.id)

}

saveFavorites()

}

**private** **func** deleteContact(\_ contact: ContactItem) {

**if** contact.isClient, **let** clientID = contact.client?.id {

FirestoreManager.shared.deleteClient(clientID: clientID) { result **in**

handleDeleteResult(result)

}

} **else** **if** **let** funktionärID = contact.funktionär?.id {

FirestoreManager.shared.deleteFunktionär(funktionärID: funktionärID) { result **in**

handleDeleteResult(result)

}

}

}

**private** **func** editContact(\_ contact: ContactItem) {

showingAddContactSheet = **true**

// Hier könntest du eine EditContactView implementieren, aber für jetzt nutzen wir AddContactView

}

**private** **func** exportToCSV() {

**let** csvString = generateCSV(from: filteredContacts)

**let** activityVC = UIActivityViewController(activityItems: [csvString], applicationActivities: **nil**)

**if** **let** windowScene = UIApplication.shared.connectedScenes.first **as**? UIWindowScene,

**let** rootVC = windowScene.windows.first?.rootViewController {

rootVC.present(activityVC, animated: **true**)

}

}

**private** **func** generateCSV(from contacts: [ContactItem]) -> String {

**var** csv = "Name,Typ,Verein,Telefon,E-Mail,Favorit\n"

**for** contact **in** contacts {

**let** row = [

contact.name,

contact.type,

contact.clubName ?? "",

contact.phone ?? "",

contact.email ?? "",

contact.isFavorite ? "Ja" : "Nein"

].map { "\"\($0.replacingOccurrences(of: "\"", with: "\"\""))\"" }.joined(separator: ",")

csv += row + "\n"

}

**return** csv

}

**private** **func** handleSaveResult(\_ result: Result<Void, Error>, type: String) {

**switch** result {

**case** .success:

loadContacts()

showingAddContactSheet = **false**

**case** .failure(**let** error):

errorMessage = "Fehler beim Speichern des \(type)s: \(error.localizedDescription)"

}

}

**private** **func** handleDeleteResult(\_ result: Result<Void, Error>) {

**switch** result {

**case** .success:

loadContacts()

**case** .failure(**let** error):

errorMessage = "Fehler beim Löschen: \(error.localizedDescription)"

}

}

}

// Platzhalter für AddContactView (bereits vorhanden, hier unverändert)

**struct** AddContactView: View {

**let** onSave: (ContactsView.ContactItem) -> Void

**let** onCancel: () -> Void

@State **private** **var** isClient = **true**

@State **private** **var** client = Client(id: **nil**, typ: "Spieler", name: "", vorname: "", geschlecht: "männlich", vereinID: **nil**, nationalitaet: [], geburtsdatum: **nil**, liga: **nil**, profilbildURL: **nil**)

@State **private** **var** funktionär = Funktionär(id: **nil**, name: "", vorname: "", kontaktTelefon: **nil**, kontaktEmail: **nil**, adresse: **nil**, clients: **nil**)

**var** body: **some** View {

NavigationView {

Form {

Picker("Kontakt-Typ", selection: $isClient) {

Text("Klient").tag(**true**)

Text("Funktionär").tag(**false**)

}

.pickerStyle(SegmentedPickerStyle())

**if** isClient {

AddClientView(client: $client, isEditing: **false**, onSave: { onSave(.init(client: $0, isFavorite: **false**)) }, onCancel: onCancel)

} **else** {

AddFunktionärView(funktionär: $funktionär, onSave: { onSave(.init(funktionär: $0, isFavorite: **false**)) }, onCancel: onCancel)

}

}

.navigationTitle("Neuer Kontakt")

}

}

}

#Preview {

ContactsView()

.environmentObject(AuthManager())

}

**extension** Array **where** Element: Hashable {

**func** uniqued() -> [Element] {

Array(Set(**self**))

}

}