**Final Capstone Project IOS**

**BookMyslot\_Interviewer**

**Github link:** [**https://github.com/MdMastan/Finalcapstoneproject.git**](https://github.com/MdMastan/Finalcapstoneproject.git)

**BookMySlot\_InterviewerApp**

import SwiftUI

@main

struct BookMySlot\_InterviewerApp: App {

let persistenceController = PersistenceController.shared

var body: some Scene {

WindowGroup {

ContentView()

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

}

}

}

**ContentView**

import SwiftUI

struct ContentView: View {

@State private var isUserLoggedIn = false

var body: some View {

if isUserLoggedIn {

DashboardView(isUserLoggedIn: $isUserLoggedIn)

} else {

LoginView(isUserLoggedIn: $isUserLoggedIn)

}

}

}

struct ContentView\_Previews: PreviewProvider {

static var previews: some View {

ContentView()

}

}

**LoginView**

import SwiftUI

struct LoginView: View {

@Environment(\.managedObjectContext) private var viewContext

@FetchRequest(entity: User.entity(), sortDescriptors: []) private var users: FetchedResults<User>

@State private var username = ""

@State private var password = ""

@State private var isAuthenticated = false

@State private var showError = false

@Binding var isUserLoggedIn: Bool

var body: some View {

NavigationView {

VStack {

Text("Welcome to BookMySlotAPP")

.padding()

Text("Login")

.font(.largeTitle)

.padding()

TextField("Username", text: $username)

.textFieldStyle(RoundedBorderTextFieldStyle())

.padding()

SecureField("Password", text: $password)

.textFieldStyle(RoundedBorderTextFieldStyle())

.padding()

if showError {

Text("Invalid username or password!")

.foregroundColor(.red)

}

Button("Login") {

authenticateUser()

}

.buttonStyle(.borderedProminent)

.padding()

NavigationLink("Want to Register? Click here ", destination: RegisterView(isUserLoggedIn: $isUserLoggedIn))

.padding()

}

.fullScreenCover(isPresented: $isAuthenticated) {

DashboardView(isUserLoggedIn: $isAuthenticated)

}

}

}

private func authenticateUser() {

if users.contains(where: { $0.username == username && $0.password == password }) {

isAuthenticated = true

} else {

showError = true

}

}

}

struct LoginView\_Previews: PreviewProvider {

static var previews: some View {

LoginView(isUserLoggedIn: .constant(false))

}

}

**RegisterView**

import SwiftUI

import CoreData

struct RegisterView: View {

@Environment(\.managedObjectContext) private var viewContext

@State private var username = ""

@State private var password = ""

@State private var email = ""

@State private var registrationSuccess = false

@Binding var isUserLoggedIn: Bool

var body: some View {

NavigationView {

VStack {

Text("Register")

.font(.largeTitle)

.padding()

TextField("Username", text: $username)

.textFieldStyle(RoundedBorderTextFieldStyle())

.padding()

TextField("Email", text: $email)

.textFieldStyle(RoundedBorderTextFieldStyle())

.padding()

.keyboardType(.emailAddress)

SecureField("Password", text: $password)

.textFieldStyle(RoundedBorderTextFieldStyle())

.padding()

Button("Register") {

registerUser()

}

.buttonStyle(.borderedProminent)

.padding()

if registrationSuccess {

Text("Registration successful!")

.foregroundColor(.green)

.padding()

NavigationLink("Go to Login", destination: LoginView(isUserLoggedIn: $isUserLoggedIn))

.padding()

}

}

.padding()

}

}

private func registerUser() {

guard !username.isEmpty, !password.isEmpty, !email.isEmpty else {

print("All fields are required!")

return

}

let newUser = User(context: viewContext)

newUser.username = username

newUser.password = password

newUser.email = email

do {

try viewContext.save()

registrationSuccess = true

clearFields()

} catch {

print("Failed to register user: \(error.localizedDescription)")

}

}

private func clearFields() {

username = ""

password = ""

email = ""

}

}

struct RegisterView\_Previews: PreviewProvider {

static var previews: some View {

RegisterView(isUserLoggedIn: .constant(false))

}

}

**DashboardView**

import SwiftUI

import CoreData

struct DashboardView: View {

@Environment(\.managedObjectContext) private var viewContext

@FetchRequest(

entity: InterviewSlot.entity(),

sortDescriptors: [NSSortDescriptor(keyPath: \InterviewSlot.date, ascending: true)]

) private var slots: FetchedResults<InterviewSlot>

@State private var selectedSlot: InterviewSlot?

@State private var showEditView = false

@State private var showAddSlotView = false

@Binding var isUserLoggedIn: Bool

var body: some View {

NavigationView {

VStack {

Text("Interview Slots")

.font(.largeTitle)

.bold()

.padding()

List {

ForEach(slots, id: \.self) { slot in

VStack(alignment: .leading) {

Text("Date: \(formattedDate(slot.date))")

.font(.headline)

Text("Time: \(slot.fromTime ?? "N/A") - \(slot.uptoTime ?? "N/A")")

.foregroundColor(.gray)

Text("Interviewer: \(slot.interviewerName ?? "Not Assigned")")

.foregroundColor(.blue)

Text("Subject: \(slot.subject ?? "N/A")")

.foregroundColor(.purple)

Text("Status: \(slot.status ?? "Unknown")")

.foregroundColor(statusColor(slot.status ?? "Unknown"))

}

.padding()

.swipeActions {

Button {

selectedSlot = slot

showEditView.toggle()

} label: {

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

}

.tint(.blue)

Button(role: .destructive) {

deleteSlot(slot)

} label: {

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

}

}

}

}

Spacer()

Button(action: { showAddSlotView.toggle() }) {

Text("Add Slot")

.frame(maxWidth: .infinity)

.padding()

.background(Color.blue)

.foregroundColor(.white)

.cornerRadius(10)

}

.padding()

}

.sheet(isPresented: $showAddSlotView) {

AddSlotView()

}

.sheet(item: $selectedSlot) { slot in

EditSlotView(slot: slot)

}

.navigationBarItems(

leading: Button(action: logout) {

Text("Logout")

.foregroundColor(.red)

},

trailing: EditButton()

)

}

}

private func deleteSlot(\_ slot: InterviewSlot) {

withAnimation {

viewContext.delete(slot)

do {

try viewContext.save()

} catch {

print("Error deleting slot: \(error.localizedDescription)")

}

}

}

private func formattedDate(\_ date: Date?) -> String {

guard let date = date else { return "N/A" }

let formatter = DateFormatter()

formatter.dateStyle = .medium

return formatter.string(from: date)

}

private func statusColor(\_ status: String) -> Color {

switch status {

case "Available": return .green

case "Booked": return .orange

case "Completed": return .gray

default: return .black

}

}

private func logout() {

isUserLoggedIn = false

}

}

**AddSlotView**

import SwiftUI

struct AddSlotView: View {

@Environment(\.managedObjectContext) private var viewContext

@Environment(\.presentationMode) var presentationMode

@State private var date = Date()

@State private var fromTime = ""

@State private var uptoTime = ""

@State private var interviewerName = ""

@State private var subject = ""

@State private var status = "Available"

let statusOptions = ["Available", "Booked", "Completed"]

var body: some View {

NavigationView {

Form {

DatePicker("Select Date", selection: $date, displayedComponents: .date)

TextField("Start Time (HH:MM AM/PM)", text: $fromTime)

TextField("End Time (HH:MM AM/PM)", text: $uptoTime)

TextField("Interviewer Name", text: $interviewerName)

TextField("Interview Subject", text: $subject)

Picker("Status", selection: $status) {

ForEach(statusOptions, id: \.self) {

Text($0)

}

}

.pickerStyle(SegmentedPickerStyle())

HStack {

Button("Cancel") {

presentationMode.wrappedValue.dismiss()

}

.frame(maxWidth: .infinity)

.padding()

.background(Color.gray)

.foregroundColor(.white)

.cornerRadius(10)

Button("Add Slot") {

addSlot()

}

.frame(maxWidth: .infinity)

.padding()

.background(Color.blue)

.foregroundColor(.white)

.cornerRadius(10)

}

}

.navigationTitle("Add Slot")

}

}

private func addSlot() {

let newSlot = InterviewSlot(context: viewContext)

newSlot.date = date

newSlot.fromTime = fromTime

newSlot.uptoTime = uptoTime

newSlot.interviewerName = interviewerName

newSlot.subject = subject

newSlot.status = status

do {

try viewContext.save()

presentationMode.wrappedValue.dismiss()

} catch {

print("Error saving slot: \(error.localizedDescription)")

}

}

}

**EditSlotView**

import SwiftUI

import CoreData

struct EditSlotView: View {

@Environment(\.managedObjectContext) private var viewContext

@Environment(\.presentationMode) var presentationMode

@ObservedObject var slot: InterviewSlot

var body: some View {

NavigationView {

VStack {

Form {

Section(header: Text("Select Date")) {

DatePicker("Date", selection: Binding(

get: { slot.date ?? Date() },

set: { slot.date = $0 }

), displayedComponents: .date)

}

Section(header: Text("Time")) {

HStack {

Text("From:")

TextField("Start Time", text: Binding(

get: { slot.fromTime ?? "" },

set: { slot.fromTime = $0 }

))

.textFieldStyle(RoundedBorderTextFieldStyle())

}

HStack {

Text("To:")

TextField("End Time", text: Binding(

get: { slot.uptoTime ?? "" },

set: { slot.uptoTime = $0 }

))

.textFieldStyle(RoundedBorderTextFieldStyle())

}

}

Section(header: Text("Interviewer")) {

TextField("Name", text: Binding(

get: { slot.interviewerName ?? "" },

set: { slot.interviewerName = $0 }

))

}

Section(header: Text("Subject")) {

TextField("Subject", text: Binding(

get: { slot.subject ?? "" },

set: { slot.subject = $0 }

))

}

Section(header: Text("Status")) {

Picker("Status", selection: Binding(

get: { slot.status ?? "Available" },

set: { slot.status = $0 }

)) {

Text("Available").tag("Available")

Text("Booked").tag("Booked")

Text("Completed").tag("Completed")

}

.pickerStyle(SegmentedPickerStyle())

}

}

HStack {

Button("Cancel") {

presentationMode.wrappedValue.dismiss()

}

.frame(maxWidth: .infinity)

.padding()

.background(Color.gray.opacity(0.2))

.cornerRadius(8)

Button("Delete") {

deleteSlot()

}

.frame(maxWidth: .infinity)

.padding()

.background(Color.red)

.foregroundColor(.white)

.cornerRadius(8)

Button("Save Changes") {

saveChanges()

}

.frame(maxWidth: .infinity)

.padding()

.background(Color.blue)

.foregroundColor(.white)

.cornerRadius(8)

}

.padding()

}

.navigationTitle("Edit Slot")

}

}

private func deleteSlot() {

withAnimation {

viewContext.delete(slot)

do {

try viewContext.save()

presentationMode.wrappedValue.dismiss()

} catch {

print("Error deleting slot: \(error.localizedDescription)")

}

}

}

private func saveChanges() {

do {

try viewContext.save()

presentationMode.wrappedValue.dismiss()

} catch {

print("Error saving slot: \(error.localizedDescription)")

}

}

}





Appium Testing For IOS

APP: BOOKMYSLOT\_INTERVIWER

Baseclass testing

import io.appium.java\_client.ios.IOSDriver;

import io.appium.java\_client.remote.MobileCapabilityType;

import org.openqa.selenium.remote.DesiredCapabilities;

import java.net.MalformedURLException;

import java.net.URL;

import java.time.Duration;

public class BaseTest {

protected static IOSDriver driver;

public static void setup() throws MalformedURLException {

DesiredCapabilities caps = new DesiredCapabilities();

caps.setCapability(MobileCapabilityType.PLATFORM\_NAME, "iOS");

caps.setCapability(MobileCapabilityType.PLATFORM\_VERSION, "17.0");

caps.setCapability(MobileCapabilityType.DEVICE\_NAME, "iPhone 14 Pro");

caps.setCapability(MobileCapabilityType.AUTOMATION\_NAME, "XCUITest");

caps.setCapability(MobileCapabilityType.APP, "/Users/yourusername/path-to-your-app.app");

caps.setCapability("noReset", true);

driver = new IOSDriver(new URL("http://127.0.0.1:4723/"), caps);

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

}

public static void tearDown() {

if (driver != null) {

driver.quit();

}

}

}

Login class testing

import org.openqa.selenium.By;

import org.testng.Assert;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class LoginTest extends BaseTest {

@BeforeClass

public void setUp() throws Exception {

setup();

}

@Test

public void testLogin() {

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='Username']")).sendKeys("testuser");

driver.findElement(By.xpath("//XCUIElementTypeSecureTextField[@name='Password']")).sendKeys("password123");

driver.findElement(By.xpath("//XCUIElementTypeButton[@name='Login']")).click();

String successMessage = driver.findElement(By.xpath("//XCUIElementTypeStaticText[@name='Welcome']")).getText();

Assert.assertEquals(successMessage, "Welcome");

}

@AfterClass

public void tearDown() {

tearDown();

}

}

Registrationclass test-

import org.openqa.selenium.By;

import org.testng.Assert;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class RegisterTest extends BaseTest {

@BeforeClass

public void setUp() throws Exception {

setup();

}

@Test

public void testUserRegistration() {

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='Username']")).sendKeys("testuser");

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='Email']")).sendKeys("testuser@example.com");

driver.findElement(By.xpath("//XCUIElementTypeSecureTextField[@name='Password']")).sendKeys("password123");

driver.findElement(By.xpath("//XCUIElementTypeButton[@name='Register']")).click();

String successMessage = driver.findElement(By.xpath("//XCUIElementTypeStaticText[@name='Registration successful!']")).getText();

Assert.assertEquals(successMessage, "Registration successful!");

}

@AfterClass

public void tearDown() {

tearDown();

}

}

AddSlot class testing:

import org.openqa.selenium.By;

import org.testng.Assert;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class AddSlotTest extends BaseTest {

@BeforeClass

public void setUp() throws Exception {

setup();

}

@Test

public void testAddSlot() {

driver.findElement(By.xpath("//XCUIElementTypeButton[@name='Add Slot']")).click();

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='Start Time']")).sendKeys("10:00 AM");

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='End Time']")).sendKeys("11:00 AM");

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='Interviewer Name']")).sendKeys("John Doe");

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='Interview Subject']")).sendKeys("Swift Interview");

driver.findElement(By.xpath("//XCUIElementTypeButton[@name='Add Slot']")).click();

String successMessage = driver.findElement(By.xpath("//XCUIElementTypeStaticText[@name='Slot added successfully!']")).getText();

Assert.assertEquals(successMessage, "Slot added successfully!");

}

@AfterClass

public void tearDown() {

tearDown();

}

}

EditSlot class testing:  
  
import org.openqa.selenium.By;

import org.testng.Assert;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class EditSlotTest extends BaseTest {

@BeforeClass

public void setUp() throws Exception {

setup();

}

@Test

public void testEditSlot() {

driver.findElement(By.xpath("//XCUIElementTypeButton[@name='Edit Slot']")).click();

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='Interviewer Name']")).clear();

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='Interviewer Name']")).sendKeys("Jane Smith");

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='Interview Subject']")).clear();

driver.findElement(By.xpath("//XCUIElementTypeTextField[@name='Interview Subject']")).sendKeys("iOS Development");

driver.findElement(By.xpath("//XCUIElementTypeButton[@name='Save Changes']")).click();

String successMessage = driver.findElement(By.xpath("//XCUIElementTypeStaticText[@name='Changes saved successfully!']")).getText();

Assert.assertEquals(successMessage, "Changes saved successfully!");

}

@AfterClass

public void tearDown() {

tearDown();

}

}