**import** SwiftUI

// For iOS Share Sheet

#if os(iOS)

**import** UIKit

#endif

// New RenameProgramDialog: cleaned-up implementation

**struct** RenameProgramDialog: View {

/// Whether the dialog is currently shown.

@Binding **var** isPresented: Bool

/// The current text the user can edit; bound to a text field.

@Binding **var** text: String

/// The headline shown at the top of the dialog, e.g. “Name ändern”.

**let** title: String

/// A subtitle prompting the user what to enter, e.g. “Gib einen Namen für das Programm ein.”.

**let** subtitle: String

/// Called when the user taps the confirm button after entering a non‑empty name.

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

/// Manages focus for the text field to automatically focus when the view appears.

@FocusState **private** **var** isFocused: Bool

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

VStack(spacing: 20) {

// Title and subtitle

VStack(alignment: .leading, spacing: 6) {

Text(title)

.font(.system(size: 22, weight: .bold))

.foregroundColor(.white)

Text(subtitle)

.font(.system(size: 15))

.foregroundColor(.white.opacity(0.7))

.fixedSize(horizontal: **false**, vertical: **true**)

}

.frame(maxWidth: .infinity, alignment: .leading)

// Input field with optional clear button

HStack(spacing: 8) {

TextField(title, text: $text)

.textInputAutocapitalization(.words)

.disableAutocorrection(**true**)

.focused($isFocused)

.foregroundColor(.white)

**if** !text.isEmpty {

Button {

text = ""

} label: {

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

.font(.system(size: 18, weight: .semibold))

.foregroundColor(.white.opacity(0.6))

}

}

}

.padding(.horizontal, 18)

.frame(height: 54)

.background(

RoundedRectangle(cornerRadius: 16)

.fill(Color.white.opacity(0.10))

)

// Action buttons

HStack(spacing: 12) {

// Cancel button

Button {

isPresented = **false**

} label: {

Text("Abbrechen")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white)

.frame(maxWidth: .infinity)

.frame(height: 52)

.background(

RoundedRectangle(cornerRadius: 16)

.fill(Color.white.opacity(0.18))

)

}

// Confirm button

Button {

**let** trimmed = text.trimmingCharacters(in: .whitespacesAndNewlines)

**guard** !trimmed.isEmpty **else** { **return** }

onConfirm()

isPresented = **false**

} label: {

Text("Ändern")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white.opacity(text.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty ? 0.35 : 1))

.frame(maxWidth: .infinity)

.frame(height: 52)

.background(

RoundedRectangle(cornerRadius: 16)

.fill(text.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty ? Color.white.opacity(0.10) : Color.pink)

)

}

.disabled(text.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty)

}

}

.padding(22)

.background(

RoundedRectangle(cornerRadius: 26)

.fill(Color(red: 0.13, green: 0.13, blue: 0.13))

.overlay(

RoundedRectangle(cornerRadius: 26)

.stroke(Color.white.opacity(0.08), lineWidth: 1)

)

)

.padding(.horizontal, 28)

.onAppear {

// Focus the text field shortly after the view appears so that the keyboard pops up.

DispatchQueue.main.asyncAfter(deadline: .now() + 0.1) {

isFocused = **true**

}

}

}

}

// **MARK: - Models**

**struct** Exercise: Identifiable, Hashable, Codable {

**var** id = UUID()

**let** name: String

**let** category: String

**let** type: String

**let** iconName: String

}

**struct** AddedExercise: Identifiable, Hashable, Codable {

**var** id = UUID()

**let** exercise: Exercise

**var** weight: String = "KG"

**var** sets: Int = 5

**var** reps: Int = 5

}

**struct** RoutineData: Identifiable, Hashable, Codable {

**var** id = UUID()

**var** name: String

**var** exercises: [AddedExercise] = []

/// The activity type associated with this routine. Defaults to traditional strength training.

**var** activityType: RoutineActivityType = .traditionellesKrafttraining

}

**struct** ProgramData: Identifiable, Hashable, Codable {

**var** id = UUID()

**var** name: String

**var** routines: [RoutineData]

}

// **MARK: - Activity Type**

/// Describes available activity types for a routine. Each case contains a user-visible title,

/// description and SF Symbols icon to display in the change activity dialog.

**enum** RoutineActivityType: String, CaseIterable, Codable {

**case** traditionellesKrafttraining = "Traditionelles Krafttraining"

**case** funktionellesKrafttraining = "Funktionelles Krafttraining"

**case** crossTraining = "Cross-Training"

**case** core = "Core"

**case** biegsamkeit = "Biegsamkeit"

/// A human‑readable description of the activity type.

**var** description: String {

**switch** **self** {

**case** .traditionellesKrafttraining:

**return** "Krafttrainingsübungen, hauptsächlich mit Maschinen oder freien Gewichten."

**case** .funktionellesKrafttraining:

**return** "Krafttraining, vorwiegend mit freien Gewichten und dem eigenen Körpergewicht."

**case** .crossTraining:

**return** "Training, das eine beliebige Mischung aus Cardio-, Kraft- und/oder Flexibilitätstraining umfasst."

**case** .core:

**return** "Training für die Rumpfmuskulatur."

**case** .biegsamkeit:

**return** "Beweglichkeit und Dehnungstraining."

}

}

/// The SF Symbol representing the activity type.

**var** icon: String {

**switch** **self** {

**case** .traditionellesKrafttraining:

**return** "figure.strengthtraining.traditional"

**case** .funktionellesKrafttraining:

**return** "figure.strengthtraining.functional"

**case** .crossTraining:

**return** "figure.cross.training"

**case** .core:

**return** "figure.core.training"

**case** .biegsamkeit:

**return** "figure.cooldown"

}

}

}

// **MARK: - Share Sheet (UIKit bridge)**

#if os(iOS)

**struct** ShareSheet: UIViewControllerRepresentable {

**let** items: [**Any**]

**var** subject: String? = **nil**

**var** completion: UIActivityViewController.CompletionWithItemsHandler? = **nil**

**func** makeUIViewController(context: Context) -> UIActivityViewController {

**let** vc = UIActivityViewController(activityItems: items, applicationActivities: **nil**)

**if** **let** subject { vc.setValue(subject, forKey: "subject") }

**return** vc

}

**func** updateUIViewController(\_ uiViewController: UIActivityViewController, context: Context) {}

}

#endif

// **MARK: - Share Helpers**

**func** makeProgramShareItems(program: ProgramData) -> [**Any**] {

**var** items: [**Any**] = []

// Human-readable summary

**var** lines: [String] = []

lines.append(" 🏋️‍♀️ Programm: \(program.name)")

lines.append("")

**for** (rIdx, routine) **in** program.routines.enumerated() {

lines.append("• Routine \(rIdx + 1): \(routine.name)")

**if** routine.exercises.isEmpty {

lines.append(" — keine Übungen")

} **else** {

**for** ex **in** routine.exercises {

lines.append(" — \(ex.exercise.name) (\(ex.exercise.category), \(ex.exercise.type)) · \(ex.sets)x\(ex.reps) \(ex.weight)")

}

}

lines.append("")

}

items.append(lines.joined(separator: "\n"))

// JSON export file

**if** **let** url = exportProgramAsJSON(program: program) {

items.append(url)

}

**return** items

}

**func** exportProgramAsJSON(program: ProgramData) -> URL? {

**do** {

**let** encoder = JSONEncoder()

encoder.outputFormatting = [.prettyPrinted, .sortedKeys]

**let** data = **try** encoder.encode(program)

**let** filename = program.name.replacingOccurrences(of: "/", with: "-")

**let** url = FileManager.default.temporaryDirectory.appendingPathComponent("\(filename).json")

**try** data.write(to: url, options: .atomic)

**return** url

} **catch** {

print("JSON export failed:", error)

**return** **nil**

}

}

// **MARK: - ViewModel**

@MainActor

**class** TrainingViewModel: ObservableObject {

@Published **var** programs: [ProgramData] = []

**func** addProgram(\_ program: ProgramData) {

// --- MODIFIKATION ---

// Fügt das neue Programm an den Anfang der Liste hinzu,

// damit es das "aktive" Programm wird.

programs.insert(program, at: 0)

// --- ENDE MODIFIKATION ---

}

// --- NEUE FUNKTION ---

/// Setzt ein Programm als "aktiv", indem es an den Anfang der Liste verschoben wird.

**func** setActiveProgram(id: UUID) {

**guard** **let** index = programs.firstIndex(where: { $0.id == id }) **else** { **return** }

// Nur etwas tun, wenn es nicht schon das aktive ist

**if** index != 0 {

**let** program = programs.remove(at: index)

programs.insert(program, at: 0)

}

}

// --- ENDE NEUE FUNKTION ---

**func** updateRoutines(for programId: UUID, routines: [RoutineData]) {

**guard** **let** idx = programs.firstIndex(where: { $0.id == programId }) **else** { **return** }

programs[idx].routines = routines

}

**func** updateProgramName(id: UUID, newName: String) {

**guard** **let** idx = programs.firstIndex(where: { $0.id == id }) **else** { **return** } // HIER WAR DER FEHLER: programId wurde zu id geändert

programs[idx].name = newName

}

**func** deleteProgram(id: UUID) {

programs.removeAll { $0.id == id }

}

}

// **MARK: - Content**

// **MARK: - Duplicate Routine Dialog**

/// A dialog that allows the user to enter a name for a duplicated routine before it is created.

/// Displays a title, a subtitle prompting for a new name, a text field with a placeholder

/// and two actions: "Abbrechen" to dismiss and "Fortfahren" to confirm. The confirm button becomes

/// fully opaque when the input is not empty and duplicates the routine via the provided `onConfirm` callback.

**struct** DuplicateRoutineDialog: View {

/// Whether this dialog is currently presented.

@Binding **var** isPresented: Bool

/// The name entered by the user for the new routine. This binding is mutated directly by the text field.

@Binding **var** text: String

/// Called when the user taps the Fortfahren button. Use this closure to perform the duplication.

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

/// Tracks focus for the text field so that it becomes first responder when the dialog appears.

@FocusState **private** **var** isFocused: Bool

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

VStack(spacing: 24) {

// Heading

VStack(alignment: .leading, spacing: 12) {

Text("Routine duplizieren")

.font(.system(size: 24, weight: .bold))

.foregroundColor(.white)

Text("Gib einen Namen für die neue Routine ein.")

.font(.system(size: 16))

.foregroundColor(.white.opacity(0.7))

}

.frame(maxWidth: .infinity, alignment: .leading)

// Input field

TextField("Bsp. Rücken und Bizeps", text: $text)

.textInputAutocapitalization(.words)

.disableAutocorrection(**true**)

.focused($isFocused)

.foregroundColor(.white)

.padding(.horizontal, 20)

.padding(.vertical, 16)

.background(

RoundedRectangle(cornerRadius: 16)

.fill(Color.white.opacity(0.08))

)

// Action buttons

HStack(spacing: 12) {

// Cancel button

Button {

isPresented = **false**

} label: {

Text("Abbrechen")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white)

.frame(maxWidth: .infinity)

.frame(height: 52)

.background(

RoundedRectangle(cornerRadius: 16)

.fill(Color.white.opacity(0.18))

)

}

// Confirm button

Button {

// Trim whitespace and only proceed if non-empty

**let** trimmed = text.trimmingCharacters(in: .whitespacesAndNewlines)

**guard** !trimmed.isEmpty **else** { **return** }

onConfirm()

isPresented = **false**

} label: {

Text("Fortfahren")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(

.white.opacity(

text.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty ? 0.35 : 1

)

)

.frame(maxWidth: .infinity)

.frame(height: 52)

.background(

RoundedRectangle(cornerRadius: 16)

.fill(

text.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty

? Color.white.opacity(0.10) : Color.pink

)

)

}

.disabled(text.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty)

}

}

.padding(22)

.background(

RoundedRectangle(cornerRadius: 26)

.fill(Color(red: 0.13, green: 0.13, blue: 0.13))

.overlay(

RoundedRectangle(cornerRadius: 26)

.stroke(Color.white.opacity(0.08), lineWidth: 1)

)

)

.padding(.horizontal, 28)

.onAppear {

// Automatically focus the text field shortly after the dialog appears

DispatchQueue.main.asyncAfter(deadline: .now() + 0.1) { isFocused = **true** }

}

}

}

**struct** ContentView: View {

@StateObject **private** **var** viewModel = TrainingViewModel()

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

TabView {

TrainingMainView()

.environmentObject(viewModel)

.tabItem { Image(systemName: "dumbbell"); Text("Training") }

ProgressViewScreen()

.tabItem { Image(systemName: "chart.bar"); Text("Fortschritt") }

DiscoverViewScreen()

.tabItem { Image(systemName: "star"); Text("Entdecken") }

SettingsViewScreen()

.tabItem { Image(systemName: "gearshape"); Text("Einstellungen") }

}

.tint(.pink)

.preferredColorScheme(.dark)

}

}

// **MARK: - PreferenceKey (ProgramCard Frame)**

**private** **struct** ProgramCardFrameKey: PreferenceKey {

**static** **var** defaultValue: CGRect = .zero

**static** **func** reduce(value: **inout** CGRect, nextValue: () -> CGRect) { value = nextValue() }

}

// **MARK: - Training Main View**

**struct** TrainingMainView: View {

@EnvironmentObject **var** viewModel: TrainingViewModel

@State **private** **var** showAddProgramSheet = **false**

@State **private** **var** showRoutineDetail = **false** // Triggers the detail sheet

@State **private** **var** showRoutineEditPill = **false**

@State **private** **var** programCardFrame: CGRect = .zero

@State **private** **var** showProgramEdit = **false**

// --- MODIFICATION ---

// This new state holds \*which\* routine was tapped

@State **private** **var** selectedRoutine: RoutineData? = **nil**

// --- END MODIFICATION ---

// --- NEUE MODIFIKATION ---

@State **private** **var** showMainMenu = **false**

@State **private** **var** showShareSheet = **false**

@State **private** **var** shareItems: [**Any**] = []

@State **private** **var** showProgramSwitcherView = **false** // <-- HIER HINZUGEFÜGT

// --- ENDE NEUE MODIFIKATION ---

**let** routineCardColors: [Color] = [

Color(red: 0.2, green: 0.3, blue: 0.3), // Dark Teal/Green

Color(red: 0.25, green: 0.2, blue: 0.3), // Dark Purple

Color(red: 0.25, green: 0.28, blue: 0.27) // Original Color

]

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

ZStack {

Color.black.ignoresSafeArea()

**if** viewModel.programs.isEmpty {

VStack(spacing: 32) {

Text("Training")

.font(.system(size: 42, weight: .bold))

.foregroundColor(.white)

.frame(maxWidth: .infinity, alignment: .leading)

.padding(.horizontal, 24)

Spacer(minLength: 0)

OnboardingCard(showAddProgramSheet: $showAddProgramSheet)

.frame(maxWidth: 360)

.padding(.horizontal, 24)

Spacer(minLength: 0)

}

.padding(.top, 40)

} **else** {

ScrollView {

VStack(alignment: .leading, spacing: 24) {

HStack {

Text("Training")

.font(.system(size: 42, weight: .bold))

.foregroundColor(.white)

Spacer()

HStack(spacing: 12) {

Button { showProgramEdit = **true** } label: { // <-- MODIFIKATION HIER

Image(systemName: "pencil")

.font(.system(size: 18, weight: .semibold))

.foregroundColor(.white)

.frame(width: 44, height: 44)

.background(RoundedRectangle(cornerRadius: 22).fill(Color.white.opacity(0.15)))

}

Button { withAnimation(.easeInOut(duration: 0.1)) { showMainMenu = **true** } } label: { // <-- MODIFIKATION HIER

Image(systemName: "ellipsis")

.font(.system(size: 18, weight: .semibold))

.foregroundColor(.white)

.frame(width: 44, height: 44)

.background(RoundedRectangle(cornerRadius: 22).fill(Color.white.opacity(0.15)))

}

}

}

.padding(.horizontal, 24)

.padding(.top, 60)

VStack(alignment: .leading, spacing: 16) {

// MODIFIKATION: Zeigt nur das erste Programm (das aktive) an

**if** **let** program = viewModel.programs.first {

HStack {

Text(program.name)

.font(.system(size: 28, weight: .bold))

.foregroundColor(.white)

Image(systemName: "chevron.right")

.font(.system(size: 20, weight: .semibold))

.foregroundColor(.white.opacity(0.5))

}

.padding(.horizontal, 24)

ScrollView(.horizontal, showsIndicators: **false**) {

HStack(spacing: 16) {

ForEach(program.routines.indices, id: \.**self**) { index **in**

**let** routine = program.routines[index]

RoutineCard(

routine: routine,

isNext: index == 0,

cardColor: routineCardColors[index % routineCardColors.count],

onTap: {

// --- MODIFICATION ---

// Store the tapped routine and show the sheet

selectedRoutine = routine

showRoutineDetail = **true**

// --- END MODIFICATION ---

}

)

.background(

GeometryReader { proxy **in**

Color.clear.preference(

key: ProgramCardFrameKey.**self**,

value: index == 0 ? proxy.frame(in: .named("TrainingSpace")) : .zero

)

}

)

.highPriorityGesture(

index == 0 ?

LongPressGesture(minimumDuration: 0.35).onEnded { \_ **in**

withAnimation(.spring(response: 0.25, dampingFraction: 0.9)) {

showRoutineEditPill = **true**

}

}

: **nil**

)

}

}

.padding(.horizontal, 24)

}

}

}

VStack(alignment: .leading, spacing: 16) {

HStack {

Text("Ad-hoc-Workouts")

.font(.system(size: 28, weight: .bold))

.foregroundColor(.white)

Image(systemName: "chevron.right")

.font(.system(size: 20, weight: .semibold))

.foregroundColor(.white.opacity(0.5))

}

.padding(.horizontal, 24)

ScrollView(.horizontal, showsIndicators: **false**) {

HStack(spacing: 16) {

AdHocCard(icon: "play.fill", title: "Starten Sie das\nleere Training", color: .pink)

AdHocCard(icon: "sparkles", title: "Training\ngenerieren", color: .pink)

AdHocCard(icon: "plus", title: "Erstellen Sie\neine neue\nAd-hoc-\nWorkout", color: Color.white.opacity(0.15))

}

.padding(.horizontal, 24)

}

}

.padding(.top, 32)

}

.padding(.bottom, 80)

}

}

**if** showRoutineEditPill {

Color.black.opacity(0.001).ignoresSafeArea()

.onTapGesture { withAnimation(.easeOut(duration: 0.15)) { showRoutineEditPill = **false** } }

ZStack(alignment: .topLeading) {

RoutineEditPill {

withAnimation(.easeOut(duration: 0.15)) {

showRoutineEditPill = **false**

showProgramEdit = **true**

}

}

.offset(x: programCardFrame.minX, y: programCardFrame.maxY + 8)

.transition(AnyTransition.move(edge: .top).combined(with: .opacity))

}

.frame(maxWidth: .infinity, maxHeight: .infinity, alignment: .topLeading)

}

// --- NEUE MODIFIKATION: Hauptmenü-Overlay ---

**if** showMainMenu {

Color.black.opacity(0.4).ignoresSafeArea().onTapGesture { withAnimation(.easeInOut(duration: 0.1)) { showMainMenu = **false** } }

VStack {

HStack {

Spacer()

TrainingMainMenuView(

onShareProgram: {

withAnimation(.easeInOut(duration: 0.1)) { showMainMenu = **false** }

// Verzögerung, damit das Menü sich schließen kann, bevor das Share-Sheet erscheint

DispatchQueue.main.asyncAfter(deadline: .now() + 0.15) {

presentShareSheetForCurrentProgram()

}

},

onShowAllPrograms: {

withAnimation(.easeInOut(duration: 0.1)) { showMainMenu = **false** }

// --- NEUE AKTION ---

// Aktion für "Meine Programme" hier einfügen

DispatchQueue.main.asyncAfter(deadline: .now() + 0.15) {

showProgramSwitcherView = **true** // Öffnet das neue Sheet

}

// --- ENDE NEUE AKTION ---

}

)

.padding(.trailing, 20)

.padding(.top, 110) // Position unter dem Header

}

Spacer()

}

.transition(.opacity.combined(with: .scale(scale: 0.9, anchor: .topTrailing)))

}

// --- ENDE NEUE MODIFIKATION ---

}

.coordinateSpace(name: "TrainingSpace")

.onPreferenceChange(ProgramCardFrameKey.**self**) { programCardFrame = $0 }

.sheet(isPresented: $showAddProgramSheet) {

AddProgramSheet(isPresented: $showAddProgramSheet).environmentObject(viewModel)

}

// --- MODIFICATION ---

// The sheet now opens RoutineDetailSheet and passes the specific routine

.sheet(isPresented: $showRoutineDetail) {

**if** **let** routine = selectedRoutine, **let** program = viewModel.programs.first {

// Pass the program ID (for saving) and the specific routine

RoutineDetailSheet(programId: program.id, routine: routine)

.environmentObject(viewModel) // Pass the view model for saving

}

}

// --- END MODIFICATION ---

.fullScreenCover(isPresented: $showProgramEdit) {

**if** **let** program = viewModel.programs.first {

ProgramEditView(program: program).environmentObject(viewModel)

}

}

// --- NEUE MODIFIKATION: Share Sheet ---

#if os(iOS)

.sheet(isPresented: $showShareSheet) {

**if** **let** program = viewModel.programs.first {

ShareSheet(items: shareItems, subject: "Programm: \(program.name)")

}

}

#endif

// --- ENDE NEUE MODIFIKATION ---

// --- NEUES FULLSCREEN COVER FÜR PROGRAMM-WECHSLER ---

.fullScreenCover(isPresented: $showProgramSwitcherView) {

ProgramSwitcherView(

isPresented: $showProgramSwitcherView,

showAddProgramSheet: $showAddProgramSheet

)

.environmentObject(viewModel)

}

// --- ENDE NEUES FULLSCREEN COVER ---

}

// --- NEUE MODIFIKATION: Helper-Funktion ---

**private** **func** presentShareSheetForCurrentProgram() {

**guard** **let** program = viewModel.programs.first **else** { **return** }

shareItems = makeProgramShareItems(program: program)

showShareSheet = **true**

}

// --- ENDE NEUE MODIFIKATION ---

}

// **MARK: - Programm-Wechsel-Ansicht (STARK MODIFIZIERT)**

**struct** ProgramSwitcherView: View {

@EnvironmentObject **var** viewModel: TrainingViewModel

@Binding **var** isPresented: Bool

@Binding **var** showAddProgramSheet: Bool // Um das Add-Sheet zu öffnen

// --- NEU ---

// Helfer, um die Programme aufzuteilen, wie vom Nutzer gewünscht

**private** **var** activeProgram: ProgramData? {

viewModel.programs.first

}

**private** **var** otherPrograms: [ProgramData] {

Array(viewModel.programs.dropFirst())

}

// --- ENDE NEU ---

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

ZStack {

Color.black.ignoresSafeArea()

VStack(alignment: .leading, spacing: 0) {

// Header

HStack {

Button {

// "Programm hinzufügen"

// Schließe dieses Sheet und öffne das "Add Program" Sheet

isPresented = **false**

// Kleine Verzögerung, damit der Übergang sauber ist

DispatchQueue.main.asyncAfter(deadline: .now() + 0.1) {

showAddProgramSheet = **true**

}

} label: {

Text("Programm hinzufügen")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white)

.padding(.horizontal, 16)

.padding(.vertical, 10)

.background(

RoundedRectangle(cornerRadius: 12)

.fill(Color.white.opacity(0.15))

)

}

Spacer()

Button {

isPresented = **false** // Schließen

} label: {

ZStack {

Circle()

.fill(Color.white.opacity(0.15))

.frame(width: 44, height: 44)

Image(systemName: "xmark")

.font(.system(size: 16, weight: .semibold))

.foregroundColor(.white)

}

}

}

.padding(.horizontal, 20)

.padding(.top, 16)

.padding(.bottom, 20)

// Inhalt (MODIFIZIERT)

ScrollView {

VStack(alignment: .leading, spacing: 12) {

// --- Aktives Programm ---

Text("Aktives Programm")

.font(.system(size: 18, weight: .bold))

.foregroundColor(.white)

.padding(.horizontal, 20)

.padding(.top, 16)

**if** **let** program = activeProgram {

ProgramSwitcherRow(

program: program,

icon: "figure.strengthtraining.traditional", // Platzhalter-Icon

iconBgColor: Color(red: 0.95, green: 0.87, blue: 0.7),

iconFgColor: Color(red: 0.8, green: 0.6, blue: 0.3)

) {

// Aktion: Wähle dieses Programm

// Da es bereits aktiv ist, schließe einfach

isPresented = **false**

}

} **else** {

// Platzhalter, falls kein Programm existiert

Text("Kein aktives Programm.")

.foregroundColor(.white.opacity(0.5))

.padding(.horizontal, 40) // Einrücken

}

// Ad-hoc-Routinen (wie im Screenshot)

ProgramSwitcherRow(

title: "Ad-hoc-Routinen",

subtitle: "Keine Routinen", // **TODO: Logik hierfür**

icon: "list.clipboard",

iconBgColor: .pink.opacity(0.3),

iconFgColor: .pink

) {

// Aktion: Zeige Ad-hoc-Routinen

// (Noch nicht implementiert)

}

// --- Andere Programme ---

**if** !otherPrograms.isEmpty {

Text("Andere Programme")

.font(.system(size: 18, weight: .bold))

.foregroundColor(.white)

.padding(.horizontal, 20)

.padding(.top, 24) // Mehr Abstand

ForEach(otherPrograms) { program **in**

ProgramSwitcherRow(

program: program,

icon: "figure.strengthtraining.functional", // Platzhalter-Icon

iconBgColor: Color(red: 0.7, green: 0.7, blue: 0.9), // Andeere Farbe

iconFgColor: Color(red: 0.3, green: 0.3, blue: 0.6)

) {

// Aktion: Setze dieses Programm als aktiv

viewModel.setActiveProgram(id: program.id)

isPresented = **false**

}

}

}

}

.padding(.top, 16)

.padding(.bottom, 40) // Platz unten

}

}

}

}

}

// Eigene Row-View für den Programm-Switcher (NEU)

**struct** ProgramSwitcherRow: View {

**let** title: String

**let** subtitle: String

**let** icon: String

**let** iconBgColor: Color

**let** iconFgColor: Color

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

// Initialisierer für ein ProgramData-Objekt

**init**(program: ProgramData, icon: String, iconBgColor: Color, iconFgColor: Color, action: **@escaping** () -> Void) {

**self**.title = program.name

**let** routineCount = program.routines.count

**let** exerciseCount = program.routines.reduce(0) { $0 + $1.exercises.count }

**let** routineText = routineCount == 1 ? "Routine" : "Routinen"

**let** exerciseText = exerciseCount == 1 ? "Übung" : "Übungen"

**self**.subtitle = "\(routineCount) \(routineText), \(exerciseCount) \(exerciseText)"

**self**.icon = icon

**self**.iconBgColor = iconBgColor

**self**.iconFgColor = iconFgColor

**self**.action = action

}

// Initialisierer für manuelle Titel (wie Ad-hoc)

**init**(title: String, subtitle: String, icon: String, iconBgColor: Color, iconFgColor: Color, action: **@escaping** () -> Void) {

**self**.title = title

**self**.subtitle = subtitle

**self**.icon = icon

**self**.iconBgColor = iconBgColor

**self**.iconFgColor = iconFgColor

**self**.action = action

}

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

Button(action: action) {

HStack(spacing: 16) {

// Icon

ZStack {

Circle()

.fill(iconBgColor)

.frame(width: 60, height: 60)

Image(systemName: icon)

.font(.system(size: 28))

.foregroundColor(iconFgColor)

}

// Text

VStack(alignment: .leading, spacing: 4) {

Text(title)

.font(.system(size: 18, weight: .semibold))

.foregroundColor(.white)

Text(subtitle)

.font(.system(size: 15))

.foregroundColor(.white.opacity(0.6))

}

Spacer()

Image(systemName: "chevron.right")

.font(.system(size: 16, weight: .semibold))

.foregroundColor(.white.opacity(0.3))

}

.padding(.horizontal, 20)

.padding(.vertical, 12)

.background(

RoundedRectangle(cornerRadius: 16)

.fill(Color.white.opacity(0.08))

)

.padding(.horizontal, 20)

}

.buttonStyle(.plain)

}

}

// --- MODIFICATION ---

// This is the new RoutineDetailSheet.

// It replaces the old TrainingDetailSheet.

// It takes a specific RoutineData object and handles saving changes back to the ViewModel.

// **MARK: - Routine Detail Sheet (MODIFIED)**

**struct** RoutineDetailSheet: View {

@EnvironmentObject **var** viewModel: TrainingViewModel

@Environment(\.dismiss) **var** dismiss

**let** programId: UUID

**var** routine: RoutineData // Die spezifische Routine

@State **private** **var** showMenu = **false**

@State **private** **var** showExerciseList = **false**

// @State private var showReorderView = false // <-- MODIFIKATION: Entfernt

// State für die Übungen \*nur\* dieser Routine

@State **private** **var** exercises: [AddedExercise]

**init**(programId: UUID, routine: RoutineData) {

**self**.programId = programId

**self**.routine = routine

// Initialisiert mit den Übungen der übergebenen Routine

\_exercises = State(initialValue: routine.exercises)

}

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

ZStack {

Color.black.ignoresSafeArea()

VStack(spacing: 0) {

// Header

HStack {

headerCircle("xmark") { dismiss() }

Spacer()

headerCircle("ellipsis") { showMenu = **true** }

}

.padding(.horizontal, 20)

.padding(.top, 16)

.padding(.bottom, 20)

ScrollView {

VStack(alignment: .leading, spacing: 24) {

// Titel ist jetzt der Routinenname

Text(routine.name)

.font(.system(size: 42, weight: .bold)).foregroundColor(.white)

.padding(.horizontal, 20)

// --- MODIFIKATION: VStack durch List ersetzt ---

List {

// Zeigt nur die Übungen aus @State exercises an

ForEach(exercises) { exercise **in**

TrainingExerciseRow(addedExercise: exercise)

.listRowBackground(Color.black)

.listRowInsets(EdgeInsets())

.listRowSeparator(.hidden)

}

.onMove { from, to **in**

// Ermöglicht direktes Neuanordnen

exercises.move(fromOffsets: from, toOffset: to)

}

}

.listStyle(.plain)

// Höhe anpassen an Inhalt. 92 = 60 (Icon) + 2\*16 (V-Padding in Row)

.frame(height: CGFloat(exercises.count) \* 92)

// --- ENDE MODIFIKATION ---

// Button zum Hinzufügen von Übungen zu dieser Routine

Button { showExerciseList = **true** } label: {

Text("Hinzufügen +")

.font(.system(size: 16, weight: .semibold)).foregroundColor(.pink)

.padding(.horizontal, 20).padding(.vertical, 10)

.background(RoundedRectangle(cornerRadius: 14).fill(Color.white.opacity(0.08)))

}

.padding(.horizontal, 20)

}

.padding(.bottom, 90) // <-- MODIFIZIERT: von 120 auf 90 reduziert

}

} // End of V1

// "Training starten" Button

VStack {

Spacer()

Button { } label: {

Text("Training starten")

.font(.system(size: 18, weight: .bold)).foregroundColor(.white)

.frame(maxWidth: .infinity).frame(height: 56)

.background(RoundedRectangle(cornerRadius: 28).fill(Color.pink))

}

.padding(.horizontal, 20)

.padding(.bottom, 20) // <-- MODIFIZIERT: von 40 auf 20 reduziert

} // End of V2

// Overflow Menü

**if** showMenu {

Color.black.opacity(0.5).ignoresSafeArea().onTapGesture { showMenu = **false** }

VStack {

HStack {

Spacer()

// --- MODIFIKATION: onReorderExercises entfernt ---

TrainingDetailMenu(

onAddExercises: { showMenu = **false** ; showExerciseList = **true** }

// onReorderExercises: { showMenu = false ; showReorderView = true }

)

.padding(.trailing, 20).padding(.top, 90)

}

Spacer()

}

}

}

// Sheet zum Hinzufügen von Übungen

.sheet(isPresented: $showExerciseList) {

ExerciseListView(onExercisesAdded: { newExercises **in**

// Fügt Übungen zur lokalen @State-Variable hinzu

exercises.append(contentsOf: newExercises.map { AddedExercise(exercise: $0) })

})

}

// --- MODIFIKATION: fullScreenCover für ReorderView entfernt ---

// .fullScreenCover(isPresented: $showReorderView) { ... }

.onDisappear {

// Wenn das Sheet geschlossen wird, speichere die Änderungen im ViewModel

saveChanges()

}

}

/// Speichert die lokale `exercises`-Liste zurück in das ViewModel.

**private** **func** saveChanges() {

// Finde das Programm im ViewModel

**guard** **let** programIndex = viewModel.programs.firstIndex(where: { $0.id == programId }) **else** { **return** }

// Finde die Routine in diesem Programm

**guard** **let** routineIndex = viewModel.programs[programIndex].routines.firstIndex(where: { $0.id == routine.id }) **else** { **return** }

// Aktualisiere die Übungsliste dieser spezifischen Routine im ViewModel

viewModel.programs[programIndex].routines[routineIndex].exercises = exercises

}

/// Helfer-View für Header-Buttons

@ViewBuilder **private** **func** headerCircle(\_ system: String, action: **@escaping** () -> Void) -> **some** View {

Button(action: action) {

ZStack {

Circle().fill(Color.white.opacity(0.15)).frame(width: 50, height: 50)

Image(systemName: system).font(.system(size: 18, weight: .semibold)).foregroundColor(.white)

}

}

}

}

// --- END MODIFICATION ---

// **MARK: - Program Edit View**

**struct** ProgramEditView: View {

@EnvironmentObject **var** viewModel: TrainingViewModel

@Environment(\.dismiss) **var** dismiss

**let** programId: UUID

@State **private** **var** name: String

@State **private** **var** routines: [RoutineData]

@State **private** **var** showCreateRoutineDialog = **false**

@State **private** **var** showExerciseList = **false**

@State **private** **var** showPlusMenu = **false**

@State **private** **var** showEllipsisMenu = **false**

@State **private** **var** selectedRoutineIndex: Int = 0

@State **private** **var** showRenameDialog = **false**

@State **private** **var** renameText: String = ""

@State **private** **var** showRoutineMenu = **false**

@State **private** **var** showRenameRoutineDialog = **false**

@State **private** **var** routineRenameText: String = ""

// Sheet "Sätze pro Muskelgruppe"

@State **private** **var** showEditSets = **false**

@State **private** **var** editSetsRoutineIndex = 0

// Share

@State **private** **var** showShareSheet = **false**

@State **private** **var** shareItems: [**Any**] = []

// Delete

@State **private** **var** showDeleteAlert = **false**

/// Controls presentation of the dialog to duplicate a routine. When true, a dialog appears

/// asking the user to provide a name for the new routine.

@State **private** **var** showDuplicateRoutineDialog = **false**

/// The provisional name for a duplicated routine. This text is bound to the text field in

/// `DuplicateRoutineDialog` and is used when creating the new routine upon confirmation.

@State **private** **var** duplicateRoutineName: String = ""

/// Controls presentation of the change activity sheet when editing a routine. See `RoutineOverflowMenu.onChangeActivity`.

@State **private** **var** showChangeActivityDialog = **false**

**init**(program: ProgramData) {

**self**.programId = program.id

\_name = State(initialValue: program.name)

\_routines = State(initialValue: program.routines)

}

**var** totalExercises: Int { routines.reduce(0) { $0 + $1.exercises.count } }

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

ZStack {

Color.black.ignoresSafeArea()

VStack(spacing: 0) {

HStack(spacing: 16) {

headerCircle("chevron.left") { dismiss() }

Spacer()

Text(name).font(.system(size: 18, weight: .semibold)).foregroundColor(.white)

Spacer()

HStack(spacing: 12) {

headerRect("plus") { showPlusMenu = **true** }

headerRect("ellipsis") { showEllipsisMenu = **true** }

}

headerCircle("checkmark") {

viewModel.updateRoutines(for: programId, routines: routines)

viewModel.updateProgramName(id: programId, newName: name)

dismiss()

}

}

.padding(.horizontal, 20).padding(.top, 16).padding(.bottom, 12)

ScrollView {

VStack(alignment: .leading, spacing: 16) {

Text(name) // Changed from "Training"

.font(.system(size: 42, weight: .bold)).foregroundColor(.white)

.padding(.horizontal, 20)

Text("\(totalExercises) Übungen")

.font(.system(size: 15)).foregroundColor(.white.opacity(0.6))

.padding(.horizontal, 20).padding(.bottom, 8)

// --- MODIFIKATION: Äußere ForEach nutzt jetzt $routines ---

ForEach($routines) { $routine **in** // Binding $routine

VStack(alignment: .leading, spacing: 16) {

HStack {

Text(routine.name) // Verwende routine.name

.font(.system(size: 28, weight: .bold)).foregroundColor(.white)

Button {

// Finde den Index für das Menü

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

selectedRoutineIndex = index

withAnimation { showRoutineMenu = **true** }

}

} label: {

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

.font(.system(size: 24)).foregroundColor(.pink)

}

}

Text("\(routine.exercises.count) Übungen") // Verwende routine.exercises

.font(.system(size: 15)).foregroundColor(.white.opacity(0.6))

**if** routine.exercises.isEmpty {

HStack {

Text("Keine Übungen")

.font(.system(size: 17)).foregroundColor(.white.opacity(0.5))

Spacer()

}

.padding(.horizontal, 20).padding(.vertical, 16)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.08)))

} **else** {

// --- MODIFIKATION: VStack durch List ersetzt ---

List {

ForEach(routine.exercises) { addedExercise **in**

AddedExerciseRow(addedExercise: addedExercise)

.listRowBackground(Color.clear) // Row-Hintergrund transparent

.listRowInsets(EdgeInsets()) // Keine Standard-Insets

.listRowSeparator(.hidden) // Keine Trennlinien

}

.onMove { from, to **in**

// .onMove funktioniert jetzt mit dem $routine binding

routine.exercises.move(fromOffsets: from, toOffset: to)

}

}

.listStyle(.plain)

// Höhe berechnen: 84 = 60 (Icon) + 2\*12 (V-Padding in Row)

.frame(height: CGFloat(routine.exercises.count) \* 84)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.08))) // Hintergrund auf die List

// --- ENDE MODIFIKATION ---

}

Button {

// Finde den Index für das Hinzufügen-Sheet

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

selectedRoutineIndex = index

showExerciseList = **true**

}

} label: {

Text("Hinzufügen +")

.font(.system(size: 16, weight: .semibold)).foregroundColor(.pink)

.padding(.horizontal, 20).padding(.vertical, 10)

.background(RoundedRectangle(cornerRadius: 14).fill(Color.white.opacity(0.08)))

}

}

.padding(.horizontal, 20).padding(.top, 8)

}

// --- ENDE MODIFIKATION ---

VStack(spacing: 16) {

Button { showCreateRoutineDialog = **true** } label: {

HStack(spacing: 16) {

ZStack {

RoundedRectangle(cornerRadius: 14).fill(Color.pink.opacity(0.15)).frame(width: 48, height: 48)

Image(systemName: "list.clipboard").font(.system(size: 24)).foregroundColor(.pink)

}

Text("Neue Routine schaffen").font(.system(size: 18, weight: .semibold)).foregroundColor(.pink)

Spacer()

}

.padding(20)

.background(RoundedRectangle(cornerRadius: 20).fill(Color.cardBackground))

}

}

.padding(.horizontal, 20).padding(.top, 8)

Spacer(minLength: 60)

}

}

}

// Menüs

**if** showPlusMenu {

dimBG { showPlusMenu = **false** }

anchoredTopRight {

PlusMenuView(

onNewRoutine: { showPlusMenu = **false** ; showCreateRoutineDialog = **true** },

onAddWeek: { showPlusMenu = **false** }

)

.padding(.trailing, 76).padding(.top, 90)

}

}

**if** showEllipsisMenu {

dimBG { showEllipsisMenu = **false** }

anchoredTopRight {

EllipsisMenuView(

onChangeName: {

showEllipsisMenu = **false**

renameText = name

showRenameDialog = **true**

},

onShareProgram: {

showEllipsisMenu = **false**

presentShareSheetForCurrentProgram()

},

onEditSets: {

showEllipsisMenu = **false**

editSetsRoutineIndex = ensureRoutineForEditSets()

showEditSets = **true**

},

// onDuplicate: { showEllipsisMenu = false }, // <-- ENTFERNT

onSetDefault: { showEllipsisMenu = **false** },

onChangeAppearance: { showEllipsisMenu = **false** },

onArchive: { showEllipsisMenu = **false** },

onResetProgress: { showEllipsisMenu = **false** },

onDeleteProgram: {

showEllipsisMenu = **false**

showDeleteAlert = **true**

}

)

.padding(.trailing, 20).padding(.top, 90)

}

}

**if** showRoutineMenu {

dimBG { withAnimation { showRoutineMenu = **false** } }

anchoredTopRight {

RoutineOverflowMenu(

onRename: {

showRoutineMenu = **false**

routineRenameText = routines[**safe**: selectedRoutineIndex]?.name ?? ""

showRenameRoutineDialog = **true**

},

onDuplicate: {

// Prepare default name for the duplicated routine and present a dialog instead of immediately duplicating

**if** **let** idx = routines.indices.contains(selectedRoutineIndex) ? selectedRoutineIndex : **nil** {

**let** src = routines[idx]

**let** suffix = " Kopie"

// If the source name already ends with " Kopie", reuse it; otherwise append the suffix

duplicateRoutineName = src.name.hasSuffix(suffix) ? src.name : (src.name + suffix)

} **else** {

duplicateRoutineName = ""

}

withAnimation { showRoutineMenu = **false** }

showDuplicateRoutineDialog = **true**

},

onChangeActivity: {

// Hide the routine menu and present the activity picker sheet

withAnimation { showRoutineMenu = **false** }

showChangeActivityDialog = **true**

},

onDelete: {

**if** routines.indices.contains(selectedRoutineIndex) { routines.remove(at: selectedRoutineIndex) }

withAnimation { showRoutineMenu = **false** }

}

)

.padding(.trailing, 24).padding(.top, 140)

}

.transition(AnyTransition.opacity.combined(with: .move(edge: .top)))

}

**if** showCreateRoutineDialog {

dimBG { showCreateRoutineDialog = **false** }

CreateRoutineDialog(isPresented: $showCreateRoutineDialog) { routines.append(RoutineData(name: $0)) }

}

**if** showRenameDialog {

Color.black.opacity(0.45).ignoresSafeArea()

RenameProgramDialog(

isPresented: $showRenameDialog,

text: $renameText,

title: "Name ändern",

subtitle: "Gib einen Namen für den Programm ein.",

onConfirm: {

**let** trimmed = renameText.trimmingCharacters(in: .whitespacesAndNewlines)

**guard** !trimmed.isEmpty **else** { **return** }

name = trimmed

viewModel.updateProgramName(id: programId, newName: trimmed)

}

)

.transition(AnyTransition.scale.combined(with: .opacity))

}

**if** showRenameRoutineDialog {

Color.black.opacity(0.45).ignoresSafeArea()

RenameProgramDialog(

isPresented: $showRenameRoutineDialog,

text: $routineRenameText,

title: "Name ändern",

subtitle: "Gib einen Namen für die Routine ein.",

onConfirm: {

**let** trimmed = routineRenameText.trimmingCharacters(in: .whitespacesAndNewlines)

**guard** !trimmed.isEmpty, routines.indices.contains(selectedRoutineIndex) **else** { **return** }

routines[selectedRoutineIndex].name = trimmed

}

)

.transition(AnyTransition.scale.combined(with: .opacity))

}

// Duplicate routine dialog overlay

**if** showDuplicateRoutineDialog {

Color.black.opacity(0.45).ignoresSafeArea()

DuplicateRoutineDialog(

isPresented: $showDuplicateRoutineDialog,

text: $duplicateRoutineName,

onConfirm: {

// Duplicate the selected routine with the provided name

**let** trimmed = duplicateRoutineName.trimmingCharacters(in: .whitespacesAndNewlines)

**guard** !trimmed.isEmpty **else** { **return** }

**if** routines.indices.contains(selectedRoutineIndex) {

**let** idx = selectedRoutineIndex

**let** src = routines[idx]

**let** newExercises = src.exercises.map { e **in**

AddedExercise(

exercise: e.exercise,

weight: e.weight,

sets: e.sets,

reps: e.reps

)

}

**let** clone = RoutineData(

name: trimmed,

exercises: newExercises,

activityType: src.activityType

)

routines.insert(clone, at: idx + 1)

}

}

)

.transition(AnyTransition.scale.combined(with: .opacity))

}

}

.sheet(isPresented: $showExerciseList) {

ExerciseListView(onExercisesAdded: { exs **in**

**for** ex **in** exs {

routines[selectedRoutineIndex].exercises.append(AddedExercise(exercise: ex))

}

})

}

.sheet(isPresented: $showEditSets) {

**if** routines.indices.contains(editSetsRoutineIndex) {

EditSetsPerMuscleView(routine: $routines[editSetsRoutineIndex], isPresented: $showEditSets)

}

}

// Present the activity type picker for a routine. The sheet appears at half height by default

// and can be dragged to full screen. The binding ensures the routine's activity type updates

// automatically when the user makes a selection.

.sheet(isPresented: $showChangeActivityDialog) {

// Guard against an out-of-range index. SwiftUI evaluates the sheet's content lazily,

// so index validity is rechecked when the sheet is presented.

**if** routines.indices.contains(selectedRoutineIndex) {

ChangeActivityDialog(

isPresented: $showChangeActivityDialog,

selectedActivity: $routines[selectedRoutineIndex].activityType

) { newType **in**

// The binding above updates the routine's activityType automatically.

// Additional side effects can be placed here if needed.

}

.presentationDetents([.fraction(0.5), .large])

.presentationDragIndicator(.visible)

}

}

#if os(iOS)

.sheet(isPresented: $showShareSheet) {

ShareSheet(items: shareItems, subject: "Programm: \(name)")

}

#endif

.alert("Programm löschen?", isPresented: $showDeleteAlert) {

Button("Löschen", role: .destructive) {

viewModel.deleteProgram(id: programId)

dismiss()

}

Button("Abbrechen", role: .cancel) {}

} message: {

Text("Das Programm „\(name)“ wird dauerhaft entfernt.")

}

}

**private** **func** duplicateRoutine(at idx: Int) {

**guard** routines.indices.contains(idx) **else** { **return** }

**let** src = routines[idx]

**let** newExercises = src.exercises.map { e **in**

AddedExercise(exercise: e.exercise, weight: e.weight, sets: e.sets, reps: e.reps)

}

**let** nameSuffix = " Kopie"

**let** newName = src.name.hasSuffix(nameSuffix) ? src.name : (src.name + nameSuffix)

// Preserve the activity type when duplicating a routine

**let** clone = RoutineData(name: newName, exercises: newExercises, activityType: src.activityType)

routines.insert(clone, at: idx + 1)

}

**private** **func** ensureRoutineForEditSets() -> Int {

**if** routines.isEmpty {

**let** newName = "Routine 1"

routines.append(RoutineData(name: newName))

selectedRoutineIndex = 0

**return** 0

}

**if** routines.indices.contains(selectedRoutineIndex) {

**return** selectedRoutineIndex

}

selectedRoutineIndex = 0

**return** 0

}

**private** **func** presentShareSheetForCurrentProgram() {

**let** current = ProgramData(name: name, routines: routines)

shareItems = makeProgramShareItems(program: current)

showShareSheet = **true**

}

@ViewBuilder **private** **func** headerCircle(\_ system: String, action: **@escaping** () -> Void) -> **some** View {

Button(action: action) {

ZStack {

Circle().fill(Color.white.opacity(0.15)).frame(width: 50, height: 50)

Image(systemName: system).font(.system(size: 18, weight: .semibold)).foregroundColor(.white)

}

}

}

@ViewBuilder **private** **func** headerRect(\_ system: String, action: **@escaping** () -> Void) -> **some** View {

Button(action: action) {

Image(systemName: system).font(.system(size: 18, weight: .semibold)).foregroundColor(.white)

.frame(width: 50, height: 50)

.background(RoundedRectangle(cornerRadius: 25).fill(Color.white.opacity(0.15)))

}

}

@ViewBuilder **private** **func** dimBG(onTap: **@escaping** () -> Void) -> **some** View {

Color.black.opacity(0.5).ignoresSafeArea().onTapGesture(perform: onTap)

}

@ViewBuilder **private** **func** anchoredTopRight<Content: View>(@ViewBuilder \_ content: () -> Content) -> **some** View {

VStack { HStack { Spacer(); content() }; Spacer() }

}

}

// **MARK: - Reorder Exercises View**

**struct** ReorderExercisesView: View {

@Environment(\.dismiss) **var** dismiss

@Binding **var** exercises: [AddedExercise]

@State **private** **var** editableExercises: [AddedExercise] = []

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

ZStack {

Color.black.ignoresSafeArea()

VStack(spacing: 0) {

HStack {

headerCircle("xmark") { dismiss() }

Spacer()

headerCircle("checkmark") { exercises = editableExercises; dismiss() }

}

.padding(.horizontal, 20).padding(.top, 16).padding(.bottom, 20)

Text("Training")

.font(.system(size: 42, weight: .bold)).foregroundColor(.white)

.frame(maxWidth: .infinity, alignment: .leading)

.padding(.horizontal, 20).padding(.bottom, 20)

List {

ForEach(editableExercises) { exercise **in**

ReorderExerciseRow(addedExercise: exercise)

.listRowBackground(Color.black)

.listRowInsets(EdgeInsets())

.listRowSeparator(.hidden)

}

.onMove { from, to **in** editableExercises.move(fromOffsets: from, toOffset: to) }

Button { } label: {

Text("Hinzufügen +")

.font(.system(size: 16, weight: .semibold)).foregroundColor(.pink)

.padding(.horizontal, 20).padding(.vertical, 10)

.background(RoundedRectangle(cornerRadius: 14).fill(Color.white.opacity(0.08)))

}

.listRowBackground(Color.black)

.listRowInsets(EdgeInsets(top: 20, leading: 20, bottom: 0, trailing: 20))

.listRowSeparator(.hidden)

}

.listStyle(.plain)

.scrollContentBackground(.hidden)

.background(Color.black)

.environment(\.editMode, .constant(.active))

}

}

.onAppear { editableExercises = exercises }

}

@ViewBuilder **private** **func** headerCircle(\_ system: String, action: **@escaping** () -> Void) -> **some** View {

Button(action: action) {

ZStack { Circle().fill(Color.white.opacity(0.15)).frame(width: 50, height: 50)

Image(systemName: system).font(.system(size: 18, weight: .semibold)).foregroundColor(.white) }

}

}

}

// **MARK: - EditSetsPerMuscleView**

**struct** EditSetsPerMuscleView: View {

@Binding **var** routine: RoutineData

@Binding **var** isPresented: Bool

@State **private** **var** selectedMuscle: String = "Brust"

@State **private** **var** showExerciseList = **false**

**private** **var** filteredExercises: [AddedExercise] {

routine.exercises.filter { $0.exercise.category == selectedMuscle }

}

**private** **var** totalSets: Int { filteredExercises.reduce(0) { $0 + $1.sets } }

**private** **var** primarySets: Int { totalSets }

**private** **var** secondarySets: Int { 0 }

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

ZStack {

Color.black.ignoresSafeArea()

VStack(spacing: 0) {

// Header

ZStack {

Text("Sätze pro Muskelgruppe")

.font(.system(size: 18, weight: .semibold)).foregroundColor(.white)

HStack {

Spacer()

Button {

isPresented = **false**

} label: {

ZStack {

Circle().fill(Color.pink).frame(width: 50, height: 50)

Image(systemName: "checkmark")

.font(.system(size: 18, weight: .semibold)).foregroundColor(.white)

}

}

}

.padding(.horizontal, 16)

}

.padding(.top, 12).padding(.bottom, 12)

ScrollView {

VStack(alignment: .leading, spacing: 20) {

Text(routine.name)

.font(.system(size: 28, weight: .bold)).foregroundColor(.white)

.padding(.horizontal, 20).padding(.top, 8)

HStack {

Text(filteredExercises.isEmpty ? "Keine „\(selectedMuscle)“-Übungen" :

"\(filteredExercises.count) „\(selectedMuscle)“-Übung(en)")

.font(.system(size: 16)).foregroundColor(.white.opacity(0.9))

.padding(.horizontal, 16).padding(.vertical, 12)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.10)))

Spacer()

}

.padding(.horizontal, 20)

Button { showExerciseList = **true** } label: {

Text("Hinzufügen +")

.font(.system(size: 15, weight: .semibold)).foregroundColor(.pink)

.padding(.horizontal, 14).padding(.vertical, 8)

.background(Capsule().fill(Color.white.opacity(0.10)))

}

.padding(.horizontal, 20)

Spacer(minLength: 20)

VStack(alignment: .leading, spacing: 16) {

HStack {

Button {

// **TODO: echten Picker/Menu einbauen**

} label: {

HStack(spacing: 6) {

Text(selectedMuscle)

.font(.system(size: 16, weight: .semibold))

Image(systemName: "chevron.down").font(.system(size: 12, weight: .bold))

}

.foregroundColor(.white)

.padding(.horizontal, 12).padding(.vertical, 8)

.background(Capsule().fill(Color.pink))

}

Spacer()

}

HStack {

Text("\(totalSets) Sätze - \(primarySets) primäre, \(secondarySets) sekundäre")

.font(.system(size: 16)).foregroundColor(.white)

.padding(.horizontal, 16).padding(.vertical, 12)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.10)))

Spacer()

}

HStack {

Text("Wöchentliches Ziel").font(.system(size: 16)).foregroundColor(.white)

.padding(.leading, 16)

Spacer()

Text("Nicht festgelegt").font(.system(size: 16)).foregroundColor(.white.opacity(0.75))

.padding(.trailing, 16)

}

.frame(height: 48)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.10)))

}

.padding(20)

.background(

RoundedRectangle(cornerRadius: 26).fill(Color.cardBackground)

.overlay(RoundedRectangle(cornerRadius: 26).stroke(Color.white.opacity(0.05), lineWidth: 1))

.shadow(color: .black.opacity(0.25), radius: 12, x: 0, y: 6)

)

.padding(.horizontal, 16)

.padding(.bottom, 40)

}

}

}

}

.sheet(isPresented: $showExerciseList) {

ExerciseListView(

onExercisesAdded: { exs **in**

exs.forEach { routine.exercises.append(AddedExercise(exercise: $0)) }

},

categoryFilter: selectedMuscle

)

}

}

}

// **MARK: - Exercise List View**

**struct** ExerciseListView: View {

@Environment(\.dismiss) **var** dismiss

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

@State **private** **var** selectedExercises: Set<UUID> = []

@State **private** **var** showExerciseMenu = **false**

@State **private** **var** showCreateExerciseDialog = **false**

@State **private** **var** createdExerciseName: String = ""

@State **private** **var** showExerciseDetailSheet = **false**

@State **private** **var** sortAscending = **true**

**let** onExercisesAdded: ([Exercise]) -> Void

**let** categoryFilter: String?

**init**(onExercisesAdded: **@escaping** ([Exercise]) -> Void, categoryFilter: String? = **nil**) {

**self**.onExercisesAdded = onExercisesAdded

**self**.categoryFilter = categoryFilter

}

// Beispiel-Daten

**let** recentExercises = [

Exercise(name: "Beinpresse sitzend", category: "Beine", type: "Maschine", iconName: "figure.strengthtraining.traditional"),

Exercise(name: "Latzug (Maschine)", category: "Rücken", type: "Maschine", iconName: "figure.pullup")

]

**let** exercises = [

Exercise(name: "Ab Wheel", category: "Bauch", type: "andere", iconName: "figure.core.training"),

Exercise(name: "Ab-Crunch an der Maschine", category: "Bauch", type: "Maschine", iconName: "figure.strengthtraining.traditional"),

Exercise(name: "Anfersen", category: "Beine", type: "Körper", iconName: "figure.run"),

Exercise(name: "Arnold Press", category: "Schultern", type: "Kurzhantel", iconName: "dumbbell"),

Exercise(name: "Bankdrücken Langhantel", category: "Brust", type: "Langhantel", iconName: "dumbbell"),

Exercise(name: "Schrägbank Kurzhantel", category: "Brust", type: "Kurzhantel", iconName: "dumbbell")

]

**var** filteredSortedExercises: [Exercise] {

**let** pre = exercises.filter { ex **in**

**guard** **let** cat = categoryFilter **else** { **return** **true** }

**return** ex.category == cat

}

**let** list = searchText.isEmpty ? pre : pre.filter {

$0.name.localizedCaseInsensitiveContains(searchText) ||

$0.category.localizedCaseInsensitiveContains(searchText) ||

$0.type.localizedCaseInsensitiveContains(searchText)

}

**return** list.sorted { sortAscending ? $0.name < $1.name : $0.name > $1.name }

}

**var** grouped: [(String, [Exercise])] {

**let** dict = Dictionary(grouping: filteredSortedExercises) { String($0.name.prefix(1)).uppercased() }

**return** dict.keys.sorted().map { ($0, dict[$0]!.sorted { $0.name < $1.name }) }

}

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

ZStack {

Color.black.ignoresSafeArea()

VStack(spacing: 0) {

HStack(spacing: 16) {

headerCircle("xmark") { dismiss() }

Text("Übungen").font(.system(size: 20, weight: .semibold)).foregroundColor(.white)

Spacer()

headerCircle("arrow.up.arrow.down") { withAnimation(.easeInOut(duration: 0.15)) { sortAscending.toggle() } }

Button {

**guard** !selectedExercises.isEmpty **else** { **return** }

**let** selected = (recentExercises + exercises).filter { selectedExercises.contains($0.id) }

onExercisesAdded(selected)

dismiss()

} label: {

Text("Hinzufügen")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(selectedExercises.isEmpty ? .white.opacity(0.45) : .black)

.padding(.horizontal, 16).frame(height: 44)

.background(Capsule().fill(selectedExercises.isEmpty ? Color.white.opacity(0.18) : Color.white))

}

.disabled(selectedExercises.isEmpty)

}

.padding(.horizontal, 20).padding(.top, 16).padding(.bottom, 12)

ScrollView {

VStack(alignment: .leading, spacing: 24) {

**if** categoryFilter == **nil** {

VStack(alignment: .leading, spacing: 12) {

Text("Aktuelle")

.font(.system(size: 24, weight: .bold)).foregroundColor(.white)

.padding(.horizontal, 20)

ScrollView(.horizontal, showsIndicators: **false**) {

HStack(spacing: 20) {

ForEach(recentExercises) { ex **in**

VStack(spacing: 8) {

ExerciseAvatar(title: ex.name, systemIcon: ex.iconName, size: 76, bg: Color(red: 0.96, green: 0.92, blue: 0.80))

.overlay(alignment: .bottomTrailing) {

**if** selectedExercises.contains(ex.id) {

ZStack {

Circle().fill(Color.pink).frame(width: 26, height: 26)

Image(systemName: "checkmark").font(.system(size: 12, weight: .bold)).foregroundColor(.white)

}

.offset(x: 6, y: 6)

}

}

Text(ex.name)

.font(.system(size: 14)).foregroundColor(.white)

.frame(width: 120).multilineTextAlignment(.center).lineLimit(2)

}

.onTapGesture { toggleSelect(ex.id) }

}

}

.padding(.horizontal, 20)

}

}

}

VStack(alignment: .leading, spacing: 8) {

HStack(spacing: 8) {

Text("Alle Übungen" + (categoryFilter != **nil** ? " – \(categoryFilter!)" : ""))

.font(.system(size: 24, weight: .bold)).foregroundColor(.white)

Image(systemName: sortAscending ? "chevron.down" : "chevron.up")

.foregroundColor(.white.opacity(0.6)).font(.system(size: 14, weight: .semibold))

}

.padding(.horizontal, 20)

ForEach(grouped, id: \.0) { section **in**

Text(section.0)

.font(.system(size: 18, weight: .bold)).foregroundColor(.white.opacity(0.55))

.padding(.horizontal, 20).padding(.top, 6).padding(.bottom, 8)

VStack(spacing: 0) {

ForEach(section.1) { exercise **in**

ExerciseRow(exercise: exercise, isSelected: selectedExercises.contains(exercise.id)) {

toggleSelect(exercise.id)

}

Divider().background(Color.white.opacity(0.08)).padding(.leading, 96)

}

}

}

}

}

.padding(.bottom, 100)

}

ZStack {

HStack(spacing: 16) {

Button { } label: {

Image(systemName: "line.3.horizontal").font(.system(size: 22)).foregroundColor(.white)

.frame(width: 50, height: 50)

}

HStack(spacing: 10) {

Image(systemName: "magnifyingglass").foregroundColor(.white.opacity(0.5))

TextField("Suchen", text: $searchText)

.foregroundColor(.white)

.textInputAutocapitalization(.none)

.disableAutocorrection(**true**)

.submitLabel(.search)

}

.padding(.horizontal, 16).frame(height: 50)

.background(Capsule().fill(Color.white.opacity(0.1)))

Button { showExerciseMenu = **true** } label: {

Image(systemName: "ellipsis").font(.system(size: 22)).foregroundColor(.white).frame(width: 50, height: 50)

}

}

.padding(.horizontal, 20).padding(.vertical, 12)

**if** !selectedExercises.isEmpty {

Text("\(selectedExercises.count) ausgewählt")

.font(.system(size: 14, weight: .semibold)).foregroundColor(.white)

.padding(.horizontal, 14).padding(.vertical, 8)

.background(Capsule().fill(Color.white.opacity(0.22)))

}

}

.background(Color.black)

}

**if** showExerciseMenu {

Color.black.opacity(0.5).ignoresSafeArea().onTapGesture { showExerciseMenu = **false** }

VStack { Spacer()

HStack { Spacer()

ExerciseListMenu(onCreateNewExercise: {

showExerciseMenu = **false** ; showCreateExerciseDialog = **true**

})

.padding(.trailing, 20).padding(.bottom, 100)

}

}

}

**if** showCreateExerciseDialog {

Color.black.opacity(0.5).ignoresSafeArea()

CreateExerciseDialog(isPresented: $showCreateExerciseDialog) {

createdExerciseName = $0; showExerciseDetailSheet = **true**

}

}

}

.fullScreenCover(isPresented: $showExerciseDetailSheet) {

ExerciseDetailSheet(exerciseName: createdExerciseName)

}

}

**private** **func** toggleSelect(\_ id: UUID) {

**if** selectedExercises.contains(id) { selectedExercises.remove(id) } **else** { selectedExercises.insert(id) }

}

@ViewBuilder **private** **func** headerCircle(\_ system: String, action: **@escaping** () -> Void) -> **some** View {

Button(action: action) {

ZStack { Circle().fill(Color.white.opacity(0.15)).frame(width: 50, height: 50)

Image(systemName: system).font(.system(size: 18, weight: .semibold)).foregroundColor(.white) }

}

}

}

// **MARK: - Kleine Views / Menüs / Rows**

// --- MODIFIKATION: "onReorderExercises" entfernt ---

**struct** TrainingDetailMenu: View {

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

// let onReorderExercises: () -> Void // <-- Entfernt

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

VStack(alignment: .leading, spacing: 0) {

MenuRow(icon: "plus", text: "Übungen hinzufügen", action: onAddExercises)

// Divider().background(Color.white.opacity(0.10)) // <-- Entfernt

// MenuRow(icon: "arrow.up.arrow.down", text: "Übungen neu\nanordnen", multiline: true , action: onReorderExercises) // <-- Entfernt

}

.background(RoundedRectangle(cornerRadius: 16).fill(Color(red: 0.2, green: 0.2, blue: 0.2)))

.frame(width: 280)

}

}

// --- ENDE MODIFIKATION ---

**struct** TrainingExerciseRow: View {

**let** addedExercise: AddedExercise

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

Button { } label: {

HStack(spacing: 16) {

ZStack {

Circle().fill(Color(red: 0.95, green: 0.87, blue: 0.7)).frame(width: 60, height: 60)

Image(systemName: addedExercise.exercise.iconName).font(.system(size: 28)).foregroundColor(Color(red: 0.8, green: 0.6, blue: 0.3))

}

VStack(alignment: .leading, spacing: 4) {

Text(addedExercise.exercise.name).font(.system(size: 17)).foregroundColor(.white)

Text("\(addedExercise.weight == "KG" ? "KG" : addedExercise.weight), \(addedExercise.sets) Sätze, \(addedExercise.reps) Wdh.")

.font(.system(size: 15)).foregroundColor(.white.opacity(0.5))

}

Spacer()

Image(systemName: "chevron.right").font(.system(size: 16, weight: .semibold)).foregroundColor(.white.opacity(0.3))

}

.padding(.horizontal, 20).padding(.vertical, 16).background(Color.black)

}

.buttonStyle(.plain)

}

}

**struct** ReorderExerciseRow: View {

**let** addedExercise: AddedExercise

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

HStack(spacing: 16) {

ZStack {

Circle().fill(Color(red: 0.95, green: 0.87, blue: 0.7)).frame(width: 60, height: 60)

Image(systemName: addedExercise.exercise.iconName).font(.system(size: 28)).foregroundColor(Color(red: 0.8, green: 0.6, blue: 0.3))

}

VStack(alignment: .leading, spacing: 4) {

Text(addedExercise.exercise.name).font(.system(size: 17)).foregroundColor(.white)

Text("\(addedExercise.weight == "KG" ? "KG" : addedExercise.weight), \(addedExercise.sets) Sätze, \(addedExercise.reps) Wdh.")

.font(.system(size: 15)).foregroundColor(.white.opacity(0.5))

}

Spacer()

}

.padding(.horizontal, 20).padding(.vertical, 16).background(Color.black)

}

}

**struct** AddedExerciseRow: View {

**let** addedExercise: AddedExercise

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

HStack(spacing: 16) {

ZStack {

Circle().fill(Color.cyan.opacity(0.3)).frame(width: 60, height: 60)

Image(systemName: addedExercise.exercise.iconName).font(.system(size: 28)).foregroundColor(.cyan)

}

VStack(alignment: .leading, spacing: 4) {

Text(addedExercise.exercise.name).font(.system(size: 17)).foregroundColor(.white)

Text("\(addedExercise.weight), \(addedExercise.sets) Sätze, \(addedExercise.reps) Wdh.")

.font(.system(size: 15)).foregroundColor(.white.opacity(0.5))

}

Spacer()

// --- MODIFIKATION: Button entfernt, nur noch Image als Indikator ---

Image(systemName: "line.3.horizontal")

.font(.system(size: 20))

.foregroundColor(.white.opacity(0.5))

// --- ENDE MODIFIKATION ---

}

.padding(.horizontal, 16).padding(.vertical, 12)

}

}

**struct** ExerciseRow: View {

**let** exercise: Exercise

**let** isSelected: Bool

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

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

Button(action: onTap) {

HStack(spacing: 16) {

ZStack(alignment: .bottomTrailing) {

ExerciseAvatar(title: exercise.name, systemIcon: exercise.iconName, size: 60, bg: Color(red: 0.96, green: 0.92, blue: 0.80))

**if** isSelected {

ZStack {

Circle().fill(Color.pink).frame(width: 24, height: 24)

Image(systemName: "checkmark").font(.system(size: 12, weight: .bold)).foregroundColor(.white)

}

.offset(x: 6, y: 6)

}

}

VStack(alignment: .leading, spacing: 4) {

Text(exercise.name).font(.system(size: 17)).foregroundColor(.white)

Text("\(exercise.category) — \(exercise.type)").font(.system(size: 15)).foregroundColor(.white.opacity(0.6))

}

Spacer()

Button { } label: { Image(systemName: "info.circle").font(.system(size: 22, weight: .semibold)).foregroundColor(.pink) }

}

.padding(.horizontal, 20).padding(.vertical, 12).background(Color.black)

}

.buttonStyle(.plain)

}

}

**struct** ExerciseListMenu: View {

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

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

Button(action: onCreateNewExercise) {

HStack(spacing: 16) {

Image(systemName: "plus").font(.system(size: 24)).foregroundColor(.black).frame(width: 28)

Text("Erstellen Sie eine\nneue Übung")

.font(.system(size: 17, weight: .medium)).foregroundColor(.black).lineLimit(2).multilineTextAlignment(.leading)

Spacer()

}

.padding(.horizontal, 24).padding(.vertical, 20)

.background(RoundedRectangle(cornerRadius: 20).fill(Color.white.opacity(0.95)))

.shadow(color: Color.black.opacity(0.2), radius: 10, x: 0, y: 5)

}

.frame(maxWidth: 340)

}

}

**struct** RoutineEditPill: View {

**var** action: () -> Void

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

Button(action: action) {

HStack(spacing: 10) {

Image(systemName: "pencil").font(.system(size: 18, weight: .semibold))

Text("Routine bearbeiten").font(.system(size: 18, weight: .semibold))

}

.foregroundColor(.white)

.padding(.horizontal, 22).frame(height: 56)

.background(Capsule().fill(Color.white.opacity(0.12)).overlay(Capsule().stroke(Color.white.opacity(0.12), lineWidth: 1)))

}

.buttonStyle(.plain)

.shadow(color: .black.opacity(0.35), radius: 12, x: 0, y: 8)

.accessibilityLabel(Text("Routine bearbeiten"))

}

}

**struct** ExerciseAvatar: View {

**let** title: String

**let** systemIcon: String

**var** size: CGFloat = 60

**var** bg: Color = Color(red: 0.95, green: 0.87, blue: 0.7)

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

ZStack {

Circle().fill(bg).frame(width: size, height: size)

Image(systemName: systemIcon).font(.system(size: size \* 0.45)).foregroundColor(Color(red: 0.18, green: 0.20, blue: 0.22))

}

.accessibilityLabel(Text(title))

}

}

**struct** PlusMenuView: View {

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

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

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

VStack(alignment: .leading, spacing: 0) {

MenuRow(icon: "plus", text: "Neue Routine schaffen", action: onNewRoutine)

Divider().background(Color.white.opacity(0.10))

MenuRow(icon: "calendar.badge.plus", text: "Woche hinzufügen", action: onAddWeek)

}

.background(RoundedRectangle(cornerRadius: 16).fill(Color(red: 0.2, green: 0.2, blue: 0.2)))

.frame(width: 280)

}

}

// \*\*\*\*\* START ÄNDERUNG \*\*\*\*\*

**struct** EllipsisMenuView: View {

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

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

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

// let onDuplicate: () -> Void // <-- ENTFERNT

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

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

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

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

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

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

VStack(alignment: .leading, spacing: 0) {

MenuRow(icon: "pencil", text: "Name ändern", action: onChangeName)

Divider().background(Color.white.opacity(0.10))

MenuRow(icon: "square.and.arrow.up", text: "Programm teilen", action: onShareProgram)

Divider().background(Color.white.opacity(0.10))

MenuRow(icon: "figure.strengthtraining.traditional", text: "Sätze pro Muskel-\ngruppe bearbeiten", multiline: **true**, action: onEditSets)

Divider().background(Color.white.opacity(0.10))

// MenuRow(icon: "doc.on.doc", text: "Programm duplizieren", action: onDuplicate) // <-- ENTFERNT

// Divider().background(Color.white.opacity(0.10)) // <-- ENTFERNT

MenuRow(icon: "archivebox", text: "Programm archivieren", action: onArchive)

Divider().background(Color.white.opacity(0.10))

MenuRow(icon: "trash", text: "Programm löschen", tint: .red, action: onDeleteProgram)

}

.background(RoundedRectangle(cornerRadius: 16).fill(Color(red: 0.20, green: 0.20, blue: 0.20)))

.frame(width: 320)

}

}

// \*\*\*\*\* ENDE ÄNDERUNG \*\*\*\*\*

// --- NEUE MODIFIKATION: Hauptmenü-View ---

**struct** TrainingMainMenuView: View {

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

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

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

VStack(alignment: .leading, spacing: 0) {

MenuRow(icon: "square.and.arrow.up", text: "Programm teilen", action: onShareProgram)

Divider().background(Color.white.opacity(0.10)).padding(.horizontal, 10)

MenuRow(icon: "list.bullet", text: "Meine Programme", action: onShowAllPrograms)

}

.background(

RoundedRectangle(cornerRadius: 20)

.fill(Color(red: 0.18, green: 0.18, blue: 0.18)) // Etwas dunklerer Hintergrund

.overlay(RoundedRectangle(cornerRadius: 20).stroke(Color.white.opacity(0.1), lineWidth: 1))

)

.frame(width: 280)

.shadow(color: .black.opacity(0.35), radius: 20, x: 0, y: 8)

}

}

// --- ENDE NEUE MODIFIKATION ---

**private** **struct** MenuRow: View {

**let** icon: String

**let** text: String

**var** tint: Color = .white

**var** multiline: Bool = **false**

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

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

Button(action: action) {

HStack(spacing: 16) {

Image(systemName: icon).font(.system(size: 20)).foregroundColor(tint).frame(width: 24)

Text(text).font(.system(size: 17)).foregroundColor(tint)

.lineLimit(multiline ? 2 : 1).multilineTextAlignment(.leading)

Spacer()

}

.padding(.horizontal, 20).padding(.vertical, 16)

}

}

}

/// Overflow menu for a routine, presented when tapping the ellipsis on a routine.

///

/// This menu presents actions such as renaming, duplicating, changing the activity type

/// and deleting a routine. The "Aktivitätsart ändern" row does \*\***not**\*\* present the

/// activity picker directly; instead it calls the provided `onChangeActivity` closure,

/// allowing the parent view to hide the menu and present a sheet. This avoids the

/// underlying menu staying visible behind the sheet.

**struct** RoutineOverflowMenu: View {

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

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

/// Called when the user taps the "Aktivitätsart ändern" row. The parent view

/// should dismiss the menu (if necessary) and present its own activity picker sheet.

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

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

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

VStack(alignment: .leading, spacing: 0) {

MenuRow(icon: "pencil", text: "Name ändern", action: onRename)

MenuRow(icon: "doc.on.doc", text: "Routine duplizieren", action: onDuplicate)

Divider().background(Color.white.opacity(0.12)).padding(.horizontal, 10)

MenuRow(icon: "figure.strengthtraining.traditional", text: "Aktivitätsart ändern", action: onChangeActivity)

Divider().background(Color.white.opacity(0.12)).padding(.horizontal, 10)

MenuRow(icon: "trash", text: "Routine löschen", tint: .red, action: onDelete)

}

.background(

RoundedRectangle(cornerRadius: 26)

.fill(Color(red: 0.12, green: 0.12, blue: 0.12))

.overlay(

RoundedRectangle(cornerRadius: 26)

.stroke(Color.white.opacity(0.08), lineWidth: 1)

)

)

.frame(width: 340)

.shadow(color: .black.opacity(0.35), radius: 24, x: 0, y: 10)

}

}

// **MARK: - Template / Custom / Dialogs / Etc.**

**struct** TemplateCard: View {

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

VStack(alignment: .leading, spacing: 12) {

ZStack {

Circle().fill(Color.pink.opacity(0.3)).frame(width: 80, height: 80)

Image(systemName: "figure.strengthtraining.traditional").font(.system(size: 36)).foregroundColor(.pink)

}

.padding(.top, 16).frame(maxWidth: .infinity)

VStack(alignment: .leading, spacing: 6) {

Text("Wendlers 5/3/1").font(.system(size: 20, weight: .bold)).foregroundColor(.white)

Text("Ein effektives 4-Wochen-Powerlifting-Programm.")

.font(.system(size: 14)).foregroundColor(.white.opacity(0.7)).lineLimit(2)

}

.padding(.horizontal, 16)

HStack(spacing: 16) {

HStack(spacing: 6) {

Image(systemName: "calendar").font(.system(size: 14)).foregroundColor(.pink)

Text("4 mal / Woche").font(.system(size: 13)).foregroundColor(.white.opacity(0.8))

}

HStack(spacing: 6) {

Image(systemName: "clock").font(.system(size: 14)).foregroundColor(.pink)

Text("30 min").font(.system(size: 13)).foregroundColor(.white.opacity(0.8))

}

}

.padding(.horizontal, 16).padding(.bottom, 16)

}

.frame(width: 240)

.background(RoundedRectangle(cornerRadius: 24).fill(Color.cardBackground))

}

}

**struct** CustomProgramOption: View {

**let** icon: String

**let** title: String

**let** description: String

**let** buttonText: String

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

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

VStack(alignment: .leading, spacing: 16) {

HStack(alignment: .top, spacing: 16) {

ZStack {

RoundedRectangle(cornerRadius: 16).fill(Color.pink.opacity(0.15)).frame(width: 56, height: 56)

Image(systemName: icon).font(.system(size: 26)).foregroundColor(.pink)

}

VStack(alignment: .leading, spacing: 8) {

Text(title).font(.system(size: 20, weight: .semibold)).foregroundColor(.white)

Text(description).font(.system(size: 15)).foregroundColor(.white.opacity(0.7)).fixedSize(horizontal: **false**, vertical: **true**)

}

}

Button(action: action) {

Text(buttonText)

.font(.system(size: 16, weight: .semibold)).foregroundColor(.pink)

.frame(maxWidth: .infinity).frame(height: 44)

.background(RoundedRectangle(cornerRadius: 14).fill(Color.white.opacity(0.08)))

}

}

.padding(20).background(RoundedRectangle(cornerRadius: 24).fill(Color.cardBackground))

}

}

**struct** CreateRoutineDialog: View {

@Binding **var** isPresented: Bool

**let** onRoutineCreated: (String) -> Void

@State **private** **var** routineName: String = ""

@FocusState **private** **var** isTextFieldFocused: Bool

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

VStack(spacing: 24) {

VStack(alignment: .leading, spacing: 12) {

Text("Neue Routine").font(.system(size: 24, weight: .bold)).foregroundColor(.white)

Text("Gib einen Namen für die Routine ein.").font(.system(size: 16)).foregroundColor(.white.opacity(0.7))

}

.frame(maxWidth: .infinity, alignment: .leading)

TextField("Bsp. Rücken und Bizeps", text: $routineName)

.font(.system(size: 17)).foregroundColor(.white)

.focused($isTextFieldFocused).tint(.pink)

.padding(.horizontal, 20).padding(.vertical, 16)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.08)))

HStack(spacing: 12) {

Button { isPresented = **false** } label: {

Text("Abbrechen").font(.system(size: 17, weight: .semibold)).foregroundColor(.white)

.frame(maxWidth: .infinity).frame(height: 50)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.12)))

}

Button {

**if** !routineName.isEmpty { onRoutineCreated(routineName); isPresented = **false** }

} label: {

Text("Erstellen")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white.opacity(routineName.isEmpty ? 0.3 : 1))

.frame(maxWidth: .infinity).frame(height: 50)

.background(RoundedRectangle(cornerRadius: 16).fill(routineName.isEmpty ? Color.white.opacity(0.06) : Color.pink))

}.disabled(routineName.isEmpty)

}

}

.padding(24)

.background(

RoundedRectangle(cornerRadius: 28).fill(Color.cardBackground)

.overlay(RoundedRectangle(cornerRadius: 28).stroke(Color.white.opacity(0.05), lineWidth: 1))

)

.padding(.horizontal, 32)

.onAppear { DispatchQueue.main.asyncAfter(deadline: .now() + 0.3) { isTextFieldFocused = **true** } }

}

}

#if **false**

// This is the old, commented-out definition. The new one is at the top of the file.

**struct** RenameProgramDialog: View {

@Binding **var** isPresented: Bool

@Binding **var** text: String

**let** title: String

**let** subtitle: String

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

@FocusState **private** **var** isFocused: Bool

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

VStack(spacing: 18) {

VStack(alignment: .leading, spacing: 6) {

Text(title).font(.system(size: 22, weight: .bold)).foregroundColor(.white)

Text(subtitle).font(.system(size: 15)).foregroundColor(.white.opacity(0.7)).fixedSize(horizontal: **false**, vertical: **true**)

}

.frame(maxWidth: .infinity, alignment: .leading)

HStack(spacing: 8) {

TextField("Programmname", text: $text)

.textInputAutocapitalization(.words).disableAutocorrection(**true**)

.focused($isFocused).foregroundColor(.white)

**if** !text.isEmpty {

Button { text = "" } label: {

Image(systemName: "xmark.circle.fill").font(.system(size: 18, weight: .semibold)).foregroundColor(.white.opacity(0.6))

}

}

}

.padding(.horizontal, 18).frame(height: 54)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.10)))

HStack(spacing: 12) {

Button { isPresented = **false** } label: {

Text("Abbrechen").font(.system(size: 17, weight: .semibold)).foregroundColor(.white)

.frame(maxWidth: .infinity).frame(height: 52)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.18)))

}

Button {

**let** trimmed = text.trimmingCharacters(in: .whitespacesAndNewlines)

**guard** !trimmed.isEmpty **else** { **return** }

onConfirm(); isPresented = **false**

} label: {

Text("Ändern").font(.system(size: 17, weight: .semibold))

.foregroundColor(.white.opacity(text.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty ? 0.35 : 1))

.frame(maxWidth: .infinity).frame(height: 52)

.background(RoundedRectangle(cornerRadius: 16).fill(text.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty ? Color.white.opacity(0.10) : Color.pink))

}

.disabled(text.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty)

}

}

.padding(22)

.background(

RoundedRectangle(cornerRadius: 26).fill(Color(red: 0.13, green: 0.13, blue: 0.13))

.overlay(RoundedRectangle(cornerRadius: 26).stroke(Color.white.opacity(0.08), lineWidth: 1))

)

.padding(.horizontal, 28)

.onAppear { DispatchQueue.main.asyncAfter(deadline: .now() + 0.1) { isFocused = **true** } }

}

}

#endif

// **MARK: - Kleine Helfer & Tabs**

**struct** AdHocCard: View {

**let** icon: String

**let** title: String

**let** color: Color

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

VStack(alignment: .leading, spacing: 0) {

ZStack {

Circle().fill(color).frame(width: 60, height: 60)

Image(systemName: icon).font(.system(size: 28, weight: .semibold)).foregroundColor(.white)

}

.padding(.top, 20).padding(.leading, 20)

Spacer()

Text(title).font(.system(size: 17, weight: .semibold)).foregroundColor(.white).lineLimit(4)

.padding(.leading, 20).padding(.bottom, 20).padding(.trailing, 20)

}

.frame(width: 240, height: 220)

.background(RoundedRectangle(cornerRadius: 24).fill(Color(red: 0.2, green: 0.22, blue: 0.22)))

}

}

**struct** OnboardingCard: View {

@Binding **var** showAddProgramSheet: Bool

**let** circleIcons = ["figure.strengthtraining.traditional","dumbbell","figure.run"]

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

VStack(spacing: 24) {

HStack(spacing: 20) {

ForEach(circleIcons, id: \.**self**) { name **in**

ZStack {

Circle().fill(Color.white.opacity(0.12)).frame(width: 80, height: 80)

Image(systemName: name).font(.system(size: 34, weight: .semibold)).foregroundColor(.white)

}

}

}

.padding(.top, 8)

Text("Beginnen").font(.system(size: 28, weight: .bold)).foregroundColor(.white)

Text("Erstellen Sie Ihren eigenen Programm von Grund auf.")

.font(.system(size: 17)).foregroundColor(.white.opacity(0.9))

.multilineTextAlignment(.center).padding(.horizontal, 12)

VStack(spacing: 14) {

Button { showAddProgramSheet = **true** } label: {

Text("Programm hinzufügen").font(.headline).frame(maxWidth: .infinity, minHeight: 50)

}

.buttonStyle(FilledRoundedButtonStyle())

}

.padding(.top, 8).padding(.bottom, 4)

}

.padding(.vertical, 32).padding(.horizontal, 20)

.background(RoundedRectangle(cornerRadius: 32).fill(Color.cardBackground))

}

}

// Button Style

**struct** FilledRoundedButtonStyle: ButtonStyle {

**func** makeBody(configuration: Configuration) -> **some** View {

configuration.label

.foregroundColor(.white)

.background(

RoundedRectangle(cornerRadius: 18)

.fill(Color.pink.opacity(configuration.isPressed ? 0.85 : 1))

)

.scaleEffect(configuration.isPressed ? 0.98 : 1)

.animation(.easeOut(duration: 0.1), value: configuration.isPressed)

}

}

**struct** AddProgramSheet: View {

@EnvironmentObject **var** viewModel: TrainingViewModel

@Binding **var** isPresented: Bool

@State **private** **var** showCreateEmptyProgram = **false**

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

ZStack {

Color.black.ignoresSafeArea()

VStack(spacing: 0) {

ZStack {

Text("Programm hinzufügen").font(.system(size: 18, weight: .semibold)).foregroundColor(.white)

HStack {

Button { isPresented = **false** } label: {

ZStack {

Circle().fill(Color.white.opacity(0.15)).frame(width: 44, height: 44)

Image(systemName: "xmark").font(.system(size: 16, weight: .semibold)).foregroundColor(.white)

}

}

Spacer()

}

.padding(.horizontal, 20)

}

.padding(.top, 16).padding(.bottom, 20)

ScrollView {

VStack(alignment: .leading, spacing: 32) {

VStack(alignment: .leading, spacing: 16) {

Text("Vorlagen").font(.system(size: 28, weight: .bold)).foregroundColor(.white).padding(.horizontal, 20)

HStack(spacing: 16) {

TemplateCard()

VStack(alignment: .trailing, spacing: 4) {

Button { } label: {

HStack(spacing: 6) {

Text("Alle Vorlagen\ndurchsuchen").font(.system(size: 17, weight: .semibold)).multilineTextAlignment(.trailing)

Image(systemName: "arrow.right.circle.fill").font(.system(size: 24))

}

.foregroundColor(.pink)

}

}

.padding(.trailing, 20)

}

.padding(.leading, 20)

}

VStack(alignment: .leading, spacing: 16) {

Text("Benutzerdefinierte Programme").font(.system(size: 28, weight: .bold)).foregroundColor(.white).padding(.horizontal, 20)

VStack(spacing: 16) {

CustomProgramOption(

icon: "list.clipboard",

title: "Von vorne beginnen",

description: "Beginnen Sie mit einem leeren Programm und fügen Sie Ihre Routinen und Übungen manuell hinzu.",

buttonText: "Leeres Programm erstellen",

action: { showCreateEmptyProgram = **true** }

)

}

.padding(.horizontal, 20)

}

}

.padding(.top, 8).padding(.bottom, 32)

}

}

}

.sheet(isPresented: $showCreateEmptyProgram) {

CreateEmptyProgramSheet(parentSheetBinding: $isPresented).environmentObject(viewModel)

}

}

}

**struct** CreateEmptyProgramSheet: View {

@EnvironmentObject **var** viewModel: TrainingViewModel

@Environment(\.dismiss) **var** dismiss

@Binding **var** parentSheetBinding: Bool

@State **private** **var** programName: String = ""

@State **private** **var** navigateToProgramDetail = **false**

@FocusState **private** **var** isTextFieldFocused: Bool

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

NavigationStack {

ZStack {

Color.black.ignoresSafeArea()

VStack(spacing: 0) {

HStack {

Text("Programm hinzufügen").font(.system(size: 17)).foregroundColor(.white.opacity(0.6))

Spacer()

Button { dismiss() } label: {

ZStack { Circle().fill(Color.white.opacity(0.15)).frame(width: 44, height: 44)

Image(systemName: "xmark").font(.system(size: 16, weight: .semibold)).foregroundColor(.white) }

}

}

.padding(.horizontal, 20).padding(.top, 16).padding(.bottom, 20)

Spacer()

VStack(spacing: 24) {

VStack(alignment: .leading, spacing: 12) {

Text("Neues Programm").font(.system(size: 24, weight: .bold)).foregroundColor(.white)

Text("Gib einen Namen für das Programm ein.").font(.system(size: 16)).foregroundColor(.white.opacity(0.7))

}

.frame(maxWidth: .infinity, alignment: .leading)

TextField("Bsp. Push Pull Beine", text: $programName)

.font(.system(size: 17)).foregroundColor(.white).focused($isTextFieldFocused).tint(.pink)

.padding(.horizontal, 20).padding(.vertical, 16)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.08)))

HStack(spacing: 12) {

Button { dismiss() } label: {

Text("Abbrechen").font(.system(size: 17, weight: .semibold)).foregroundColor(.white)

.frame(maxWidth: .infinity).frame(height: 50)

.background(RoundedRectangle(cornerRadius: 16).fill(Color.white.opacity(0.12)))

}

Button { navigateToProgramDetail = **true** } label: {

Text("Erstellen")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white.opacity(programName.isEmpty ? 0.3 : 1))

.frame(maxWidth: .infinity).frame(height: 50)

.background(RoundedRectangle(cornerRadius: 16).fill(programName.isEmpty ? Color.white.opacity(0.06) : Color.pink))

}.disabled(programName.isEmpty)

}

}

.padding(24)

.background(

RoundedRectangle(cornerRadius: 28).fill(Color.cardBackground)

.overlay(RoundedRectangle(cornerRadius: 28).stroke(Color.white.opacity(0.05), lineWidth: 1))

)

.padding(.horizontal, 32)

Spacer()

}

}

.navigationDestination(isPresented: $navigateToProgramDetail) {

ProgramDetailView(parentSheetBinding: $parentSheetBinding, programName: programName).environmentObject(viewModel)

}

.onAppear { DispatchQueue.main.asyncAfter(deadline: .now() + 0.5) { isTextFieldFocused = **true** } }

}

}

}

**struct** ProgramDetailView: View {

@EnvironmentObject **var** viewModel: TrainingViewModel

@Environment(\.dismiss) **var** dismiss

// kommt vom CreateEmptyProgramSheet

@Binding **var** parentSheetBinding: Bool

// programmname im Erstellungs-Flow veränderbar

**let** initialProgramName: String

@State **private** **var** programName: String = ""

// Routinen & Auswahl

@State **private** **var** routines: [RoutineData] = []

@State **private** **var** selectedRoutineIndex: Int = 0

// Menüs / Dialoge

@State **private** **var** showPlusMenu = **false**

@State **private** **var** showEllipsisMenu = **false**

@State **private** **var** showRoutineMenu = **false**

// Programm umbenennen

@State **private** **var** showRenameDialog = **false**

@State **private** **var** renameText: String = ""

// Routine umbenennen

@State **private** **var** showRenameRoutineDialog = **false**

@State **private** **var** routineRenameText: String = ""

// Übungsliste

@State **private** **var** showExerciseList = **false**

// „Sätze pro Muskelgruppe“

@State **private** **var** showEditSets = **false**

@State **private** **var** editSetsRoutineIndex = 0

// Share

@State **private** **var** showShareSheet = **false**

@State **private** **var** shareItems: [**Any**] = []

// Delete (Entwurf)

@State **private** **var** showDeleteAlert = **false**

/// Controls presentation of the dialog to duplicate a routine in the creation flow.

@State **private** **var** showDuplicateRoutineDialog = **false**

/// The provisional name for a duplicated routine. Bound to the text field in the duplication dialog.

@State **private** **var** duplicateRoutineName: String = ""

/// Controls presentation of the change activity sheet when editing a routine in the program creation flow.

@State **private** **var** showChangeActivityDialog = **false**

// eigener Init, damit programName @State ist

**init**(parentSheetBinding: Binding<Bool>, programName: String) {

**self**.\_parentSheetBinding = parentSheetBinding

**self**.initialProgramName = programName

**self**.\_programName = State(initialValue: programName)

}

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

ZStack {

Color.black.ignoresSafeArea()

VStack(spacing: 0) {

// Header

HStack(spacing: 16) {

headerCircle("chevron.left") { dismiss() }

Spacer()

Text(programName)

.font(.system(size: 18, weight: .semibold))

.foregroundColor(.white)

Spacer()

HStack(spacing: 12) {

headerRect("plus") { showPlusMenu = **true** }

headerRect("ellipsis") { showEllipsisMenu = **true** }

}

// Speichern

headerCircle("checkmark") {

**let** program = ProgramData(name: programName, routines: routines)

viewModel.addProgram(program)

parentSheetBinding = **false**

}

}

.padding(.horizontal, 20)

.padding(.top, 16)

.padding(.bottom, 12)

// Inhalt

ScrollView {

VStack(alignment: .leading, spacing: 16) {

// --- MODIFIKATION: Äußere ForEach nutzt jetzt $routines ---

ForEach($routines) { $routine **in** // Binding $routine

VStack(alignment: .leading, spacing: 16) {

HStack {

Text(routine.name) // Verwende routine.name

.font(.system(size: 28, weight: .bold))

.foregroundColor(.white)

Button {

// Finde den Index für das Menü

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

selectedRoutineIndex = index

withAnimation { showRoutineMenu = **true** }

}

} label: {

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

.font(.system(size: 24))

.foregroundColor(.pink)

}

}

Text("\(routine.exercises.count) Übungen") // Verwende routine.exercises

.font(.system(size: 15))

.foregroundColor(.white.opacity(0.6))

**if** routine.exercises.isEmpty {

HStack {

Text("Keine Übungen")

.font(.system(size: 17))

.foregroundColor(.white.opacity(0.5))

Spacer()

}

.padding(.horizontal, 20)

.padding(.vertical, 16)

.background(

RoundedRectangle(cornerRadius: 16)

.fill(Color.white.opacity(0.08))

)

} **else** {

// --- MODIFIKATION: VStack durch List ersetzt ---

List {

ForEach(routine.exercises) { addedExercise **in**

AddedExerciseRow(addedExercise: addedExercise)

.listRowBackground(Color.clear)

.listRowInsets(EdgeInsets())

.listRowSeparator(.hidden)

}

.onMove { from, to **in**

// .onMove funktioniert jetzt mit dem $routine binding

routine.exercises.move(fromOffsets: from, toOffset: to)

}

}

.listStyle(.plain)

.frame(height: CGFloat(routine.exercises.count) \* 84) // Höhe anpassen

.background(

RoundedRectangle(cornerRadius: 16)

.fill(Color.white.opacity(0.08))

)

// --- ENDE MODIFIKATION ---

}

Button {

// Finde den Index für das Hinzufügen-Sheet

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

selectedRoutineIndex = index

showExerciseList = **true**

}

} label: {

Text("Hinzufügen +")

.font(.system(size: 16, weight: .semibold))

.foregroundColor(.pink)

.padding(.horizontal, 20)

.padding(.vertical, 10)

.background(

RoundedRectangle(cornerRadius: 14)

.fill(Color.white.opacity(0.08))

)

}

}

.padding(.horizontal, 20)

.padding(.top, 8)

}

// --- ENDE MODIFIKATION ---

// Neue Routine

VStack(spacing: 16) {

Button { showCreateRoutineDialog = **true** } label: {

HStack(spacing: 16) {

ZStack {

RoundedRectangle(cornerRadius: 14)

.fill(Color.pink.opacity(0.15))

.frame(width: 48, height: 48)

Image(systemName: "list.clipboard")

.font(.system(size: 24))

.foregroundColor(.pink)

}

Text("Neue Routine schaffen")

.font(.system(size: 18, weight: .semibold))

.foregroundColor(.pink)

Spacer()

}

.padding(20)

.background(

RoundedRectangle(cornerRadius: 20)

.fill(Color.cardBackground)

)

}

}

.padding(.horizontal, 20)

.padding(.top, routines.isEmpty ? 8 : 16)

}

}

}

// PLUS-Menü

**if** showPlusMenu {

overlayDim { showPlusMenu = **false** }

anchoredTopRight {

PlusMenuView(

onNewRoutine: { showPlusMenu = **false** ; showCreateRoutineDialog = **true** },

onAddWeek: { showPlusMenu = **false** }

)

.padding(.trailing, 76)

.padding(.top, 90)

}

}

// PROGRAMM: Ellipsis-Menü

**if** showEllipsisMenu {

overlayDim { showEllipsisMenu = **false** }

anchoredTopRight {

EllipsisMenuView(

onChangeName: {

showEllipsisMenu = **false**

renameText = programName

showRenameDialog = **true**

},

onShareProgram: {

showEllipsisMenu = **false**

presentShareSheetForCurrentDraft()

},

onEditSets: {

showEllipsisMenu = **false**

editSetsRoutineIndex = ensureRoutineForEditSets()

showEditSets = **true**

},

// onDuplicate: { showEllipsisMenu = false }, // <-- ENTFERNT

onSetDefault: { showEllipsisMenu = **false** },

onChangeAppearance: { showEllipsisMenu = **false** },

onArchive: { showEllipsisMenu = **false** },

onResetProgress: { showEllipsisMenu = **false** },

onDeleteProgram: {

showEllipsisMenu = **false**

showDeleteAlert = **true**

}

)

.padding(.trailing, 20)

.padding(.top, 90)

}

}

// ROUTINE: Overflow-Menü „…“

**if** showRoutineMenu {

overlayDim { withAnimation { showRoutineMenu = **false** } }

anchoredTopRight {

RoutineOverflowMenu(

onRename: {

showRoutineMenu = **false**

routineRenameText = routines[**safe**: selectedRoutineIndex]?.name ?? ""

showRenameRoutineDialog = **true**

},

onDuplicate: {

// Prepare default name for the duplicated routine and present a dialog instead of duplicating immediately

**if** routines.indices.contains(selectedRoutineIndex) {

**let** src = routines[selectedRoutineIndex]

**let** suffix = " Kopie"

duplicateRoutineName = src.name.hasSuffix(suffix) ? src.name : (src.name + suffix)

} **else** {

duplicateRoutineName = ""

}

withAnimation { showRoutineMenu = **false** }

showDuplicateRoutineDialog = **true**

},

onChangeActivity: {

// Hide the routine menu and present the activity picker sheet

withAnimation { showRoutineMenu = **false** }

showChangeActivityDialog = **true**

},

onDelete: {

**if** routines.indices.contains(selectedRoutineIndex) {

routines.remove(at: selectedRoutineIndex)

}

withAnimation { showRoutineMenu = **false** }

}

)

.padding(.trailing, 24)

.padding(.top, 140)

}

.transition(AnyTransition.opacity.combined(with: .move(edge: .top)))

}

// PROGRAMM: Rename-Dialog

**if** showRenameDialog {

Color.black.opacity(0.45).ignoresSafeArea()

RenameProgramDialog(

isPresented: $showRenameDialog,

text: $renameText,

title: "Name ändern",

subtitle: "Gib einen Namen für den Programm ein.",

onConfirm: {

**let** trimmed = renameText.trimmingCharacters(in: .whitespacesAndNewlines)

**guard** !trimmed.isEmpty **else** { **return** }

programName = trimmed

}

)

.transition(AnyTransition.scale.combined(with: .opacity))

}

// ROUTINE: Rename-Dialog

**if** showRenameRoutineDialog {

Color.black.opacity(0.45).ignoresSafeArea()

RenameProgramDialog(

isPresented: $showRenameRoutineDialog,

text: $routineRenameText,

title: "Name ändern",

subtitle: "Gib einen Namen für die Routine ein.",

onConfirm: {

**let** trimmed = routineRenameText.trimmingCharacters(in: .whitespacesAndNewlines)

**guard** !trimmed.isEmpty,

routines.indices.contains(selectedRoutineIndex) **else** { **return** }

routines[selectedRoutineIndex].name = trimmed

}

)

.transition(AnyTransition.scale.combined(with: .opacity))

}

// Duplicate routine dialog overlay

**if** showDuplicateRoutineDialog {

Color.black.opacity(0.45).ignoresSafeArea()

DuplicateRoutineDialog(

isPresented: $showDuplicateRoutineDialog,

text: $duplicateRoutineName,

onConfirm: {

// Duplicate the selected routine with the provided name

**let** trimmed = duplicateRoutineName.trimmingCharacters(in: .whitespacesAndNewlines)

**guard** !trimmed.isEmpty **else** { **return** }

**if** routines.indices.contains(selectedRoutineIndex) {

**let** idx = selectedRoutineIndex

**let** src = routines[idx]

**let** newExercises = src.exercises.map { e **in**

AddedExercise(

exercise: e.exercise,

weight: e.weight,

sets: e.sets,

reps: e.reps

)

}

**let** clone = RoutineData(

name: trimmed,

exercises: newExercises,

activityType: src.activityType

)

routines.insert(clone, at: idx + 1)

}

}

)

.transition(AnyTransition.scale.combined(with: .opacity))

}

}

.navigationBarHidden(**true**)

.sheet(isPresented: $showExerciseList) {

ExerciseListView(onExercisesAdded: { exs **in**

**for** ex **in** exs {

routines[selectedRoutineIndex].exercises.append(AddedExercise(exercise: ex))

}

})

}

.sheet(isPresented: $showEditSets) {

**if** routines.indices.contains(editSetsRoutineIndex) {

EditSetsPerMuscleView(routine: $routines[editSetsRoutineIndex], isPresented: $showEditSets)

}

}

#if os(iOS)

// Present the activity type picker for a routine in the draft creation flow. The sheet appears at half height by default

// and can be dragged to full screen. The binding ensures the routine's activity type updates automatically when the user selects a type.

.sheet(isPresented: $showChangeActivityDialog) {

**if** routines.indices.contains(selectedRoutineIndex) {

ChangeActivityDialog(

isPresented: $showChangeActivityDialog,

selectedActivity: $routines[selectedRoutineIndex].activityType

) { newType **in**

// The binding above updates the activity type automatically. Additional actions can be placed here.

}

.presentationDetents([.fraction(0.5), .large])

.presentationDragIndicator(.visible)

}

}

#endif

#if os(iOS)

.sheet(isPresented: $showShareSheet) {

ShareSheet(items: shareItems, subject: "Programm: \(programName)")

}

#endif

// neuer Routine-Dialog

.overlay(

Group {

**if** showCreateRoutineDialog {

Color.black.opacity(0.5).ignoresSafeArea()

.onTapGesture { showCreateRoutineDialog = **false** }

CreateRoutineDialog(

isPresented: $showCreateRoutineDialog,

onRoutineCreated: { routines.append(RoutineData(name: $0)) }

)

}

}

)

.alert("Entwurf verwerfen?", isPresented: $showDeleteAlert) {

Button("Verwerfen", role: .destructive) {

parentSheetBinding = **false** // gesamten Add-Flow schließen

}

Button("Abbrechen", role: .cancel) {}

} message: {

Text("„\(programName)“ wurde noch nicht gespeichert.")

}

}

// **MARK: - kleine Helfer**

@ViewBuilder **private** **func** headerCircle(\_ system: String, action: **@escaping** () -> Void) -> **some** View {

Button(action: action) {

ZStack {

Circle().fill(Color.white.opacity(0.15)).frame(width: 50, height: 50)

Image(systemName: system).font(.system(size: 18, weight: .semibold)).foregroundColor(.white)

}

}

}

@ViewBuilder **private** **func** headerRect(\_ system: String, action: **@escaping** () -> Void) -> **some** View {

Button(action: action) {

Image(systemName: system)

.font(.system(size: 18, weight: .semibold))

.foregroundColor(.white)

.frame(width: 50, height: 50)

.background(RoundedRectangle(cornerRadius: 25).fill(Color.white.opacity(0.15)))

}

}

@ViewBuilder **private** **func** overlayDim(onTap: **@escaping** () -> Void) -> **some** View {

Color.black.opacity(0.5).ignoresSafeArea().onTapGesture(perform: onTap)

}

@ViewBuilder **private** **func** anchoredTopRight<Content: View>(@ViewBuilder \_ content: () -> Content) -> **some** View {

VStack { HStack { Spacer(); content() }; Spacer() }

}

// Flag für Overlay unten:

@State **private** **var** showCreateRoutineDialog = **false**

// Routine sicherstellen für Edit-Sets (Erstellungs-Flow)

**private** **func** ensureRoutineForEditSets() -> Int {

**if** routines.isEmpty {

**let** newName = "Routine 1"

routines.append(RoutineData(name: newName))

selectedRoutineIndex = 0

**return** 0

}

**if** routines.indices.contains(selectedRoutineIndex) {

**return** selectedRoutineIndex

}

selectedRoutineIndex = 0

**return** 0

}

**private** **func** presentShareSheetForCurrentDraft() {

**let** draft = ProgramData(name: programName, routines: routines)

shareItems = makeProgramShareItems(program: draft)

showShareSheet = **true**

}

}

// **MARK: - RoutineCard**

**struct** RoutineCard: View {

**let** routine: RoutineData

**let** isNext: Bool

**let** cardColor: Color

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

**var** totalExercises: Int { routine.exercises.count }

**var** totalWeight: Int { 0 } // Dummy value, logic for "KG" string parsing not implemented

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

Button(action: onTap) {

ZStack(alignment: .topTrailing) {

VStack(alignment: .leading, spacing: 16) {

// Icon

ZStack {

Circle()

.fill(Color(red: 0.95, green: 0.87, blue: 0.7))

.frame(width: 120, height: 120)

Image(systemName: "figure.strengthtraining.traditional") // Placeholder icon

.font(.system(size: 60))

.foregroundColor(Color(red: 0.8, green: 0.6, blue: 0.3))

}

.frame(maxWidth: .infinity)

.padding(.top, 32)

// Titel (Routine Name) + Zahlen

VStack(alignment: .leading, spacing: 8) {

Text(routine.name) // Use Routine name

.font(.system(size: 24, weight: .bold))

.foregroundColor(.white)

.lineLimit(1)

HStack(spacing: 24) {

HStack(spacing: 4) {

Text("\(totalExercises)")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white)

Text("ÜBUNGEN")

.font(.system(size: 13, weight: .medium))

.foregroundColor(.white.opacity(0.6))

}

HStack(spacing: 4) {

Text("\(totalWeight)")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white)

Text("KG")

.font(.system(size: 13, weight: .medium))

.foregroundColor(.white.opacity(0.6))

}

}

}

.padding(.horizontal, 20)

// Übungsliste (kurz)

VStack(alignment: .leading, spacing: 8) {

// Loop over exercises in \*this\* routine

ForEach(routine.exercises.prefix(3)) { exercise **in**

Text(exercise.exercise.name)

.font(.system(size: 17))

.foregroundColor(.white)

.lineLimit(1)

}

}

.padding(.horizontal, 20)

.padding(.bottom, 20)

.frame(minHeight: 100, alignment: .top) // Give space for exercises

}

.frame(width: 260) // Set a fixed width for horizontal scrolling

.background(

RoundedRectangle(cornerRadius: 24, style: .continuous)

.fill(cardColor) // Use the passed-in color

)

// „Nächstes“-Badge

**if** isNext {

Text("NÄCHS-\nTES")

.font(.system(size: 11, weight: .bold))

.foregroundColor(.white)

.multilineTextAlignment(.center)

.padding(.horizontal, 12)

.padding(.vertical, 8)

.background(Circle().fill(Color.pink))

.offset(x: -16, y: 16)

}

}

}

.buttonStyle(.plain)

}

}

// **MARK: - CreateExerciseDialog**

**struct** CreateExerciseDialog: View {

@Binding **var** isPresented: Bool

**let** onExerciseCreated: (String) -> Void

@State **private** **var** exerciseName: String = ""

@FocusState **private** **var** isTextFieldFocused: Bool

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

VStack(spacing: 24) {

VStack(alignment: .leading, spacing: 12) {

Text("Übung erstellen")

.font(.system(size: 24, weight: .bold))

.foregroundColor(.white)

Text("Gib den Namen der Übung ein.")

.font(.system(size: 16))

.foregroundColor(.white.opacity(0.7))

}

.frame(maxWidth: .infinity, alignment: .leading)

TextField("", text: $exerciseName)

.font(.system(size: 17))

.foregroundColor(.white)

.focused($isTextFieldFocused)

.tint(.pink)

.padding(.horizontal, 20)

.padding(.vertical, 16)

.background(

RoundedRectangle(cornerRadius: 16, style: .continuous)

.fill(Color.white.opacity(0.08))

)

HStack(spacing: 12) {

Button { isPresented = **false** } label: {

Text("Abbrechen")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white)

.frame(maxWidth: .infinity)

.frame(height: 50)

.background(

RoundedRectangle(cornerRadius: 16, style: .continuous)

.fill(Color.white.opacity(0.12))

)

}

Button {

**guard** !exerciseName.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty **else** { **return** }

onExerciseCreated(exerciseName)

isPresented = **false**

} label: {

Text("Weiter")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white.opacity(exerciseName.isEmpty ? 0.3 : 1))

.frame(maxWidth: .infinity)

.frame(height: 50)

.background(

RoundedRectangle(cornerRadius: 16, style: .continuous)

.fill(exerciseName.isEmpty ? Color.white.opacity(0.06) : Color.pink)

)

}

.disabled(exerciseName.isEmpty)

}

}

.padding(24)

.background(

RoundedRectangle(cornerRadius: 28, style: .continuous)

.fill(Color(red: 0.2, green: 0.22, blue: 0.22))

.overlay(

RoundedRectangle(cornerRadius: 28, style: .continuous)

.stroke(Color.white.opacity(0.05), lineWidth: 1)

)

)

.padding(.horizontal, 32)

.onAppear { DispatchQueue.main.asyncAfter(deadline: .now() + 0.3) { isTextFieldFocused = **true** } }

}

}

// **MARK: - ExerciseDetailSheet**

**struct** ExerciseDetailSheet: View {

@Environment(\.dismiss) **var** dismiss

**let** exerciseName: String

@State **private** **var** primaryMuscle = "Brust"

@State **private** **var** secondaryMuscles = "Nicht festgelegt"

@State **private** **var** equipment = "Langhantel"

@State **private** **var** barbellWeight = "10 kg"

@State **private** **var** timerEnabled = **false**

@State **private** **var** doubleVolumeEnabled = **false**

@State **private** **var** trackOneRM = **true**

@State **private** **var** warmupCalculator = "Nicht gesetzt"

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

ZStack {

Color.black.ignoresSafeArea()

VStack(spacing: 0) {

// Header

HStack {

headerCircle("xmark") { dismiss() }

Spacer()

Text(exerciseName)

.font(.system(size: 18, weight: .semibold))

.foregroundColor(.white)

Spacer()

Button { dismiss() } label: {

ZStack {

Circle().fill(Color.pink).frame(width: 50, height: 50)

Image(systemName: "checkmark")

.font(.system(size: 18, weight: .semibold))

.foregroundColor(.white)

}

}

}

.padding(.horizontal, 20)

.padding(.top, 16)

.padding(.bottom, 20)

// Inhalt

ScrollView {

VStack(spacing: 0) {

rowNav(title: "Primäre Muskelgruppe", value: primaryMuscle)

divider()

rowNav(title: "Andere Muskeln", value: secondaryMuscles)

divider()

rowNavPink(title: "Equipment", value: equipment)

VStack(alignment: .leading, spacing: 0) {

HStack {

Text("Langhantel").font(.system(size: 17)).foregroundColor(.white)

Spacer()

Text(barbellWeight).font(.system(size: 17)).foregroundColor(.pink)

}

.padding(.horizontal, 20)

.padding(.vertical, 16)

.background(Color(red: 0.15, green: 0.15, blue: 0.15))

Text("Sie können alle Ihre Geräte in den Einstellungen > Ausrüstung verwalten.")

.font(.system(size: 13))

.foregroundColor(.white.opacity(0.5))

.padding(.horizontal, 20)

.padding(.top, 8)

.padding(.bottom, 20)

.frame(maxWidth: .infinity, alignment: .leading)

.background(Color(red: 0.15, green: 0.15, blue: 0.15))

}

.padding(.top, 20)

toggleRow(title: "Übung mit Timer", on: $timerEnabled,

hint: "Aktivieren, wenn Wiederholungen nicht sinnvoll sind (z. B. Planks).")

.padding(.top, 20)

toggleRow(title: "Doppelte Volumen", on: $doubleVolumeEnabled,

hint: "Für einseitige/Kurzhantel-Übungen – korrektes Volumen.")

.padding(.top, 20)

toggleRow(title: "1RM aufzeichnen", on: $trackOneRM,

hint: "Verfolge deinen One-Rep-Max für diese Übung.")

.padding(.top, 20)

VStack(alignment: .leading, spacing: 0) {

Button { } label: {

HStack {

Text("Aufwärm-Rechner").font(.system(size: 17)).foregroundColor(.white)

Spacer()

Text(warmupCalculator).font(.system(size: 17)).foregroundColor(.pink)

}

.padding(.horizontal, 20).padding(.vertical, 16)

.background(Color(red: 0.15, green: 0.15, blue: 0.15))

}

Text("Wird als Standard für diese Übung verwendet.")

.font(.system(size: 13)).foregroundColor(.white.opacity(0.5))

.padding(.horizontal, 20).padding(.top, 8).padding(.bottom, 20)

.frame(maxWidth: .infinity, alignment: .leading)

.background(Color(red: 0.15, green: 0.15, blue: 0.15))

}

.padding(.top, 20)

Button { } label: {

Text("Namen ändern")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.pink)

.frame(maxWidth: .infinity)

.padding(.vertical, 16)

}

.padding(.horizontal, 20)

.padding(.top, 32)

.padding(.bottom, 40)

}

}

}

}

}

// kleine Helfer

**private** **func** rowNav(title: String, value: String) -> **some** View {

Button { } label: {

HStack {

Text(title).font(.system(size: 17)).foregroundColor(.white)

Spacer()

Text(value).font(.system(size: 17)).foregroundColor(.white.opacity(0.5))

Image(systemName: "chevron.right").font(.system(size: 14, weight: .semibold)).foregroundColor(.white.opacity(0.3))

}

.padding(.horizontal, 20).padding(.vertical, 16).background(Color(red: 0.15, green: 0.15, blue: 0.15))

}

}

**private** **func** rowNavPink(title: String, value: String) -> **some** View {

Button { } label: {

HStack {

Text(title).font(.system(size: 17)).foregroundColor(.white)

Spacer()

Text(value).font(.system(size: 17)).foregroundColor(.pink)

}

.padding(.horizontal, 20).padding(.vertical, 16).background(Color(red: 0.15, green: 0.15, blue: 0.15))

}

}

**private** **func** divider() -> **some** View {

Divider().background(Color.white.opacity(0.1)).padding(.leading, 20)

}

**private** **func** toggleRow(title: String, on: Binding<Bool>, hint: String) -> **some** View {

VStack(alignment: .leading, spacing: 0) {

HStack {

Text(title).font(.system(size: 17)).foregroundColor(.white)

Spacer()

Toggle("", isOn: on).tint(.pink)

}

.padding(.horizontal, 20).padding(.vertical, 16)

.background(Color(red: 0.15, green: 0.15, blue: 0.15))

Text(hint)

.font(.system(size: 13)).foregroundColor(.white.opacity(0.5))

.padding(.horizontal, 20).padding(.top, 8).padding(.bottom, 20)

.frame(maxWidth: .infinity, alignment: .leading)

.background(Color(red: 0.15, green: 0.15, blue: 0.15))

}

}

@ViewBuilder **private** **func** headerCircle(\_ system: String, action: **@escaping** () -> Void) -> **some** View {

Button(action: action) {

ZStack {

Circle().fill(Color.white.opacity(0.15)).frame(width: 50, height: 50)

Image(systemName: system).font(.system(size: 18, weight: .semibold)).foregroundColor(.white)

}

}

}

}

// **MARK: - Tabs**

**struct** ProgressViewScreen: View {

**var** body: **some** View { ZStack { Color.black.ignoresSafeArea(); Text("Fortschritt").foregroundColor(.white.opacity(0.9)).font(.title) } }

}

**struct** DiscoverViewScreen: View {

**var** body: **some** View { ZStack { Color.black.ignoresSafeArea(); Text("Entdecken").foregroundColor(.white.opacity(0.9)).font(.title) } }

}

// --- START ÄNDERUNG: SettingsViewScreen wurde überarbeitet ---

/// Zeigt eine einzelne Zeile in den Einstellungen, die zu einer Detailansicht navigiert.

**struct** SettingsNavigationRow: View {

**let** iconName: String

**let** iconColor: Color

**let** title: String

**let** subtitle: String?

**init**(iconName: String, iconColor: Color, title: String, subtitle: String? = **nil**) {

**self**.iconName = iconName

**self**.iconColor = iconColor

**self**.title = title

**self**.subtitle = subtitle

}

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

HStack(spacing: 16) {

ZStack {

RoundedRectangle(cornerRadius: 6)

.fill(iconColor)

.frame(width: 28, height: 28)

Image(systemName: iconName)

.font(.system(size: 15, weight: .medium))

.foregroundColor(.white)

}

Text(title)

.font(.system(size: 17))

.foregroundColor(.white)

Spacer()

**if** **let** subtitle {

Text(subtitle)

.font(.system(size: 17))

.foregroundColor(.white.opacity(0.6))

}

Image(systemName: "chevron.right")

.font(.system(size: 14, weight: .semibold))

.foregroundColor(.white.opacity(0.3))

}

.padding(.vertical, 10) // Etwas mehr Platz

.background(Color(red: 0.1, green: 0.1, blue: 0.1)) // Hintergrund für die Zeile, damit der Klickbereich klar ist

}

}

// ==========================================================

// 1) NEU: Übungen-Katalog als nicht-modale Ansicht (Push)

// ==========================================================

**struct** ExercisesCatalogView: View {

@Environment(\.dismiss) **private** **var** dismiss

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

@State **private** **var** sortAscending: Bool = **true**

@State **private** **var** showCreateExerciseDialog: Bool = **false**

@State **private** **var** createdExerciseName: String = ""

// Quelle: nutzt dein vorhandenes Exercise-Modell

@State **private** **var** allExercises: [Exercise] = ExerciseCatalog.defaultExercises

// Suche + Sortierung

**private** **var** filteredSorted: [Exercise] {

**var** list = allExercises

**if** !searchText.isEmpty {

list = list.filter {

$0.name.localizedCaseInsensitiveContains(searchText)

|| $0.category.localizedCaseInsensitiveContains(searchText)

|| $0.type.localizedCaseInsensitiveContains(searchText)

}

}

list.sort { sortAscending ? $0.name < $1.name : $0.name > $1.name }

**return** list

}

// Alphabetische Gruppierung

**private** **var** grouped: [(String, [Exercise])] {

**let** dict = Dictionary(grouping: filteredSorted) {

String($0.name.prefix(1)).uppercased()

}

**return** dict.keys.sorted().map { key **in**

(key, dict[key]!.sorted { $0.name < $1.name })

}

}

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

ZStack {

Color.black.ignoresSafeArea()

VStack(spacing: 0) {

// Header wie im Screenshot (Zurück links, runde Buttons rechts)

HStack {

headerCircle("chevron.left") { dismiss() }

Spacer()

HStack(spacing: 12) {

headerRoundButton(icon: "plus") { showCreateExerciseDialog = **true** }

headerRoundButton(icon: "arrow.up.arrow.down") { withAnimation(.easeInOut(duration: 0.15)) { sortAscending.toggle() } }

}

}

.padding(.horizontal, 20)

.padding(.top, 16)

.padding(.bottom, 8)

// Suchleiste

HStack(spacing: 10) {

Image(systemName: "magnifyingglass")

.foregroundColor(.white.opacity(0.5))

TextField("Suchen", text: $searchText)

.foregroundColor(.white)

.textInputAutocapitalization(.none)

.disableAutocorrection(**true**)

}

.padding(.horizontal, 16)

.frame(height: 46)

.background(

Capsule().fill(Color.white.opacity(0.12))

)

.padding(.horizontal, 20)

.padding(.bottom, 8)

// Scrollbare Liste mit Abschnittsbuchstaben

ScrollView {

LazyVStack(alignment: .leading, spacing: 0) {

ForEach(grouped, id: \.0) { section **in**

// Abschnittsbuchstabe (A, B, C, …)

Text(section.0)

.font(.system(size: 20, weight: .bold))

.foregroundColor(.white.opacity(0.6))

.padding(.horizontal, 20)

.padding(.top, 14)

.padding(.bottom, 6)

// Rows

ForEach(section.1) { ex **in**

NavigationLink {

// Du hast bereits eine Detail-View für Übungen,

// sie funktioniert auch als Push-Ziel.

ExerciseDetailSheet(exerciseName: ex.name)

.navigationBarBackButtonHidden(**true**) // wir nutzen eigenen Back-Header dort

} label: {

ExerciseCatalogRow(exercise: ex)

}

.buttonStyle(.plain)

// Dezente Trennlinie wie im Screenshot

Divider()

.background(Color.white.opacity(0.08))

.padding(.leading, 96)

}

}

// Falls keine Treffer

**if** grouped.isEmpty {

Text("Keine Übungen gefunden")

.foregroundColor(.white.opacity(0.6))

.font(.system(size: 16))

.frame(maxWidth: .infinity)

.padding(.vertical, 40)

}

}

.padding(.bottom, 30)

}

}

// Overlay: neue Übung anlegen (kein Sheet, bleibt in derselben Ansicht)

**if** showCreateExerciseDialog {

Color.black.opacity(0.5)

.ignoresSafeArea()

.onTapGesture { showCreateExerciseDialog = **false** }

CreateExerciseDialog(isPresented: $showCreateExerciseDialog) { newName **in**

// Minimaler Flow: neue Übung im Katalog ergänzen

**let** new = Exercise(

name: newName,

category: "Nicht gesetzt",

type: "Nicht gesetzt",

iconName: "figure.strengthtraining.traditional"

)

allExercises.append(new)

// optional: direkt in die Suche übernehmen

searchText = newName

}

.transition(.scale.combined(with: .opacity))

}

}

.navigationBarHidden(**true**) // eigener Header

}

// Runde Header-Buttons

@ViewBuilder **private** **func** headerRoundButton(icon: String, action: **@escaping** () -> Void) -> **some** View {

Button(action: action) {

Image(systemName: icon)

.font(.system(size: 18, weight: .semibold))

.foregroundColor(.white)

.frame(width: 44, height: 44)

.background(RoundedRectangle(cornerRadius: 22).fill(Color.white.opacity(0.15)))

}

}

@ViewBuilder **private** **func** headerCircle(\_ system: String, action: **@escaping** () -> Void) -> **some** View {

Button(action: action) {

ZStack {

Circle().fill(Color.white.opacity(0.15)).frame(width: 44, height: 44)

Image(systemName: system).font(.system(size: 16, weight: .semibold)).foregroundColor(.white)

}

}

}

}

// Einzelne Zeile wie im Screenshot (Avatar, Titel, Unterzeile, Chevron)

**private** **struct** ExerciseCatalogRow: View {

**let** exercise: Exercise

// Farbpalette für Avatare – leicht variierend

**private** **let** palette: [Color] = [

Color.pink.opacity(0.30),

Color.mint.opacity(0.35),

Color.yellow.opacity(0.30),

Color.purple.opacity(0.30),

Color.cyan.opacity(0.30)

]

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

HStack(spacing: 16) {

ExerciseAvatar(

title: exercise.name,

systemIcon: exercise.iconName,

size: 60,

bg: palette[abs(exercise.name.hashValue) % palette.count]

)

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

Text(exercise.name)

.font(.system(size: 17))

.foregroundColor(.white)

Text("\(exercise.category) — \(exercise.type)")

.font(.system(size: 15))

.foregroundColor(.white.opacity(0.6))

}

Spacer()

Image(systemName: "chevron.right")

.font(.system(size: 16, weight: .semibold))

.foregroundColor(.white.opacity(0.3))

}

.padding(.horizontal, 20)

.padding(.vertical, 12)

.background(Color.black)

}

}

// Versorgung mit Beispiel-/Startdaten für den Katalog

**private** **enum** ExerciseCatalog {

**static** **let** defaultExercises: [Exercise] = [

// A

Exercise(name: "Ausstoßen (Split Jerk)", category: "Ganzkörper", type: "Langhantel", iconName: "figure.strengthtraining.traditional"),

// B

Exercise(name: "Ball Slams", category: "Ganzkörper", type: "Medizinball", iconName: "figure.cross.training"),

Exercise(name: "Band Pullaparts", category: "Schultern", type: "Bänder", iconName: "figure.cooldown"),

Exercise(name: "Bankdrücken", category: "Brust", type: "Langhantel", iconName: "dumbbell"),

Exercise(name: "Bankdrücken (Enger Griff)", category: "Trizeps", type: "Langhantel", iconName: "dumbbell"),

Exercise(name: "Bankdrücken (Kurzhantel)", category: "Brust", type: "Kurzhantel", iconName: "dumbbell"),

Exercise(name: "Bankdrücken (Smith Machine)", category: "Brust", type: "Smith Machine (Multipresse)", iconName: "dumbbell"),

Exercise(name: "Bankdrücken (Weiter Griff)", category: "Brust", type: "Langhantel", iconName: "dumbbell"),

Exercise(name: "Bauchtrainer Ab-Roller", category: "Bauch", type: "andere", iconName: "figure.core.training"),

Exercise(name: "Beinbeuger (Beincurls)", category: "Beinbeuger", type: "Maschine", iconName: "figure.strengthtraining.traditional"),

Exercise(name: "Beinheben im Hang", category: "Bauch", type: "Körper", iconName: "figure.core.training"),

Exercise(name: "Beinpresse", category: "Quadrizeps", type: "Maschine", iconName: "figure.strengthtraining.traditional"),

Exercise(name: "Beinpresse sitzend", category: "Quadrizeps", type: "Maschine", iconName: "figure.strengthtraining.traditional"),

Exercise(name: "Beinstrecken", category: "Quadrizeps", type: "Maschine", iconName: "figure.strengthtraining.traditional"),

Exercise(name: "Bicycle Crunch", category: "Bauch", type: "Körper", iconName: "figure.core.training"),

Exercise(name: "Bizepscurl (Kurzhantel)", category: "Bizeps", type: "Kurzhantel", iconName: "dumbbell"),

Exercise(name: "Bizepscurl (Maschine)", category: "Bizeps", type: "Maschine", iconName: "figure.strengthtraining.traditional"),

Exercise(name: "Bizepscurl im Obergriff (Langhantel)", category: "Bizeps", type: "Langhantel", iconName: "dumbbell"),

Exercise(name: "Bizepscurl (EZ-Hantel)", category: "Bizeps", type: "EZ Curlstange", iconName: "dumbbell"),

Exercise(name: "Bizepscurl (Kabel)", category: "Bizeps", type: "Kabel", iconName: "figure.pullup"),

Exercise(name: "Box Squat", category: "Quadrizeps", type: "Langhantel", iconName: "dumbbell"),

Exercise(name: "Brustpresse (Band)", category: "Brust", type: "Bänder", iconName: "figure.cooldown"),

Exercise(name: "Brustpresse (Kabel)", category: "Brust", type: "Kabel", iconName: "figure.pullup"),

Exercise(name: "Brustpresse (Maschine)", category: "Brust", type: "Maschine", iconName: "figure.strengthtraining.traditional"),

Exercise(name: "Bulgarischer Squat", category: "Quadrizeps", type: "Kurzhantel", iconName: "dumbbell"),

Exercise(name: "Burpee", category: "Ganzkörper", type: "Körper", iconName: "figure.run"),

// C

Exercise(name: "Cable Flys", category: "Brust", type: "Kabel", iconName: "figure.pullup"),

Exercise(name: "Cuban Rotation", category: "Schultern", type: "Kurzhantel", iconName: "dumbbell"),

// … bei Bedarf beliebig erweitern

]

}

// ==========================================================

// 2) ERSETZEN: Einstellungen → „Übungen“ öffnet die neue Ansicht

// (kompletter SettingsViewScreen)

// ==========================================================

**struct** SettingsViewScreen: View {

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

NavigationStack {

ZStack {

Color.black.ignoresSafeArea()

List {

// Pro Banner

Section {

Button(action: {}) {

HStack(spacing: 16) {

ZStack {

RoundedRectangle(cornerRadius: 12)

.fill(Color.yellow.opacity(0.2))

.frame(width: 44, height: 44)

Image(systemName: "dumbbell.fill")

.font(.system(size: 20, weight: .bold))

.foregroundColor(.yellow)

}

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

Text("PowerGym pro")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white)

Text("Schalten Sie das volle Erlebnis frei")

.font(.system(size: 15))

.foregroundColor(.white.opacity(0.6))

}

Spacer()

}

.padding(.vertical, 8)

}

}

.listRowBackground(Color(red: 0.1, green: 0.1, blue: 0.1))

// Abschnitt 1: Training

Section {

NavigationLink(destination: Text("Timers Details")) {

SettingsNavigationRow(iconName: "timer", iconColor: .blue, title: "Timers")

}

NavigationLink(destination: Text("Equipment Details")) {

SettingsNavigationRow(iconName: "slider.horizontal.3", iconColor: .gray, title: "Equipment")

}

// HIER: auf die neue nicht-modale Seite navigieren

NavigationLink(destination: ExercisesCatalogView()) {

SettingsNavigationRow(iconName: "figure.strengthtraining.traditional", iconColor: .green, title: "Übungen")

}

NavigationLink(destination: Text("Aufwärm-Rechner Details")) {

SettingsNavigationRow(iconName: "calculator", iconColor: .orange, title: "Aufwärm-Rechner")

}

NavigationLink(destination: Text("Plattenrechner Details")) {

SettingsNavigationRow(iconName: "circle.grid.3x3", iconColor: .blue.opacity(0.7), title: "Plattenrechner")

}

NavigationLink(destination: Text("Einheiten Details")) {

SettingsNavigationRow(iconName: "ruler", iconColor: .purple, title: "Einheiten", subtitle: "kg")

}

NavigationLink(destination: Text("Apple Watch Details")) {

SettingsNavigationRow(iconName: "applewatch", iconColor: .red, title: "Apple Watch")

}

}

.listRowBackground(Color(red: 0.1, green: 0.1, blue: 0.1))

// Abschnitt 2: Allgemein

Section {

NavigationLink(destination: Text("Allgemein Details")) {

SettingsNavigationRow(iconName: "gear", iconColor: .gray, title: "Allgemein")

}

NavigationLink(destination: Text("Mitteilungen Details")) {

SettingsNavigationRow(iconName: "bell", iconColor: .red, title: "Mitteilungen")

}

NavigationLink(destination: Text("Integrationen Details")) {

SettingsNavigationRow(iconName: "link", iconColor: .blue, title: "Integrationen")

}

}

.listRowBackground(Color(red: 0.1, green: 0.1, blue: 0.1))

// Abschnitt 3: App

Section {

NavigationLink(destination: Text("Aussehen Details")) {

SettingsNavigationRow(iconName: "sparkles", iconColor: .pink, title: "Aussehen")

}

NavigationLink(destination: Text("Datenschutz Details")) {

SettingsNavigationRow(iconName: "hand.raised", iconColor: .blue, title: "Datenschutz")

}

NavigationLink(destination: Text("Import Export Details")) {

SettingsNavigationRow(iconName: "doc.text.image", iconColor: .green, title: "Import Export")

}

NavigationLink(destination: Text("iCloud-Backup Details")) {

SettingsNavigationRow(iconName: "icloud", iconColor: .blue.opacity(0.5), title: "iCloud-Backup")

}

}

.listRowBackground(Color(red: 0.1, green: 0.1, blue: 0.1))

// Abschnitt 4: Support

Section {

NavigationLink(destination: Text("Bewerten Details")) {

SettingsNavigationRow(iconName: "star", iconColor: .yellow, title: "PowerGym Bewerten")

}

NavigationLink(destination: Text("Feedback Details")) {

SettingsNavigationRow(iconName: "envelope", iconColor: .blue, title: "Feedback senden")

}

NavigationLink(destination: Text("Subreddit Details")) {

SettingsNavigationRow(iconName: "bubble.left.and.bubble.right", iconColor: .orange, title: "Offizielles Subreddit")

}

NavigationLink(destination: Text("Über Details")) {

SettingsNavigationRow(iconName: "info.circle", iconColor: .blue, title: "Über")

}

}

.listRowBackground(Color(red: 0.1, green: 0.1, blue: 0.1))

}

.listStyle(.insetGrouped)

.scrollContentBackground(.hidden)

.navigationTitle("Einstellungen")

.navigationBarTitleDisplayMode(.large)

}

}

}

}

// --- ENDE ÄNDERUNG ---

// **MARK: - Extensions**

**extension** Collection {

**subscript** (safe index: Index) -> Element? { indices.contains(index) ? **self**[index] : **nil** }

}

**extension** Color { **static** **let** cardBackground = Color(red: 0.10, green: 0.13, blue: 0.14) }

// **MARK: - Change Activity Dialog**

/// A dialog that allows the user to choose a new activity type for a routine.

/// Displays the available activity types with their icons and descriptions and highlights the currently selected one.

**struct** ChangeActivityDialog: View {

/// Controls whether the dialog is presented.

@Binding **var** isPresented: Bool

/// The currently selected activity type for the routine. This binding will be updated when the user selects a new type.

@Binding **var** selectedActivity: RoutineActivityType

/// Called when the user selects a new activity type. Use this to perform additional actions, such as updating parent state.

**let** onSelect: (RoutineActivityType) -> Void

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

ZStack {

// Semi-transparent backdrop for focus

Color.black.opacity(0.45).ignoresSafeArea()

VStack(spacing: 0) {

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

Button {

selectedActivity = type

onSelect(type)

isPresented = **false**

} label: {

HStack(alignment: .center, spacing: 12) {

Image(systemName: type.icon)

.font(.system(size: 20))

.foregroundColor(.white)

.frame(width: 32, height: 32)

.background(Circle().fill(Color.white.opacity(0.1)))

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

Text(type.rawValue)

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.white)

Text(type.description)

.font(.system(size: 13))

.foregroundColor(.white.opacity(0.6))

.fixedSize(horizontal: **false**, vertical: **true**)

}

Spacer()

**if** selectedActivity == type {

Image(systemName: "checkmark")

.font(.system(size: 17, weight: .semibold))

.foregroundColor(.pink)

}

}

.padding(.horizontal, 20)

.padding(.vertical, 14)

}

**if** type != RoutineActivityType.allCases.last {

Divider()

.background(Color.white.opacity(0.1))

.padding(.leading, 20)

}

}

}

.background(

RoundedRectangle(cornerRadius: 24)

.fill(Color(red: 0.12, green: 0.12, blue: 0.12))

.overlay(RoundedRectangle(cornerRadius: 24).stroke(Color.white.opacity(0.08), lineWidth: 1))

)

.padding(.horizontal, 32)

}

}

}

// **MARK: - Preview**

#Preview {

ContentView()

.environmentObject(TrainingViewModel()) // Add environment object for preview

}