```
// File: app\src\androidTest\java\com\example\androidbloodbank\ExampleInstrumentedTest.kt

package com.example.androidbloodbank

import androidx.test.platform.app.InstrumentationRegistry
import androidx.test.ext.junit.runners.AndroidJUnit4

import org.junit.Test
import org.junit.runner.RunWith

import org.junit.Assert.*

/**
 * Instrumented test, which will execute on an Android device.
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        assertEquals("com.example.androidbloodbank", appContext.packageName)
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\MainActivity.kt

package com.example.androidbloodbank

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.runtime.remember
import androidx.navigation.compose.rememberNavController
import com.example.androidbloodbank.data.LocalRepo
import com.example.androidbloodbank.navigation.AppNavHost
import com.example.androidbloodbank.ui.theme.AndroidBloodBankTheme // <-- import this

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            AndroidBloodBankTheme {
                val navController = rememberNavController()
                val repo = remember { LocalRepo(this) }
                AppNavHost(navController, repo) { msg -> println("ALERT: $msg") }
            }
        }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\data\BloodRepository.kt

package com.example.androidbloodbank.data

import com.example.androidbloodbank.data.model.BloodRequest
import com.example.androidbloodbank.data.model.UserProfile

class BloodRepository {

    private val requests = mutableListOf<BloodRequest>()
    private var userProfile: UserProfile? = null

    // --- Blood Requests ---
    fun addRequest(request: BloodRequest) {
        requests.add(request)
    }

    fun getRequests(): List<BloodRequest> = requests

    // --- User Profile ---
    fun setUserProfile(profile: UserProfile) {
        userProfile = profile
    }

    fun getUserProfile(): UserProfile? = userProfile

    fun updateProfile(updated: UserProfile) {
        userProfile = updated
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\data\LocalRepo.kt

package com.example.androidbloodbank.data

import android.content.Context
import android.content.SharedPreferences
import com.example.androidbloodbank.data.model.*
import com.google.gson.Gson
import com.google.gson.reflect.TypeToken

class LocalRepo(context: Context) {
    private val prefs: SharedPreferences =
        context.getSharedPreferences("abb_prefs", Context.MODE_PRIVATE)
    private val gson = Gson()

    companion object {
        private const val KEY_DONORS = "donors_json"
        private const val KEY_USERS = "users_json"
        private const val KEY_SCHEDULES = "schedules_json"
        private const val KEY_REQUESTS = "blood_requests_json"

        // session/profile snippets
        private const val KEY_CURRENT_USER_JSON = "current_user_json"
        private const val KEY_MY_DONOR = "my_donor_json"
    }

    private fun <T> fromJson(json: String?, typeToken: TypeToken<T>): T? {
        if (json.isNullOrBlank()) return null
        return gson.fromJson(json, typeToken.type)
    }
    private fun toJson(value: Any?): String = gson.toJson(value)

    // ---- Donors ----
    fun loadDonors(): MutableList<Donor> {
        val json = prefs.getString(KEY_DONORS, null)
        return fromJson(json, object : TypeToken<MutableList<Donor>>() {}) ?: defaultDonors().toMutableList()
    }
    fun saveDonors(list: List<Donor>) {
        prefs.edit().putString(KEY_DONORS, toJson(list)).apply()
    }

    // ---- Users ----
    fun loadUsers(): MutableList<User> {
        val json = prefs.getString(KEY_USERS, null)
        return fromJson(json, object : TypeToken<MutableList<User>>() {}) ?: mutableListOf()
    }
    fun saveUsers(list: List<User>) {
        prefs.edit().putString(KEY_USERS, toJson(list)).apply()
    }

    // ---- Schedules (unchanged) ----
    fun loadSchedules(): MutableList<Schedule> {
        val json = prefs.getString(KEY_SCHEDULES, null)
        return fromJson(json, object : TypeToken<MutableList<Schedule>>() {}) ?: mutableListOf()
    }
```

```
    fun saveSchedules(list: List<Schedule>) {
        prefs.edit().putString(KEY_SCHEDULES, toJson(list)).apply()
    }

    // ---- Blood Requests (NEW) ----
    fun loadRequests(): MutableList<BloodRequest> {
        val json = prefs.getString(KEY_REQUESTS, null)
        return fromJson(json, object : TypeToken<MutableList<BloodRequest>>() {}) ?: mutableListOf()
    }
    fun saveRequests(list: List<BloodRequest>) {
        prefs.edit().putString(KEY_REQUESTS, toJson(list)).apply()
    }

    // ---- Small session/profile helpers (used by Login/Profile screens) ----
    fun loadCurrentUserJson(): String? = prefs.getString(KEY_CURRENT_USER_JSON, null)
    fun saveCurrentUserJson(json: String?) {
        if (json == null) prefs.edit().remove(KEY_CURRENT_USER_JSON).apply()
        else prefs.edit().putString(KEY_CURRENT_USER_JSON, json).apply()
    }
    fun logoutCurrentUser() {
        prefs.edit().remove(KEY_CURRENT_USER_JSON).apply()
    }
    fun loadMyDonor(): Donor? {
        val json = prefs.getString(KEY_MY_DONOR, null) ?: return null
        return fromJson(json, object : TypeToken<Donor>() {})
    }
    fun saveMyDonor(d: Donor) {
        prefs.edit().putString(KEY_MY_DONOR, toJson(d)).apply()
    }

    // ---- Seed ----
    private fun defaultDonors() = listOf(
        Donor(name = "Rahim Uddin", bloodGroup = BloodGroup.O_NEG, phone = "01710000001", verified = true),
        Donor(name = "Ayesha Khan", bloodGroup = BloodGroup.A_POS, phone = "01710000002", verified = true),
        Donor(name = "Jahangir", bloodGroup = BloodGroup.B_NEG, phone = "01710000003"),
        Donor(name = "Mina Sultana", bloodGroup = BloodGroup.AB_NEG, phone = "01710000004"),
        Donor(name = "Sohan", bloodGroup = BloodGroup.O_POS, phone = "01710000005", verified = true),
    )
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\data\model\BloodGroup.kt

package com.example.androidbloodbank.data.model

enum class BloodGroup(val label: String) {
    A_POS("A+"), A_NEG("A-"),
    B_POS("B+"), B_NEG("B-"),
    AB_POS("AB+"), AB_NEG("AB-"),
    O_POS("O+"), O_NEG("O-");

    override fun toString(): String = label
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\data\model\BloodRequest.kt

package com.example.androidbloodbank.data.model

enum class RequestStatus { PENDING, MATCHED, CLOSED }

data class BloodRequest(
    val requesterName: String,
    val hospital: String,
    val location: String,
    val contactNumber: String,
    val bloodGroup: String,
    val neededDateMillis: Long = 0L,
    val timestamp: Long = 0L,
    val id: String? = null,
    val status: RequestStatus = RequestStatus.PENDING // NEW
)
```

```
// File: app\src\main\java\com\example\androidbloodbank\data\model\Donor.kt

package com.example.androidbloodbank.data.model

import java.util.UUID

data class Donor(
    val id: String = UUID.randomUUID().toString(),
    val name: String,
    val bloodGroup: BloodGroup,
    val phone: String? = null,
    val verified: Boolean = false,

    // NEW optional fields (so old data still works)
    val age: Int? = null,
    val city: String? = null,
    val area: String? = null,
    /** Epoch millis when this donor LAST donated. Null if unknown or new donor. */
    val lastDonationMillis: Long? = null,
    /** Preferred hospital/clinic or usual donation place */
    val hospital: String? = null
)
```

```
// File: app\src\main\java\com\example\androidbloodbank\data\model\Schedule.kt

package com.example.androidbloodbank.data.model

import java.util.UUID

data class Schedule(
    val id: String = UUID.randomUUID().toString(),
    val donorId: String,
    val dateIso: String,
    val notes: String? = null
)
```

```
// File: app\src\main\java\com\example\androidbloodbank\data\model\User.kt

package com.example.androidbloodbank.data.model

import java.util.UUID

data class User(
    val id: String = UUID.randomUUID().toString(),
    val name: String,
    val email: String,
    val bloodGroup: BloodGroup
)
```

```
// File: app\src\main\java\com\example\androidbloodbank\data\model\UserProfile.kt

package com.example.androidbloodbank.data.model

import java.util.concurrent.TimeUnit

data class UserProfile(
    var name: String,
    var bloodGroup: String,
    var lastDonationMillis: Long?,  // timestamp
    var totalDonations: Int,
    var contactNumber: String,
    var location: String
) {
    // Eligible if 90+ days passed
    val isEligible: Boolean
        get() = lastDonationMillis?.let {
            val diff = System.currentTimeMillis() - it
            val days = TimeUnit.MILLISECONDS.toDays(diff)
            days >= 90
        } ?: true

    // Remaining days until next donation
    val daysRemaining: Long
        get() = lastDonationMillis?.let {
            val nextEligible = it + TimeUnit.DAYS.toMillis(90)
            val diff = nextEligible - System.currentTimeMillis()
            if (diff > 0) TimeUnit.MILLISECONDS.toDays(diff) else 0
        } ?: 0
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\data\remote\FirebaseRepo.kt

package com.example.androidbloodbank.data.remote

import com.example.androidbloodbank.data.model.BloodGroup
import com.example.androidbloodbank.data.model.Donor
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ServerValue
import kotlinx.coroutines.tasks.await

/**
 * Realtime Database layout
 *
 * /users/{uid}/profile           -> private full profile (only this user can read/write)
 * /donors_public/{uid}           -> public donor card (visible to everyone for search)
 * /requests/{uid}/{requestId}    -> (optional) user's requests
 */
class FirebaseRepo {
    private val auth = FirebaseAuth.getInstance()
    private val db = FirebaseDatabase.getInstance().reference

    private fun uid(): String =
        auth.currentUser?.uid ?: throw IllegalStateException("Not logged in")

    // Map enum -> stable string for DB
    private fun BloodGroup.toCode(): String = name   // e.g. O_POS, A_NEG, ...

    // ---------- PROFILE UPSERT ----------
    suspend fun upsertProfile(
        name: String,
        email: String?,
        phone: String?,
        donor: Donor
    ) {
        val uid = uid()

        // Private profile (full)
        val profile = hashMapOf(
            "uid" to uid,
            "name" to name,
            "email" to email,
            "phone" to phone,
            "bloodGroup" to donor.bloodGroup.toCode(),
            "age" to donor.age,
            "city" to (donor.city ?: ""),
            "area" to (donor.area ?: ""),
            "hospital" to (donor.hospital ?: ""),
            "lastDonationMillis" to (donor.lastDonationMillis ?: 0L),
            "verified" to donor.verified,
            "updatedAt" to ServerValue.TIMESTAMP
        )

        // Public donor card (no sensitive fields)
        val public = hashMapOf(
```

```
            "uid" to uid,
            "name" to name,
            "bloodGroup" to donor.bloodGroup.toCode(),
            "city" to (donor.city ?: ""),
            "area" to (donor.area ?: ""),
            "lastDonationMillis" to (donor.lastDonationMillis ?: 0L),
            "verified" to donor.verified,
            "updatedAt" to ServerValue.TIMESTAMP
        )

        // Write both in a single multi-path update
        val updates = hashMapOf<String, Any>(
            "/users/$uid/profile" to profile,
            "/donors_public/$uid" to public
        )
        db.updateChildren(updates).await()
    }

    // ---------- LOAD PROFILE (optional helper) ----------
    suspend fun loadProfile(): Map<String, Any?>? {
        val snap = db.child("users").child(uid()).child("profile").get().await()
        return if (snap.exists()) (snap.value as? Map<String, Any?>) else null
    }

    // ---------- (Optional) Donors query by group ----------
    suspend fun listDonorsByGroup(groupCode: String): List<Map<String, Any?>> {
        // Simple query by child; add indexOn in rules for performance
        val q = db.child("donors_public").orderByChild("bloodGroup").equalTo(groupCode)
        val snap = q.get().await()
        if (!snap.exists()) return emptyList()
        return snap.children.mapNotNull { it.value as? Map<String, Any?> }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\navigation\AppNavHost.kt

package com.example.androidbloodbank.navigation

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.navigation.NavHostController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.currentBackStackEntryAsState
import com.example.androidbloodbank.data.LocalRepo

// Screens
import com.example.androidbloodbank.ui.screens.HomeScreen
import com.example.androidbloodbank.ui.screens.ProfileScreen
import com.example.androidbloodbank.ui.screens.RequestBloodScreen
import com.example.androidbloodbank.ui.LoginScreen
import com.example.androidbloodbank.ui.SignupScreen
import com.example.androidbloodbank.ui.SchedulesScreen
import com.example.androidbloodbank.ui.EmergencyScreen
import com.example.androidbloodbank.ui.flow.*

// Bottom bar

// Firebase
import com.google.firebase.auth.FirebaseAuth

@Composable
fun AppNavHost(
    navController: NavHostController,
    repo: LocalRepo,
    onAlert: (String) -> Unit = {}
) {
    val snackbarHostState = remember { SnackbarHostState() }

    // show bottom bar on main app sections only
    val backStack by navController.currentBackStackEntryAsState()
    val route = backStack?.destination?.route
    val showBottomBar = when {
        route == null -> false
        route.startsWith(Route.Splash.path) -> false
        route.startsWith(Route.Gate.path)   -> false
        route.startsWith(Route.SignIn.path) -> false
        route.startsWith(Route.SignUp.path) -> false
        else -> true
    }

    Scaffold(
        snackbarHost = { SnackbarHost(hostState = snackbarHostState) },
        bottomBar = { if (showBottomBar) BottomNavBar(navController) }
    ) { padding ->
```

```
NavHost(
    navController = navController,
    startDestination = Route.Splash.path,
    modifier = Modifier.padding(padding)
) {
    // Splash — decide start based on Firebase or local session
    composable(Route.Splash.path) {
        val auth = remember { FirebaseAuth.getInstance() }
        val hasLocal = remember { repo.loadCurrentUserJson() != null }
        LaunchedEffect(Unit) {
            val dest = if (auth.currentUser != null || hasLocal) Route.Home.path else Route.Gate.path
            navController.navigate(dest) {
                popUpTo(Route.Splash.path) { inclusive = true }
                launchSingleTop = true
            }
        }
        Box(Modifier, contentAlignment = Alignment.Center) {
            CircularProgressIndicator()
        }
    }

    // Gate: ONLY Login, Sign up, Emergency SOS
    composable(Route.Gate.path) {
        val auth = remember { FirebaseAuth.getInstance() }
        // If already logged in, bounce to Home
        LaunchedEffect(auth.currentUser, repo.loadCurrentUserJson()) {
            if (auth.currentUser != null || repo.loadCurrentUserJson() != null) {
                navController.navigate(Route.Home.path) {
                    popUpTo(Route.Gate.path) { inclusive = true }
                    launchSingleTop = true
                }
            }
        }
        AccountGateScreen(
            onLogin = { navController.navigate(Route.SignIn.path) },
            onSignUp = { navController.navigate(Route.SignUp.path) },
            onEmergency = { navController.navigate(Route.EmergencySOS.path) }
        )
    }

    // Emergency SOS (offline)
    composable(Route.EmergencySOS.path) {
        EmergencySosScreen(repo = repo, onBack = { navController.popBackStack() })
    }

    // Auth
    composable(Route.SignIn.path) {
        LoginScreen(
            repo = repo,
            onBack = { navController.popBackStack() },
            onLoginSuccess = {
                navController.navigate(Route.Home.path) {
                    popUpTo(Route.Gate.path) { inclusive = true }
                    launchSingleTop = true
```

```
                }
            }
        )

    }
    composable(Route.SignUp.path) {
        SignupScreen(
            repo = repo,
            onBack = { navController.popBackStack() },
            onSignupSuccess = {
                navController.navigate(Route.Home.path) {
                    popUpTo(Route.Gate.path) { inclusive = true }
                    launchSingleTop = true
                }
            }
        )
    }

    // HOME
    composable(Route.Home.path) {
        HomeScreen(
            onDonate      = { navController.navigate(Route.Donate.path) },
            onFindDonors  = { navController.navigate(Route.FindDonors.path) },
            onBloodBank   = { navController.navigate(Route.BloodBank.path) },
            onRequestBlood = { navController.navigate(Route.RequestBlood.path) },
            onProfile     = { navController.navigate(Route.Profile.path) }
        )
    }

    // Donate flow
    composable(Route.Donate.path) {
        DonateScreen(
            onViewRequests = { navController.navigate(Route.ViewRequests.path) },
            onPostRequest  = { navController.navigate(Route.PostRequest.path) },
            onBack = { navController.popBackStack() }
        )
    }
    composable(Route.ViewRequests.path) { ViewRequestsScreen(repo = repo, onBack = { navController.popBackStack() }) }
        composable(Route.PostRequest.path)  { PostRequestScreen(repo = repo, onPosted = { navController.popBackStack() },
onBack = { navController.popBackStack() }) }

    // Find donors flow
    composable(Route.FindDonors.path) {
        FindDonorsScreen(
            repo = repo,
            onSelectBG = { navController.navigate(Route.SelectBloodGroup.path) },
            onOpenDonor = { id -> navController.navigate(Route.DonorProfile.create(id)) },
            onBack = { navController.popBackStack() }
        )
    }
     composable(Route.SelectBloodGroup.path) { SelectBloodGroupScreen(onDone = { navController.popBackStack() }, onBack
= { navController.popBackStack() }) }
    composable(Route.DonorProfile.path) { backStackEntry ->
        val donorId = backStackEntry.arguments?.getString(Route.DonorProfile.ArgKey) ?: return@composable
        DonorProfileScreen(donorId = donorId, repo = repo, onBack = { navController.popBackStack() })
```

```
        }

        // Blood bank flow
        composable(Route.BloodBank.path) {
            BloodBankScreen(
                onNearby    = { navController.navigate(Route.NearbyBloodBank.path) },
                onAvailable = { navController.navigate(Route.AvailableBlood.path) },
                onBack      = { navController.popBackStack() }
            )
        }
        composable(Route.NearbyBloodBank.path) { NearbyBloodBankScreen(onBack = { navController.popBackStack() }) }
        composable(Route.AvailableBlood.path) { AvailableBloodScreen(onBack = { navController.popBackStack() }) }

        // Request blood
        composable(Route.RequestBlood.path) {
            var snack by remember { mutableStateOf<String?>(null) }
            snack?.let { msg ->
                LaunchedEffect(msg) {
                    snackbarHostState.showSnackbar(message = msg, withDismissAction = true)
                    snack = null
                }
            }
            RequestBloodScreen(
                repo = repo,
                onBack = { navController.popBackStack() }
            )
        }
        composable(Route.TrackRequest.path) { TrackRequestScreen(repo = repo, onBack = { navController.popBackStack() }) }

        // Profile (with proper sign-out)
        composable(Route.Profile.path) {
            ProfileScreen(
                repo = repo,
                onBack = { navController.popBackStack() },
                onLoggedOut = {
                    // Ensure both sessions are cleared
                    runCatching { FirebaseAuth.getInstance().signOut() }
                    repo.logoutCurrentUser()
                    navController.navigate(Route.Gate.path) {
                        popUpTo(Route.Home.path) { inclusive = true }
                        launchSingleTop = true
                    }
                }
            )
        }

        // Optional legacy entries
        composable("schedules") { SchedulesScreen(repo = repo, onBack = { navController.popBackStack() }) }
        composable("emergency") { EmergencyScreen(repo = repo, onBack = { navController.popBackStack() }) }
    }
  }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\navigation\BottomNav.kt

package com.example.androidbloodbank.navigation

import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.navigation.NavHostController
import androidx.navigation.compose.currentBackStackEntryAsState
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.outlined.Home
import androidx.compose.material.icons.outlined.Search
import androidx.compose.material.icons.outlined.AddCircle
import androidx.compose.material.icons.outlined.Bloodtype
import androidx.compose.material.icons.outlined.Person

data class BottomItem(val route: String, val label: String, val icon: ImageVector)

@Composable
fun BottomNavBar(navController: NavHostController) {
    val items = listOf(
        BottomItem(Route.Home.path,         "Home",    Icons.Outlined.Home),
        BottomItem(Route.FindDonors.path,   "Donors",  Icons.Outlined.Search),
        BottomItem(Route.RequestBlood.path, "Request", Icons.Outlined.AddCircle),
        BottomItem(Route.BloodBank.path,    "Bank",    Icons.Outlined.Bloodtype),
        BottomItem(Route.Profile.path,      "Profile", Icons.Outlined.Person),
    )

    val backStack by navController.currentBackStackEntryAsState()
    val currentRoute = backStack?.destination?.route

    NavigationBar(
        containerColor = MaterialTheme.colorScheme.surface,
        contentColor   = MaterialTheme.colorScheme.onSurfaceVariant
    ) {
        items.forEach { item ->
            val selected = currentRoute?.startsWith(item.route) == true
            NavigationBarItem(
                selected = selected,
                onClick = {
                    if (!selected) {
                        navController.navigate(item.route) {
                            popUpTo(Route.Home.path) { saveState = true }
                            launchSingleTop = true
                            restoreState = true
                        }
                    }
                },
                icon = { Icon(item.icon, contentDescription = item.label) },
                label = { Text(item.label) },
                colors = NavigationBarItemDefaults.colors(
                    selectedIconColor   = MaterialTheme.colorScheme.primary,
                    selectedTextColor   = MaterialTheme.colorScheme.primary,
                    indicatorColor      = MaterialTheme.colorScheme.surfaceVariant,
                    unselectedIconColor = MaterialTheme.colorScheme.onSurfaceVariant,
```

```
                unselectedTextColor = MaterialTheme.colorScheme.onSurfaceVariant
            )
        )
    }
  }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\navigation\Routes.kt

package com.example.androidbloodbank.navigation

sealed class Route(val path: String) {
    // Entry
    data object Splash : Route("splash")
    data object Gate   : Route("gate")

    // Auth
    data object SignIn : Route("signin")
    data object SignUp : Route("signup")

    // Main
    data object Home   : Route("home")

    // Donate flow
    data object Donate       : Route("donate")
    data object ViewRequests : Route("view_requests")
    data object PostRequest  : Route("post_request")

    // Find donors flow
    data object FindDonors       : Route("find_donors")
    data object SelectBloodGroup : Route("select_bg")
    data object DonorProfile : Route("donor_profile/{donorId}") {
        private const val KEY = "donorId"
        fun create(donorId: String) = "donor_profile/$donorId"
        const val ArgKey = KEY
    }

    // Blood bank flow
    data object BloodBank       : Route("blood_bank")
    data object NearbyBloodBank : Route("nearby_blood_bank")
    data object AvailableBlood  : Route("available_blood")

    // Request blood flow
    data object RequestBlood : Route("request_blood")
    data object TrackRequest : Route("track_request")

    // Profile & info
    data object Profile   : Route("profile")
    data object BloodInfo : Route("blood_info")

    // NEW: Emergency SOS (offline donors)
    data object EmergencySOS : Route("emergency_sos")
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\AccountGateScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp

@Composable
fun AccountGateScreen(
    onLogin: () -> Unit,
    onSignUp: () -> Unit,
    onEmergency: () -> Unit
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(24.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            "Android Blood Bank",
            style = MaterialTheme.typography.headlineMedium,
            fontWeight = FontWeight.Bold
        )
        Spacer(Modifier.height(8.dp))
        Text(
            "Find donors, request blood — or use SOS offline list.",
            style = MaterialTheme.typography.bodyMedium,
            color = MaterialTheme.colorScheme.onSurfaceVariant
        )

        Spacer(Modifier.height(32.dp))

        Button(
            onClick = onLogin,
            modifier = Modifier.fillMaxWidth().height(52.dp)
        ) { Text("Login") }

        Spacer(Modifier.height(12.dp))

        FilledTonalButton(
            onClick = onSignUp,
            modifier = Modifier.fillMaxWidth().height(52.dp)
        ) { Text("Sign up") }

        Spacer(Modifier.height(16.dp))

        OutlinedButton(
            onClick = onEmergency,
```

```
            modifier = Modifier.fillMaxWidth().height(48.dp)
        ) { Text("Emergency SOS (offline donors)") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\AvailableBloodScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@Composable
fun AvailableBloodScreen(onBack: () -> Unit) {
    Column(Modifier.fillMaxSize().padding(16.dp), verticalArrangement = Arrangement.spacedBy(8.dp)) {
        Text("Available Blood (stub)", style = MaterialTheme.typography.headlineSmall)
        // TODO: inventory UI
        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\BloodBankScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@Composable
fun BloodBankScreen(onNearby: () -> Unit, onAvailable: () -> Unit, onBack: () -> Unit) {
    Column(Modifier.fillMaxSize().padding(16.dp), verticalArrangement = Arrangement.spacedBy(12.dp)) {
        Text("Blood bank", style = MaterialTheme.typography.headlineSmall)
        Button(onClick = onNearby, modifier = Modifier.fillMaxWidth()) { Text("Nearby Blood bank") }
        Button(onClick = onAvailable, modifier = Modifier.fillMaxWidth()) { Text("See available blood") }
        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\BloodInfoScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@Composable
fun BloodInfoScreen(onBack: () -> Unit) {
    Column(Modifier.fillMaxSize().padding(16.dp)) {
        Text("Blood Info (education)", style = MaterialTheme.typography.headlineSmall)
        // TODO: educational content
        Spacer(Modifier.height(8.dp))
        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\DonateScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@Composable
fun DonateScreen(
    onViewRequests: () -> Unit,
    onPostRequest: () -> Unit,
    onBack: () -> Unit
) {
    Column(
        Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(12.dp)
    ) {
        Text("Donate blood", style = MaterialTheme.typography.headlineSmall)
        Button(onClick = onViewRequests, modifier = Modifier.fillMaxWidth()) { Text("View Requests") }
        Button(onClick = onPostRequest, modifier = Modifier.fillMaxWidth()) { Text("Post Request") }
        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\DonorProfileScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo

@Composable
fun DonorProfileScreen(donorId: String, repo: LocalRepo, onBack: () -> Unit) {
    val donor = remember { repo.loadDonors().find { it.id == donorId } }

    Column(Modifier.fillMaxSize().padding(16.dp), verticalArrangement = Arrangement.spacedBy(8.dp)) {
        Text(donor?.name ?: "Unknown donor", style = MaterialTheme.typography.headlineSmall)
        Text("Blood: ${donor?.bloodGroup?.label ?: "—"}")
        Text("Verified: ${if (donor?.verified == true) "Yes" else "No"}")
        donor?.phone?.let { Text("Phone: $it") }
        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\EmergencySosScreen.kt

package com.example.androidbloodbank.ui.flow

import android.content.Intent
import android.net.Uri
import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.outlined.ArrowBack
import androidx.compose.material.icons.outlined.Call
import androidx.compose.material.icons.outlined.Place
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo
import com.example.androidbloodbank.data.model.Donor

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun EmergencySosScreen(
    repo: LocalRepo,
    onBack: () -> Unit
) {
    val context = LocalContext.current
    val donors = remember { repo.loadDonors() } // predefined offline list from LocalRepo

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Emergency SOS (offline)") },
                navigationIcon = {
                    IconButton(onClick = onBack) {
                        Icon(Icons.Outlined.ArrowBack, contentDescription = "Back")
                    }
                }
            )
        }
    ) { padding ->
        if (donors.isEmpty()) {
            Box(
                modifier = Modifier
                    .padding(padding)
                    .fillMaxSize(),
                contentAlignment = Alignment.Center
            ) {
```

```
            Text("No offline donors available.", color = MaterialTheme.colorScheme.onSurfaceVariant)
        }
    } else {
        LazyColumn(
            modifier = Modifier
                .padding(padding)
                .fillMaxSize()
                .padding(16.dp),
            verticalArrangement = Arrangement.spacedBy(12.dp)
        ) {
            items(donors, key = { it.id }) { d ->
                EmergencyDonorRow(
                    donor = d,
                    onCall = {
                        d.phone?.let { ph ->
                            val i = Intent(Intent.ACTION_DIAL, Uri.parse("tel:$ph"))
                            context.startActivity(i)
                        }
                    }
                )
            }
        }
    }
}

@Composable
private fun EmergencyDonorRow(
    donor: Donor,
    onCall: () -> Unit
) {
    val border = MaterialTheme.colorScheme.outlineVariant
    val placeholder = MaterialTheme.colorScheme.surfaceVariant

    Surface(
        shape = MaterialTheme.shapes.large,
        tonalElevation = 0.dp,
        color = MaterialTheme.colorScheme.surface,
        border = BorderStroke(1.dp, border)
    ) {
        Column(Modifier.fillMaxWidth().padding(12.dp)) {
            Row(verticalAlignment = Alignment.CenterVertically) {
                // Placeholder avatar
                Box(
                    modifier = Modifier
                        .size(48.dp)
                        .clip(MaterialTheme.shapes.medium)
                        .background(placeholder)
                )
                Spacer(Modifier.width(12.dp))

                Column(Modifier.weight(1f)) {
                    Text(donor.name, style = MaterialTheme.typography.titleMedium, fontWeight = FontWeight.SemiBold)
                    Row(verticalAlignment = Alignment.CenterVertically) {
                        Icon(Icons.Outlined.Place, contentDescription = null, tint = MaterialTheme.colorScheme.onSurfaceVariant)
```

```
                Spacer(Modifier.width(4.dp))
                val loc = listOfNotNull(donor.area, donor.city).joinToString(", ").ifBlank { "—" }
                Text(loc, style = MaterialTheme.typography.bodySmall, color = MaterialTheme.colorScheme.onSurfaceVariant)
            }
        }

        Column(horizontalAlignment = Alignment.End) {
            Text(donor.bloodGroup.label, style = MaterialTheme.typography.titleLarge)
            IconButton(onClick = onCall, enabled = donor.phone != null) {
                Icon(Icons.Outlined.Call, contentDescription = "Call")
            }
        }
    }

    if (!donor.hospital.isNullOrBlank()) {
        Spacer(Modifier.height(8.dp))
                        Text("Pref. hospital: ${donor.hospital}", style = MaterialTheme.typography.bodySmall, color =
MaterialTheme.colorScheme.onSurfaceVariant)
    }
  }
 }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\FindDonorsScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.outlined.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo
import com.example.androidbloodbank.data.model.BloodGroup
import com.example.androidbloodbank.data.model.Donor
import com.example.androidbloodbank.ui.components.BloodGroupChips

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun FindDonorsScreen(
    repo: LocalRepo,
    onSelectBG: () -> Unit,   // kept for compatibility; you can remove if unused
    onOpenDonor: (String) -> Unit,
    onBack: () -> Unit
) {
    var selected by remember { mutableStateOf<BloodGroup?>(null) }
    val all = remember { repo.loadDonors() }
    val donors = remember(selected, all) {
        if (selected == null) all else all.filter { it.bloodGroup == selected }
    }

    Scaffold(
        topBar = {
            TopAppBar( // <- use TopAppBar for wider compatibility
                title = { Text("Find a donor") },
                navigationIcon = {
                    IconButton(onClick = onBack) {
                        Icon(Icons.Outlined.ArrowBack, contentDescription = "Back")
                    }
                }
            )
        }
    ) { padding ->
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(padding)
                .padding(16.dp),
```

```
            verticalArrangement = Arrangement.spacedBy(16.dp)
        ) {
            Text(
                "Select your blood group",
                style = MaterialTheme.typography.labelLarge,
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )

            BloodGroupChips(
                selected = selected,
                onSelected = { selected = it }
            )

            Text(
                "Results",
                style = MaterialTheme.typography.titleMedium,
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )

            LazyColumn(
                modifier = Modifier.fillMaxSize(),
                verticalArrangement = Arrangement.spacedBy(12.dp)
            ) {
                items(donors, key = { it.id }) { donor ->
                    DonorListItem(
                        donor = donor,
                        onDetails = { onOpenDonor(donor.id) },
                        onDonate = { onOpenDonor(donor.id) }, // hook to your donate flow if different
                        onCall = { /* TODO open dialer with donor.phone */ }
                    )
                }
            }
        }
    }
}

@Composable
private fun DonorListItem(
    donor: Donor,
    onDetails: () -> Unit,
    onDonate: () -> Unit,
    onCall: () -> Unit
) {
    val borderColor = MaterialTheme.colorScheme.outlineVariant
    val container = MaterialTheme.colorScheme.surface
    val placeholder = MaterialTheme.colorScheme.surfaceVariant

    // Helpers
    // Helpers
    fun daysSince(millis: Long?): Long? =
        millis?.let { (System.currentTimeMillis() - it) / (1000L * 60 * 60 * 24) }

    val days = daysSince(donor.lastDonationMillis)

// Eligible if 90+ days or unknown
```

```
    val eligible = days?.let { it >= 90L } ?: true

// Human text for last donation
    val lastDonationText = days?.let { d ->
        when {
            d < 1L    -> "Today"
            d < 30L   -> "${d}d ago"
            d < 365L  -> "${d / 30L} mo ago"
            else      -> "${d / 365L} yr ago"
        }
    } ?: "No record"

    val locationText = listOfNotNull(donor.area, donor.city).joinToString(separator = ", ").ifBlank { "Location N/A" }

    Surface(
        shape = MaterialTheme.shapes.large,
        tonalElevation = 0.dp,
        color = container,
        border = BorderStroke(1.dp, borderColor)
    ) {
        Column(modifier = Modifier.fillMaxWidth().padding(12.dp)) {
            Row(verticalAlignment = Alignment.CenterVertically) {
                // Avatar placeholder
                Box(
                    modifier = Modifier
                        .size(48.dp)
                        .clip(MaterialTheme.shapes.medium)
                        .background(placeholder)
                )

                Spacer(Modifier.width(12.dp))

                Column(modifier = Modifier.weight(1f)) {
                    Row(verticalAlignment = Alignment.CenterVertically) {
                        Text(
                            donor.name,
                            style = MaterialTheme.typography.titleMedium,
                            fontWeight = FontWeight.SemiBold
                        )
                        Spacer(Modifier.width(8.dp))
                        if (donor.verified) {
                            AssistChip(
                                onClick = {},
                                label = { Text("Verified") },
                                leadingIcon = { Icon(Icons.Outlined.Verified, null) }
                            )
                        }
                        if (eligible) {
                            Spacer(Modifier.width(6.dp))
                            AssistChip(onClick = {}, label = { Text("Eligible") })
                        }
                    }

                    // Location + age
                    Row(verticalAlignment = Alignment.CenterVertically) {
```

```
            Icon(
                Icons.Outlined.Place,
                contentDescription = null,
                tint = MaterialTheme.colorScheme.onSurfaceVariant
            )
            Spacer(Modifier.width(4.dp))
            val ageText = donor.age?.let { " • Age $it" } ?: ""
            Text(
                "$locationText$ageText",
                color = MaterialTheme.colorScheme.onSurfaceVariant,
                style = MaterialTheme.typography.bodySmall
            )
        }
    }

    // Right column: big blood group + call
    Column(horizontalAlignment = Alignment.End) {
        Text(
            donor.bloodGroup.label,
            style = MaterialTheme.typography.titleLarge,
            color = MaterialTheme.colorScheme.onSurface
        )
        IconButton(onClick = onCall) {
            Icon(Icons.Outlined.Call, contentDescription = "Call")
        }
    }
}

Spacer(Modifier.height(8.dp))

// Hospital row
Row(verticalAlignment = Alignment.CenterVertically) {
    Icon(
        Icons.Outlined.LocalHospital,
        contentDescription = null,
        tint = MaterialTheme.colorScheme.onSurfaceVariant
    )
    Spacer(Modifier.width(6.dp))
    Text(
        donor.hospital ?: "Preferred hospital not set",
        style = MaterialTheme.typography.bodySmall,
        color = MaterialTheme.colorScheme.onSurfaceVariant
    )
}

// Last donation row
Row(verticalAlignment = Alignment.CenterVertically) {
    Icon(
        Icons.Outlined.Schedule,
        contentDescription = null,
        tint = MaterialTheme.colorScheme.onSurfaceVariant
    )
    Spacer(Modifier.width(6.dp))
    Text(
        "Last donation: $lastDonationText",
```

```
                style = MaterialTheme.typography.bodySmall,
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )
        }

        Spacer(Modifier.height(10.dp))

        Row(horizontalArrangement = Arrangement.spacedBy(12.dp), modifier = Modifier.fillMaxWidth()) {
            OutlinedButton(onClick = onDetails, modifier = Modifier.weight(1f)) {
                Text("View Details")
            }
            Button(onClick = onDonate, modifier = Modifier.weight(1f)) {
                Text("Donate")
            }
        }
    }
  }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\NearbyBloodBankScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@Composable
fun NearbyBloodBankScreen(onBack: () -> Unit) {
    Column(Modifier.fillMaxSize().padding(16.dp), verticalArrangement = Arrangement.spacedBy(8.dp)) {
        Text("Nearby Blood Banks (stub)", style = MaterialTheme.typography.headlineSmall)
        // TODO: show list/map of nearby banks
        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\PostRequestScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo
import com.example.androidbloodbank.data.model.BloodRequest
import com.example.androidbloodbank.data.model.RequestStatus

@Composable
fun PostRequestScreen(
    repo: LocalRepo,
    onPosted: () -> Unit,
    onBack: () -> Unit
) {
    var name by remember { mutableStateOf("") }
    var hospital by remember { mutableStateOf("") }
    var location by remember { mutableStateOf("") }
    var contact by remember { mutableStateOf("") }
    var bg by remember { mutableStateOf("O+") }

    Column(
        Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        Text("Post Request", style = MaterialTheme.typography.headlineSmall)

        OutlinedTextField(name, { name = it }, label = { Text("Name") }, modifier = Modifier.fillMaxWidth())
        OutlinedTextField(hospital, { hospital = it }, label = { Text("Hospital") }, modifier = Modifier.fillMaxWidth())
        OutlinedTextField(location, { location = it }, label = { Text("Location") }, modifier = Modifier.fillMaxWidth())
        OutlinedTextField(contact, { contact = it }, label = { Text("Contact") }, modifier = Modifier.fillMaxWidth())
        OutlinedTextField(bg, { bg = it }, label = { Text("Blood Group") }, modifier = Modifier.fillMaxWidth())

        Button(
            onClick = {
                val list = repo.loadRequests().apply {
                    add(
                        com.example.androidbloodbank.data.model.BloodRequest(
                            requesterName = name,
                            hospital = hospital,
                            location = location,
                            contactNumber = contact,
                            bloodGroup = bg,
                            status = RequestStatus.PENDING   // <-- NEW
                        )
                    )
                }
                repo.saveRequests(list)
                onPosted()

            },
```

```
            modifier = Modifier.fillMaxWidth()
        ) { Text("Submit") }

        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\SelectBloodGroupScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.model.BloodGroup
import com.example.androidbloodbank.ui.components.BloodGroupSelector

@Composable
fun SelectBloodGroupScreen(
    onDone: () -> Unit,
    onBack: () -> Unit
) {
    var selected by remember { mutableStateOf(BloodGroup.O_POS) }

    Column(
        modifier = Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(12.dp)
    ) {
        Text("Select Blood Group", style = MaterialTheme.typography.headlineSmall)

        BloodGroupSelector(
            value = selected,
            onChange = { selected = it },
            modifier = Modifier.fillMaxWidth()
        )

        Spacer(Modifier.height(8.dp))

        Button(
            onClick = { onDone() }, // (Optional) pass 'selected' back via a lambda if needed
            modifier = Modifier.fillMaxWidth()
        ) { Text("Done") }

        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\SplashScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@Composable
fun SplashScreen(onDone: () -> Unit) {
    LaunchedEffect(Unit) { onDone() } // simple handoff; add delay/branding if you like
    Box(Modifier.fillMaxSize().padding(16.dp)) { Text("Splash...") }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\TrackRequestScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo
import com.example.androidbloodbank.data.model.RequestStatus


@Composable
fun TrackRequestScreen(repo: LocalRepo, onBack: () -> Unit) {
    val requests = remember { repo.loadRequests() } // see LocalRepo additions below
    Column(Modifier.fillMaxSize().padding(16.dp), verticalArrangement = Arrangement.spacedBy(8.dp)) {
        Text("Track Request", style = MaterialTheme.typography.headlineSmall)
        requests.forEach { r ->
            Text("• ${r.requesterName} – ${r.bloodGroup}  [${r.status}]")
        }

        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\flow\ViewRequestsScreen.kt

package com.example.androidbloodbank.ui.flow

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo

@Composable
fun ViewRequestsScreen(repo: LocalRepo, onBack: () -> Unit) {
    val requests = remember { repo.loadRequests() } // see LocalRepo additions below
    Column(
        Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        Text("Requests (${requests.size})", style = MaterialTheme.typography.headlineSmall)
        if (requests.isEmpty()) Text("No requests yet.")
        requests.forEach { r ->
            Text("• ${r.requesterName} – ${r.bloodGroup} @ ${r.hospital}")
        }
        Spacer(Modifier.height(8.dp))
        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\ChatScreen.kt

package com.example.androidbloodbank.ui.screens

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@Composable
fun ChatScreen(onBack: () -> Unit) {
    Column(modifier = Modifier.fillMaxSize().padding(16.dp)) {
        Text("Chat (placeholder)", style = MaterialTheme.typography.headlineSmall)
        Spacer(Modifier.height(8.dp))
        Text("If you integrated a chatbot, messages would appear here.")
        Spacer(Modifier.height(16.dp))
        Button(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\Color.kt

package com.example.androidbloodbank.ui.theme

import androidx.compose.ui.graphics.Color

// ---------- LIGHT (unchanged vibe, soft & airy) ----------
val PrimaryLight             = Color(0xFFE56B6F) // soft rose
val OnPrimaryLight           = Color(0xFFFFFFFF)
val PrimaryContainerLight    = Color(0xFFFFD9DC)
val OnPrimaryContainerLight  = Color(0xFF3A0D12)

val SecondaryLight           = Color(0xFFF3A0A7) // blush
val OnSecondaryLight         = Color(0xFF3F2326)
val SecondaryContainerLight  = Color(0xFFFFE3E6)
val OnSecondaryContainerLight = Color(0xFF35161A)

val TertiaryLight            = Color(0xFF7AB6B1) // soft mint (eye relief)
val OnTertiaryLight          = Color(0xFF0E2F2C)
val TertiaryContainerLight   = Color(0xFFD7F2EF)
val OnTertiaryContainerLight  = Color(0xFF0B2322)

val BackgroundLight          = Color(0xFFFFFBFB)
val SurfaceLight             = Color(0xFFFFFFFF)
val SurfaceVariantLight      = Color(0xFFF2E7E9)
val OnSurfaceLight           = Color(0xFF1E1E1F)
val OnSurfaceVariantLight    = Color(0xFF514A4C)
val OutlineLight             = Color(0xFF7A7275)
val OutlineVariantLight      = Color(0xFFE5DADC)

// ---------- DARK (NEW: cozy, muted, no pure black) ----------
val PrimaryDark              = Color(0xFFFF8C95) // gentle rose accent
val OnPrimaryDark            = Color(0xFF2C0A0E)
val PrimaryContainerDark     = Color(0xFF3E1419)
val OnPrimaryContainerDark   = Color(0xFFFFEDEE)

val SecondaryDark            = Color(0xFFF2B8BD)
val OnSecondaryDark          = Color(0xFF2D1316)
val SecondaryContainerDark   = Color(0xFF3D2024)
val OnSecondaryContainerDark  = Color(0xFFFFECEE)

val TertiaryDark             = Color(0xFFAEDDD8)
val OnTertiaryDark           = Color(0xFF062927)
val TertiaryContainerDark    = Color(0xFF123B38)
val OnTertiaryContainerDark  = Color(0xFFD9F4F1)

val BackgroundDark           = Color(0xFF0F1113) // deep blue-grey, not black
val SurfaceDark              = Color(0xFF131517)
val SurfaceVariantDark       = Color(0xFF22282B)
val OnSurfaceDark            = Color(0xFFE3E6E8)
val OnSurfaceVariantDark     = Color(0xFFB2B8BC)
val OutlineDark              = Color(0xFF394145)
val OutlineVariantDark       = Color(0xFF30373B)
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\DashboardScreen.kt

package com.example.androidbloodbank.ui.screens

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.material3.CardDefaults
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo



@Composable
fun DashboardScreen(
    repo: LocalRepo,
    onRequest: () -> Unit,
    onSchedules: () -> Unit,
    onEmergency: () -> Unit,
    onProfile: () -> Unit
) {
    Column(modifier = Modifier.fillMaxSize().padding(16.dp), verticalArrangement = Arrangement.spacedBy(12.dp)) {
        Text("Dashboard", style = MaterialTheme.typography.headlineMedium)
        Spacer(Modifier.height(8.dp))

        Row(modifier = Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.spacedBy(12.dp)) {
            ActionTile(icon = Icons.Default.Search, label = "Find Donor", onClick = onRequest)
            ActionTile(icon = Icons.Default.Person, label = "My Profile", onClick = onProfile)
        }

        Row(modifier = Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.spacedBy(12.dp)) {
            ActionTile(icon = Icons.Default.Schedule, label = "Schedules", onClick = onSchedules)
            ActionTile(icon = Icons.Default.Warning, label = "Emergency", onClick = onEmergency, accent = true)
        }

        Spacer(Modifier.height(12.dp))
        Text("Nearby Donors", style = MaterialTheme.typography.titleMedium)
        Spacer(Modifier.height(8.dp))

        // Keep your existing donor list UI here (cards) — it will scroll below
        // For brevity show a placeholder:
        Card(modifier = Modifier.fillMaxWidth(), elevation = CardDefaults.cardElevation(4.dp)) {
            Column(modifier = Modifier.padding(12.dp)) {
                Text("Rahim Uddin — O- — Verified")
                Text("Ayesha Khan — A+ — Verified")
            }
        }
    }
}
```

```
@Composable
private fun ActionTile(icon: androidx.compose.ui.graphics.vector.ImageVector, label: String, onClick: () -> Unit, accent: Boolean = false) {
    val bg = if (accent) MaterialTheme.colorScheme.error else MaterialTheme.colorScheme.surfaceVariant
    val tint = if (accent) MaterialTheme.colorScheme.onError else MaterialTheme.colorScheme.onSurfaceVariant

    Row(
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.spacedBy(12.dp)
    ) {
        ElevatedCard(
            modifier = Modifier
                .weight(1f)  //  weight applied here in Row scope
                .height(120.dp)
                .clickable { onClick() },
            elevation = CardDefaults.elevatedCardElevation(6.dp)
        ) {
            Column(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(12.dp),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            ) {
                Icon(icon, contentDescription = label, tint = tint, modifier = Modifier.size(36.dp))
                Spacer(Modifier.height(8.dp))
                Text(label)
            }
        }
    }


}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\EmergencyScreen.kt

package com.example.androidbloodbank.ui

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.androidbloodbank.data.LocalRepo
import com.example.androidbloodbank.data.model.Donor

@Composable
fun EmergencyScreen(repo: LocalRepo, onBack: () -> Unit) {
    val donors = remember { repo.loadDonors() }
    val priority = donors.sortedWith(compareByDescending<Donor> { it.verified }.thenBy { it.bloodGroup.label })

    Column(modifier = Modifier.fillMaxSize().padding(16.dp), horizontalAlignment = Alignment.CenterHorizontally) {
        Box(modifier = Modifier.fillMaxWidth().height(120.dp).background(Color(0xFFB00020)), contentAlignment = Alignment.Center)
{
            Text("EMERGENCY MODE", color = Color.White, fontSize = 22.sp, fontWeight = FontWeight.ExtraBold)
        }
        Spacer(Modifier.height(12.dp))
        Text("We will show nearby verified donors and priority rare blood types (simulated).")
        Spacer(Modifier.height(12.dp))
        Text("Priority list:", fontWeight = FontWeight.SemiBold)
        Spacer(Modifier.height(8.dp))
        Column(modifier = Modifier.fillMaxWidth().verticalScroll(rememberScrollState())) {
            priority.forEach { d ->
                Card(modifier = Modifier.fillMaxWidth().padding(6.dp)) {
                    Row(modifier = Modifier.padding(12.dp), verticalAlignment = Alignment.CenterVertically) {
                        Column(modifier = Modifier.weight(1f)) {
                            Text(d.name, fontWeight = FontWeight.SemiBold)
                            Text("Blood: ${d.bloodGroup.label}")
                        }
                        if (d.verified) Text("Verified", color = Color(0xFF2E7D32))
                        Spacer(Modifier.width(8.dp))
                        Button(onClick = { /* production: call/message donor */ }) { Text("Call (sim)") }
                    }
                }
            }
        }
        Spacer(Modifier.height(12.dp))
        TextButton(onClick = onBack) { Text("Exit Emergency") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\HomeScreen.kt

package com.example.androidbloodbank.ui.screens

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

@Composable
fun HomeScreen(
    onDonate: () -> Unit,
    onFindDonors: () -> Unit,
    onBloodBank: () -> Unit,
    onRequestBlood: () -> Unit,
    onProfile: () -> Unit
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        // Hero banner (no cards)
        Box(
            modifier = Modifier
                .fillMaxWidth()
                .height(160.dp)
                .background(
                    brush = Brush.verticalGradient(
                        colors = listOf(
                            MaterialTheme.colorScheme.primary,
                            MaterialTheme.colorScheme.primary.copy(alpha = 0.75f)
                        )
                    ),
                    shape = MaterialTheme.shapes.extraLarge
                )
                .padding(20.dp),
            contentAlignment = Alignment.BottomStart
        ) {
            Column {
                Text("Android Blood Bank", color = Color.White, fontSize = 22.sp, fontWeight = FontWeight.Bold)
                Text("Find donors fast. Request in seconds.", color = Color.White.copy(alpha = 0.9f))
            }
        }

        Spacer(Modifier.height(4.dp))
```

```
        // Minimal, button-first layout (no cards)
        Button(onClick = onRequestBlood, modifier = Modifier.fillMaxWidth().height(52.dp)) {
            Text("Request Blood", fontSize = 16.sp, fontWeight = FontWeight.SemiBold)
        }
        FilledTonalButton(onClick = onFindDonors, modifier = Modifier.fillMaxWidth().height(52.dp)) {
            Text("Find Donors")
        }

        Row(horizontalArrangement = Arrangement.spacedBy(12.dp), modifier = Modifier.fillMaxWidth()) {
            OutlinedButton(onClick = onDonate, modifier = Modifier.weight(1f).height(48.dp)) { Text("Donate") }
            OutlinedButton(onClick = onBloodBank, modifier = Modifier.weight(1f).height(48.dp)) { Text("Blood Bank") }
        }

        OutlinedButton(onClick = onProfile, modifier = Modifier.fillMaxWidth().height(48.dp)) {
            Text("Profile")
        }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\LoginScreen.kt

package com.example.androidbloodbank.ui

import android.util.Patterns
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.outlined.ArrowBack
import androidx.compose.material.icons.outlined.Visibility
import androidx.compose.material.icons.outlined.VisibilityOff
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.input.VisualTransformation
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo
import com.google.firebase.auth.FirebaseAuth
import com.google.gson.Gson
import kotlinx.coroutines.launch
import kotlinx.coroutines.tasks.await

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun LoginScreen(
    repo: LocalRepo,
    onBack: () -> Unit,
    onLoginSuccess: () -> Unit
) {
    val auth = remember { FirebaseAuth.getInstance() }
    val snackbarHostState = remember { SnackbarHostState() }
    val scope = rememberCoroutineScope()

    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var showPassword by remember { mutableStateOf(false) }
    var loading by remember { mutableStateOf(false) }

    fun isEmail(s: String) = Patterns.EMAIL_ADDRESS.matcher(s.trim()).matches()
    fun canSubmit() = isEmail(email) && password.isNotBlank()

    suspend fun tryLogin() {
        loading = true
        try {
            auth.signInWithEmailAndPassword(email.trim(), password).await()

            // OPTIONAL: keep a tiny local session so Splash/Gate guards work with or without Firebase
            val sessionJson = Gson().toJson(mapOf("email" to email.trim(), "uid" to (auth.currentUser?.uid ?: "")))
            repo.saveCurrentUserJson(sessionJson)

            onLoginSuccess()
        } catch (e: Exception) {
```

```
            snackbarHostState.showSnackbar(e.localizedMessage ?: "Invalid email or password.")
        } finally {
            loading = false
        }
    }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Login") },
                navigationIcon = {
                    IconButton(onClick = onBack) {
                        Icon(Icons.Outlined.ArrowBack, contentDescription = "Back")
                    }
                }
            )
        },
        snackbarHost = { SnackbarHost(hostState = snackbarHostState) }
    ) { padding ->
        Column(
            modifier = Modifier
                .padding(padding)
                .padding(16.dp)
                .fillMaxSize(),
            verticalArrangement = Arrangement.spacedBy(16.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            OutlinedTextField(
                value = email,
                onValueChange = { email = it },
                label = { Text("Email") },
                singleLine = true,
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email),
                modifier = Modifier.fillMaxWidth()
            )

            OutlinedTextField(
                value = password,
                onValueChange = { password = it },
                label = { Text("Password") },
                singleLine = true,
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
                visualTransformation = if (showPassword) VisualTransformation.None else PasswordVisualTransformation(),
                trailingIcon = {
                    val icon = if (showPassword) Icons.Outlined.VisibilityOff else Icons.Outlined.Visibility
                    IconButton(onClick = { showPassword = !showPassword }) {
                        Icon(icon, contentDescription = if (showPassword) "Hide password" else "Show password")
                    }
                },
                modifier = Modifier.fillMaxWidth()
            )

            Spacer(Modifier.height(8.dp))

            Button(
```

```
            onClick = { scope.launch { tryLogin() } },  // <-- run suspend function in coroutine
            enabled = canSubmit() && !loading,
            modifier = Modifier.fillMaxWidth().height(52.dp)
        ) {
            if (loading) {
                CircularProgressIndicator(strokeWidth = 2.dp, modifier = Modifier.size(18.dp))
                Spacer(Modifier.width(12.dp))
                Text("Signing in...")
            } else {
                Text("Login")
            }
        }
    }
  }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\ProfileScreen.kt

package com.example.androidbloodbank.ui.screens

import android.content.Intent
import android.os.Build
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.outlined.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.ui.unit.dp
import coil.compose.AsyncImage
import com.example.androidbloodbank.data.LocalRepo
import com.example.androidbloodbank.data.model.BloodGroup
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.util.*

// ---------- small utils ----------
private fun formatDate(millis: Long?): String {
    if (millis == null || millis == 0L) return "Not set"
    return try {
        if (Build.VERSION.SDK_INT >= 26) {
            val fmt = java.time.format.DateTimeFormatter.ofPattern("dd-MM-yyyy")
            val dt = java.time.Instant.ofEpochMilli(millis).atZone(java.time.ZoneId.systemDefault()).toLocalDate()
            fmt.format(dt)
        } else {
            SimpleDateFormat("dd-MM-yyyy", Locale.getDefault()).format(Date(millis))
        }
    } catch (_: Exception) { "Not set" }
}

private val genderOptions = listOf("Male", "Female", "Other", "Prefer not to say")
private val bloodLabels = listOf("A+","A-","B+","B-","AB+","AB-","O+","O-")
private fun labelToEnum(label: String) = when (label) {
    "A+" -> BloodGroup.A_POS; "A-" -> BloodGroup.A_NEG
    "B+" -> BloodGroup.B_POS; "B-" -> BloodGroup.B_NEG
```

```
    "AB+" -> BloodGroup.AB_POS; "AB-" -> BloodGroup.AB_NEG
    "O+" -> BloodGroup.O_POS; else -> BloodGroup.O_NEG
}

// ---------- Screen ----------
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ProfileScreen(
    repo: LocalRepo,
    onBack: () -> Unit,
    onLoggedOut: () -> Unit
) {
    val scope = rememberCoroutineScope()
    val snack = remember { SnackbarHostState() }
    val context = LocalContext.current
    val scroll = rememberScrollState()

    // pull whatever you saved last time; safe parsing with regex
    val current = remember { repo.loadCurrentUserJson() }
    fun pick(field: String, def: String = "") =
        Regex("\"$field\"\\s*:\\s*\"([^\"]*)\"").find(current ?: "")?.groupValues?.getOrNull(1) ?: def
    fun pickLong(field: String) =
        Regex("\"$field\"\\s*:\\s*(\\d+)").find(current ?: "")?.groupValues?.getOrNull(1)?.toLongOrNull()

    var editing by remember { mutableStateOf(false) }

    var name by remember { mutableStateOf(pick("name", "Your name")) }
    var email by remember { mutableStateOf(pick("email")) }
    var phone by remember { mutableStateOf(pick("phone")) }
    var gender by remember { mutableStateOf(pick("gender")) }
    var address by remember { mutableStateOf(pick("address")) }
                        var age by remember { mutableStateOf(Regex("\"age\"\\s*:\\s*(\\d+)").find(current ?:
"")?.groupValues?.getOrNull(1)?.toIntOrNull() ?: 0) }
    var bloodLabel by remember { mutableStateOf(pick("bloodGroup").ifBlank { "O+" }.let { if (it in bloodLabels) it else "O+" }) }
    var lastDonationMillis by remember { mutableStateOf<Long?>(pickLong("lastDonationMillis")) }
    var photoUri by remember { mutableStateOf(pick("photoUri").ifBlank { null }) }

    // date picker
    var showDatePicker by remember { mutableStateOf(false) }
    if (showDatePicker) {
        DatePickerDialog(
            onDismissRequest = { showDatePicker = false },
            confirmButton = {
                TextButton(onClick = { showDatePicker = false }) { Text("Done") }
            }
        ) {
            val state = rememberDatePickerState(initialSelectedDateMillis = lastDonationMillis ?: System.currentTimeMillis())
            DatePicker(state = state, title = { Text("Last donation date") })
            LaunchedEffect(state.selectedDateMillis) { lastDonationMillis = state.selectedDateMillis }
        }
    }

    // image picker: OpenDocument so we can persist access across restarts
    val pickImage = rememberLauncherForActivityResult(
        ActivityResultContracts.OpenDocument()
```

```
) { uri ->
    if (uri != null) {
        // Persist read permission so image still loads after reboot
        try {
            context.contentResolver.takePersistableUriPermission(
                uri,
                Intent.FLAG_GRANT_READ_URI_PERMISSION
            )
        } catch (_: SecurityException) { /* some providers may not support persist; it's ok */ }
        photoUri = uri.toString()
    }
}

Scaffold(
    snackbarHost = { SnackbarHost(snack) },
    topBar = {
        TopAppBar(
            title = { Text(if (editing) "Edit Profile" else "Profile") },
            navigationIcon = { IconButton(onClick = onBack) { Icon(Icons.Outlined.ArrowBack, null) } },
            actions = {
                if (!editing) {
                    TextButton(onClick = { editing = true }) { Text("Edit") }
                } else {
                    TextButton(onClick = {
                        // build and persist a tiny JSON snapshot (add photoUri)
                        val json = buildString {
                            append("{")
                            append("\"name\":\"${name.trim()}\",")
                            append("\"email\":\"${email.trim()}\",")
                            append("\"phone\":\"${phone.trim()}\",")
                            append("\"gender\":\"${gender.trim()}\",")
                            append("\"address\":\"${address.trim()}\",")
                            append("\"age\":${age.coerceAtLeast(0)},")
                            append("\"bloodGroup\":\"$bloodLabel\",")
                            append("\"lastDonationMillis\":${lastDonationMillis ?: 0L},")
                            append("\"photoUri\":\"${photoUri ?: ""}\"")
                            append("}")
                        }
                        repo.saveCurrentUserJson(json)
                        scope.launch { snack.showSnackbar("Profile saved") }
                        editing = false
                    }) { Text("Save", fontWeight = FontWeight.SemiBold) }
                }
            }
        )
    }
) { padding ->
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(padding)
            .verticalScroll(scroll)
            .navigationBarsPadding()
            .imePadding()
    ) {
```

```
        // Header with gradient & avatar
        Box(
            modifier = Modifier
                .fillMaxWidth()
                .height(220.dp)
                .background(
                    Brush.verticalGradient(
                        listOf(Color(0xFFFF6F00), Color(0xFFFF8A80)) // orange -> soft coral
                    ),
                    shape = RoundedCornerShape(bottomStart = 24.dp, bottomEnd = 24.dp)
                )
        ) {
            Column(
                modifier = Modifier.fillMaxSize(),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            ) {
                Box(
                    modifier = Modifier
                        .size(96.dp)
                        .clip(CircleShape)
                        .background(Color(0x1AFFFFFF))
                        .let { base ->
                            if (editing) base.clickable { pickImage.launch(arrayOf("image/*")) } else base
                        },
                    contentAlignment = Alignment.Center
                ) {
                    if (photoUri != null) {
                        AsyncImage(
                            model = photoUri,
                            contentDescription = "Profile photo",
                            modifier = Modifier.matchParentSize().clip(CircleShape)
                        )
                    } else {
                        Icon(Icons.Outlined.Person, contentDescription = null, tint = Color.White, modifier = Modifier.size(44.dp))
                    }
                    if (editing) {
                        Box(
                            modifier = Modifier
                                .align(Alignment.BottomEnd)
                                .offset(x = 2.dp, y = 2.dp)
                                .size(26.dp)
                                .clip(CircleShape)
                                .background(Color.White.copy(alpha = 0.9f)),
                            contentAlignment = Alignment.Center
                        ) {
                            Icon(Icons.Outlined.Edit, contentDescription = null, tint = Color(0xFF6D4C41), modifier = Modifier.size(16.dp))
                        }
                    }
                }
                Spacer(Modifier.height(12.dp))
                        Text(text = name.ifBlank { "Your name" }, color = Color.White, style = MaterialTheme.typography.titleLarge,
fontWeight = FontWeight.Bold)
                                if (email.isNotBlank()) Text(text = email, color = Color.White.copy(alpha = 0.9f), style =
```

```
MaterialTheme.typography.bodyMedium)
            }
        }

        Spacer(Modifier.height(12.dp))
        Text("Account Info", style = MaterialTheme.typography.titleLarge, modifier = Modifier.padding(horizontal = 16.dp))

        Column(
            modifier = Modifier.padding(horizontal = 16.dp, vertical = 8.dp).fillMaxWidth(),
            verticalArrangement = Arrangement.spacedBy(10.dp)
        ) {
            InfoRow(Icons.Outlined.Badge, "Full name") {
                    if (editing) OutlinedTextField(value = name, onValueChange = { name = it }, singleLine = true, modifier =
Modifier.fillMaxWidth())
                else Text(name.ifBlank { "—" })
            }
            InfoRow(Icons.Outlined.Phone, "Mobile") {
                    if (editing) OutlinedTextField(value = phone, onValueChange = { phone = it.filter { c -> c.isDigit() || c == '+'
}.take(16) },
                        singleLine = true, keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Phone), modifier =
Modifier.fillMaxWidth())
                else Text(phone.ifBlank { "—" })
            }
            InfoRow(Icons.Outlined.Email, "Email") {
                if (editing) OutlinedTextField(value = email, onValueChange = { email = it }, singleLine = true,
                    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email), modifier = Modifier.fillMaxWidth())
                else Text(email.ifBlank { "—" })
            }
            InfoRow(Icons.Outlined.Wc, "Gender") {
                 if (editing) ExposedDropdown(current = gender.ifBlank { genderOptions.last() }, options = genderOptions) { gender =
it }
                else Text(gender.ifBlank { "—" })
            }
            InfoRow(Icons.Outlined.Home, "Address") {
                    if (editing) OutlinedTextField(value = address, onValueChange = { address = it }, singleLine = false, minLines = 2,
modifier = Modifier.fillMaxWidth())
                else Text(address.ifBlank { "—" })
            }
            InfoRow(Icons.Outlined.Bloodtype, "Blood group") {
                if (editing) ExposedDropdown(current = bloodLabel, options = bloodLabels) { bloodLabel = it }
                else Text(bloodLabel)
            }
            InfoRow(Icons.Outlined.Cake, "Age") {
                if (editing) OutlinedTextField(
                    value = (age.takeIf { it > 0 }?.toString() ?: ""),
                    onValueChange = { age = it.filter(Char::isDigit).toIntOrNull() ?: 0 },
                        singleLine = true, keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number), modifier =
Modifier.width(120.dp)
                ) else Text(if (age > 0) "$age" else "—")
            }
            InfoRow(Icons.Outlined.Event, "Last donation") {
                if (editing) OutlinedButton(onClick = { showDatePicker = true }) {
                    Icon(Icons.Outlined.DateRange, null); Spacer(Modifier.width(8.dp)); Text(formatDate(lastDonationMillis))
                } else Text(formatDate(lastDonationMillis))
            }
```

```
            Spacer(Modifier.height(8.dp))
            OutlinedButton(
                onClick = {
                    repo.logoutCurrentUser()
                    onLoggedOut()
                },
                modifier = Modifier.fillMaxWidth(),
                colors = ButtonDefaults.outlinedButtonColors(contentColor = MaterialTheme.colorScheme.error)
            ) {
                Icon(Icons.Outlined.Logout, contentDescription = null)
                Spacer(Modifier.width(8.dp))
                Text("Log out")
            }
            Spacer(Modifier.height(24.dp))
        }
    }
  }
}

@Composable
private fun InfoRow(
  icon: androidx.compose.ui.graphics.vector.ImageVector,
  label: String,
  valueContent: @Composable () -> Unit
) {
  Column(Modifier.fillMaxWidth()) {
    Row(verticalAlignment = Alignment.CenterVertically, modifier = Modifier.fillMaxWidth()) {
      Box(
        modifier = Modifier.size(36.dp).clip(CircleShape)
          .background(MaterialTheme.colorScheme.secondaryContainer),
        contentAlignment = Alignment.Center
      ) { Icon(icon, null, tint = MaterialTheme.colorScheme.onSecondaryContainer) }
      Spacer(Modifier.width(12.dp))
      Column(Modifier.weight(1f)) {
        Text(label, style = MaterialTheme.typography.labelLarge, color = MaterialTheme.colorScheme.onSurfaceVariant)
        Spacer(Modifier.height(6.dp))
        valueContent()
      }
    }
    Spacer(Modifier.height(10.dp)); Divider()
  }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun ExposedDropdown(
  current: String,
  options: List<String>,
  onSelect: (String) -> Unit
) {
  var expanded by remember { mutableStateOf(false) }
  ExposedDropdownMenuBox(expanded = expanded, onExpandedChange = { expanded = !expanded }) {
    OutlinedTextField(
      value = current, onValueChange = {}, readOnly = true,
```

```
            trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded) },
            modifier = Modifier.menuAnchor().fillMaxWidth()
        )
        ExposedDropdownMenu(expanded = expanded, onDismissRequest = { expanded = false }) {
            options.forEach { opt ->
                DropdownMenuItem(text = { Text(opt) }, onClick = { onSelect(opt); expanded = false })
            }
        }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\RequestBloodScreen.kt

package com.example.androidbloodbank.ui.screens

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.outlined.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo
import com.example.androidbloodbank.data.model.BloodGroup
import com.example.androidbloodbank.data.model.BloodRequest
import com.google.gson.Gson
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.util.*

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun RequestBloodScreen(
    repo: LocalRepo,
    onBack: () -> Unit
) {
    val snackbar = remember { SnackbarHostState() }
    val scope = rememberCoroutineScope()

    var showingForm by rememberSaveable { mutableStateOf(false) }

    // Live list of requests (sorted newest first)
    var requests by remember {
        mutableStateOf(
            repo.loadRequests().sortedByDescending { readLong(it, "timestamp", "time") ?: 0L }
        )
    }

    // ---------- LIST ----------
    if (!showingForm) {
        Scaffold(
            topBar = {
                TopAppBar(
                    title = { Text("Blood requests") },
                    navigationIcon = { IconButton(onClick = onBack) { Icon(Icons.Outlined.ArrowBack, null) } }
                )
            },
            snackbarHost = { SnackbarHost(snackbar) },
            floatingActionButton = {
                ExtendedFloatingActionButton(
```

```
                onClick = { showingForm = true },
                icon = { Icon(Icons.Outlined.Bloodtype, null) },
                text = { Text("Request Blood") }
            )
        }
    ) { padding ->
        if (requests.isEmpty()) {
            Box(Modifier.fillMaxSize().padding(padding), contentAlignment = Alignment.Center) {
                Text("No requests yet. Tap “Request Blood” to post one.")
            }
        } else {
            LazyColumn(
                modifier = Modifier.fillMaxSize().padding(padding),
                contentPadding = PaddingValues(horizontal = 16.dp, vertical = 16.dp),
                verticalArrangement = Arrangement.spacedBy(12.dp)
            ) {
                itemsIndexed(requests) { index, req ->
                    RequestItem(
                        req = req,
                        key = readString(req, "id") ?: readString(req, "requestId")
                        ?: "${readLong(req, "timestamp", "time") ?: 0L}-$index"
                    )
                }
            }
        }
    }
    return
}

// ---------- FORM (full screen) ----------
RequestBloodFormScreen(
    onBack = { showingForm = false },
    onSubmit = { name, group, hospital, location, contact, needMillis ->
        // 1) Save
        scope.launch {
            val created = buildRequestObject(name, group, hospital, location, contact, needMillis)
            val updated = repo.loadRequests().toMutableList().apply { add(0, created) }
            repo.saveRequests(updated)
            requests = updated.sortedByDescending { readLong(it, "timestamp", "time") ?: 0L }
            snackbar.showSnackbar("Request posted")
        }
        // 2) Close the form immediately (return to list)
        showingForm = false
    }
)
}

/* ------------------ Form with DATE and immediate close ------------------ */

@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun RequestBloodFormScreen(
    onBack: () -> Unit,
    onSubmit: (name: String, group: BloodGroup, hospital: String, location: String, contact: String, needMillis: Long?) -> Unit
) {
```

```
    val snackbar = remember { SnackbarHostState() }
    val scope = rememberCoroutineScope()

    var name by remember { mutableStateOf("") }
    var group by remember { mutableStateOf<BloodGroup?>(null) }
    var hospital by remember { mutableStateOf("") }
    var location by remember { mutableStateOf("") }
    var contact by remember { mutableStateOf("") }

    var needDateMillis by remember { mutableStateOf<Long?>(System.currentTimeMillis()) }
    var showDatePicker by remember { mutableStateOf(false) }

    if (showDatePicker) {
        DatePickerDialog(
            onDismissRequest = { showDatePicker = false },
            confirmButton = { TextButton(onClick = { showDatePicker = false }) { Text("Done") } }
        ) {
            val state = rememberDatePickerState(initialSelectedDateMillis = needDateMillis ?: System.currentTimeMillis())
            DatePicker(state = state, title = { Text("Needed on") })
            LaunchedEffect(state.selectedDateMillis) { needDateMillis = state.selectedDateMillis }
        }
    }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Request Blood") },
                navigationIcon = { IconButton(onClick = onBack) { Icon(Icons.Outlined.Close, null) } }
            )
        },
        snackbarHost = { SnackbarHost(snackbar) }
    ) { padding ->
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(padding)
                .padding(horizontal = 16.dp, vertical = 12.dp),
            verticalArrangement = Arrangement.spacedBy(12.dp)
        ) {
             OutlinedTextField(value = name, onValueChange = { name = it }, label = { Text("Name") }, singleLine = true, modifier =
Modifier.fillMaxWidth())
            BloodGroupDropdown(selected = group, onSelected = { group = it })
                OutlinedTextField(value = hospital, onValueChange = { hospital = it }, label = { Text("Hospital") }, singleLine = true,
modifier = Modifier.fillMaxWidth())
                OutlinedTextField(value = location, onValueChange = { location = it }, label = { Text("Location") }, singleLine = true,
modifier = Modifier.fillMaxWidth())
            OutlinedTextField(
                value = contact, onValueChange = { contact = it },
                label = { Text("Contact") }, singleLine = true,
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Phone),
                modifier = Modifier.fillMaxWidth()
            )
            OutlinedButton(onClick = { showDatePicker = true }, modifier = Modifier.fillMaxWidth().height(52.dp)) {
                Icon(Icons.Outlined.DateRange, null); Spacer(Modifier.width(8.dp)); Text(formatDate(needDateMillis))
            }
```

```
        Spacer(Modifier.height(8.dp))
        Button(
            onClick = {
                if (name.isBlank() || group == null || contact.isBlank()) {
                    scope.launch { snackbar.showSnackbar("Please fill name, blood group and contact") }
                } else {
                    // Call parent submit AND close the form via onBack(), guaranteed return to list
                    onSubmit(name.trim(), group!!, hospital.trim(), location.trim(), contact.trim(), needDateMillis)
                    onBack()
                }
            },
            modifier = Modifier.fillMaxWidth().height(52.dp)
        ) { Text("Submit request") }
    }
  }
}

/* ------------------ List item ------------------ */

@Composable
private fun RequestItem(req: BloodRequest, key: String) {
  ElevatedCard(modifier = Modifier.fillMaxWidth(), colors = CardDefaults.elevatedCardColors()) {
    Column(Modifier.fillMaxWidth().padding(14.dp), verticalArrangement = Arrangement.spacedBy(8.dp)) {
        Row(verticalAlignment = Alignment.CenterVertically) {
            Icon(Icons.Outlined.Person, null); Spacer(Modifier.width(8.dp))
            val who = readString(req, "name", "requesterName") ?: "—"
            Text(who, style = MaterialTheme.typography.titleMedium)
            Spacer(Modifier.weight(1f))
            val bg = readString(req, "bloodGroup", "group") ?: "—"
            AssistChip(onClick = {}, label = { Text(bg) })
        }
                                Row(verticalAlignment = Alignment.CenterVertically) { Icon(Icons.Outlined.LocalHospital, null); Spacer(Modifier.width(8.dp)); Text(readString(req, "hospital") ?: "—") }
         Row(verticalAlignment = Alignment.CenterVertically) { Icon(Icons.Outlined.LocationOn, null); Spacer(Modifier.width(8.dp)); Text(readString(req, "location", "address") ?: "—") }
             Row(verticalAlignment = Alignment.CenterVertically) { Icon(Icons.Outlined.Phone, null); Spacer(Modifier.width(8.dp)); Text(readString(req, "contact", "contactNumber", "phone") ?: "—") }
        val need = readLong(req, "neededDateMillis", "needDate", "dateNeeded", "requiredOn")
        if (need != null && need > 0) {
            Row(verticalAlignment = Alignment.CenterVertically) {
                Icon(Icons.Outlined.DateRange, null); Spacer(Modifier.width(8.dp))
                Text("Needed: ${formatDate(need)}", style = MaterialTheme.typography.bodyMedium)
            }
        }
        val t = readLong(req, "timestamp", "time")
        if (t != null && t > 0) {
            val df = remember { SimpleDateFormat("dd MMM, yyyy h:mma", Locale.getDefault()) }
                                        Text(df.format(Date(t)), style = MaterialTheme.typography.labelSmall, color = MaterialTheme.colorScheme.onSurfaceVariant)
        }
    }
  }
}

/* ------------------ Helpers ------------------ */
```

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun BloodGroupDropdown(
    selected: BloodGroup?,
    onSelected: (BloodGroup) -> Unit
) {
    var expanded by remember { mutableStateOf(false) }
    val all = BloodGroup.values().toList()

    ExposedDropdownMenuBox(expanded = expanded, onExpandedChange = { expanded = !expanded }) {
        OutlinedTextField(
            value = selected?.let { label(it) } ?: "Select blood group",
            onValueChange = {}, readOnly = true,
            trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded) },
            modifier = Modifier.menuAnchor().fillMaxWidth(),
            label = { Text("Blood group") }
        )
        ExposedDropdownMenu(expanded = expanded, onDismissRequest = { expanded = false }) {
            all.forEach { bg ->
                DropdownMenuItem(text = { Text(label(bg)) }, onClick = { onSelected(bg); expanded = false })
            }
        }
    }
}

private fun label(bg: BloodGroup): String = when (bg) {
    BloodGroup.A_POS -> "A+"; BloodGroup.A_NEG -> "A-"
    BloodGroup.B_POS -> "B+"; BloodGroup.B_NEG -> "B-"
    BloodGroup.AB_POS -> "AB+"; BloodGroup.AB_NEG -> "AB-"
    BloodGroup.O_POS -> "O+"; BloodGroup.O_NEG -> "O-"
}

private fun formatDate(millis: Long?): String {
    if (millis == null || millis <= 0) return "Select date"
    val df = SimpleDateFormat("dd MMM, yyyy", Locale.getDefault())
    return df.format(Date(millis))
}

/** Build a request object tolerant to different data-class field names (using Gson). */
private fun buildRequestObject(
    name: String,
    group: BloodGroup,
    hospital: String,
    location: String,
    contact: String,
    needMillis: Long?
): BloodRequest {
    val payload = mapOf(
        "requesterName" to name, "name" to name,
        "bloodGroup" to label(group), "group" to label(group),
        "hospital" to hospital, "location" to location,
        "contactNumber" to contact, "contact" to contact, "phone" to contact,
        "neededDateMillis" to (needMillis ?: 0L), "needDate" to (needMillis ?: 0L),
        "dateNeeded" to (needMillis ?: 0L), "requiredOn" to (needMillis ?: 0L),
```

```
        "timestamp" to System.currentTimeMillis(), "time" to System.currentTimeMillis(),
        "id" to UUID.randomUUID().toString()
    )
    val json = Gson().toJson(payload)
    return Gson().fromJson(json, BloodRequest::class.java)
}

/** Read a String field by trying multiple names. */
private fun readString(any: Any, vararg names: String): String? = runCatching {
    val c = any::class.java
    for (n in names) try {
        val f = c.getDeclaredField(n); f.isAccessible = true
        (f.get(any) as? String)?.let { return it }
    } catch (_: NoSuchFieldException) {}
    null
}.getOrNull()

/** Read a Long/Number field by trying multiple names. */
private fun readLong(any: Any, vararg names: String): Long? = runCatching {
    val c = any::class.java
    for (n in names) try {
        val f = c.getDeclaredField(n); f.isAccessible = true
        val v = f.get(any)
        when (v) { is Long -> return v; is Int -> return v.toLong(); is Number -> return v.toLong(); is String -> return v.toLongOrNull() }
    } catch (_: NoSuchFieldException) {}
    null
}.getOrNull()
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\SchedulesScreen.kt

package com.example.androidbloodbank.ui

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo
import com.example.androidbloodbank.data.model.Schedule
import java.text.SimpleDateFormat
import java.util.*

@Composable
fun SchedulesScreen(repo: LocalRepo, onBack: () -> Unit) {
    val sdf = remember { SimpleDateFormat("yyyy-MM-dd", Locale.getDefault()) }
    val schedules = remember { repo.loadSchedules().toMutableStateList() }
    val donors = remember { repo.loadDonors() }
    var donorQuery by remember { mutableStateOf("") }
    var dateText by remember { mutableStateOf(sdf.format(Date())) }
    var notes by remember { mutableStateOf("") }
    var selectedDonorId by remember { mutableStateOf<String?>(donors.firstOrNull()?.id) }

    Column(modifier = Modifier.fillMaxSize().padding(16.dp).verticalScroll(rememberScrollState())) {
        Text("Donation Schedules", style = MaterialTheme.typography.headlineSmall)
        Spacer(Modifier.height(12.dp))

        Text("Select donor (search by name):")
         OutlinedTextField(value = donorQuery, onValueChange = { donorQuery = it }, label = { Text("Search name") }, modifier =
Modifier.fillMaxWidth())
        Spacer(Modifier.height(8.dp))
        Column {
            donors.filter { donorQuery.isBlank() || it.name.contains(donorQuery, ignoreCase = true) }.forEach { d ->
                Row(
                    modifier = Modifier.fillMaxWidth().clickable { selectedDonorId = d.id }.padding(8.dp),
                    verticalAlignment = androidx.compose.ui.Alignment.CenterVertically
                ) {
                    RadioButton(selected = d.id == selectedDonorId, onClick = { selectedDonorId = d.id })
                    Text("${d.name} (${d.bloodGroup.label})", modifier = Modifier.padding(start = 8.dp))
                }
            }
        }

        Spacer(Modifier.height(12.dp))
         OutlinedTextField(value = dateText, onValueChange = { dateText = it }, label = { Text("Date (yyyy-MM-dd)") }, modifier =
Modifier.fillMaxWidth())
        Spacer(Modifier.height(8.dp))
            OutlinedTextField(value = notes, onValueChange = { notes = it }, label = { Text("Notes (optional)") }, modifier =
Modifier.fillMaxWidth())

        Spacer(Modifier.height(12.dp))
```

```
        Button(onClick = {
            selectedDonorId?.let { did ->
                val s = Schedule(donorId = did, dateIso = dateText, notes = notes)
                schedules.add(s)
                repo.saveSchedules(schedules)
            }
        }, modifier = Modifier.fillMaxWidth()) { Text("Schedule Donation") }

        Spacer(Modifier.height(16.dp))
        Text("Upcoming schedules:", fontWeight = androidx.compose.ui.text.font.FontWeight.SemiBold)
        Spacer(Modifier.height(8.dp))
        if (schedules.isEmpty()) Text("No scheduled donations.")
        schedules.forEach { s ->
            val donor = donors.find { it.id == s.donorId }
            Card(modifier = Modifier.fillMaxWidth().padding(6.dp)) {
                Column(modifier = Modifier.padding(12.dp)) {
                    Text(donor?.name ?: "Unknown")
                    Text("Date: ${s.dateIso}")
                    if (!s.notes.isNullOrBlank()) Text("Notes: ${s.notes}")
                }
            }
        }

        Spacer(Modifier.height(12.dp))
        TextButton(onClick = onBack) { Text("Back") }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\SignupScreen.kt

package com.example.androidbloodbank.ui

import android.util.Patterns
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.outlined.ArrowBack
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.LocalRepo
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseAuthException
import com.google.firebase.database.FirebaseDatabase
import com.google.gson.Gson
import kotlinx.coroutines.launch
import kotlinx.coroutines.tasks.await
import kotlinx.coroutines.withTimeoutOrNull

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SignupScreen(
    repo: LocalRepo,
    onBack: () -> Unit,
    onSignupSuccess: () -> Unit
) {
    val snackbarHostState = remember { SnackbarHostState() }
    val scope = rememberCoroutineScope()

    val auth = remember { FirebaseAuth.getInstance() }
    val dbRef = remember { FirebaseDatabase.getInstance().reference } // optional profile save

    var name by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var phone by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var confirm by remember { mutableStateOf("") }
    var loading by remember { mutableStateOf(false) }

    fun isEmail(s: String) = Patterns.EMAIL_ADDRESS.matcher(s.trim()).matches()
    fun canSubmit(): Boolean {
        val emailOk = isEmail(email)
        val passOk = password.length >= 6 && password == confirm
        val nameOk = name.trim().length >= 2
        return emailOk && passOk && nameOk
    }

    suspend fun doSignup() {
        loading = true
```

```
    try {
        // 1) Create the account in Firebase Auth (20s timeout to avoid infinite spinner)
        val authResult = withTimeoutOrNull(20_000) {
            auth.createUserWithEmailAndPassword(email.trim(), password).await()
        }
        if (authResult == null) {
            snackbarHostState.showSnackbar("Signup timed out. Check internet and try again.")
            return
        }

        val uid = auth.currentUser?.uid.orEmpty()
        if (uid.isEmpty()) {
            snackbarHostState.showSnackbar("Signup failed: no UID returned.")
            return
        }

        // 2) Optional: store public profile to Realtime Database
        val profile = mapOf(
            "uid" to uid,
            "name" to name.trim(),
            "email" to email.trim(),
            "phone" to phone.trim().ifEmpty { null }
        )
        // Don’t block UX if DB write fails; but we try with timeout
        withTimeoutOrNull(10_000) {
            dbRef.child("users").child(uid).setValue(profile).await()
        }

        // 3) Optional: keep a tiny local session snapshot so your Splash/Gate logic works
        repo.saveCurrentUserJson(Gson().toJson(profile))

        snackbarHostState.showSnackbar("Account created.")
        onSignupSuccess()
    } catch (e: Exception) {
        val msg = when (e) {
            is FirebaseAuthException -> when (e.errorCode) {
                "ERROR_EMAIL_ALREADY_IN_USE" -> "This email is already in use."
                "ERROR_INVALID_EMAIL" -> "Invalid email address."
                "ERROR_WEAK_PASSWORD" -> "Weak password. Use 6+ characters."
                "ERROR_OPERATION_NOT_ALLOWED" -> "Email/Password sign-in not enabled in Firebase."
                "ERROR_NETWORK_REQUEST_FAILED" -> "Network error. Check your connection."
                else -> e.localizedMessage ?: "Sign up failed."
            }
            else -> e.localizedMessage ?: "Sign up failed."
        }
        snackbarHostState.showSnackbar(msg)
    } finally {
        loading = false
    }
}

Scaffold(
    topBar = {
        TopAppBar(
            title = { Text("Sign up") },
```

```
            navigationIcon = {
                IconButton(onClick = onBack) { Icon(Icons.Outlined.ArrowBack, contentDescription = "Back") }
            }
        )
    },
    snackbarHost = { SnackbarHost(hostState = snackbarHostState) }
) { padding ->
    Column(
        modifier = Modifier
            .padding(padding)
            .padding(16.dp)
            .fillMaxSize(),
        verticalArrangement = Arrangement.spacedBy(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        OutlinedTextField(
            value = name, onValueChange = { name = it },
            label = { Text("Full name") }, singleLine = true,
            modifier = Modifier.fillMaxWidth()
        )
        OutlinedTextField(
            value = email, onValueChange = { email = it },
            label = { Text("Email") }, singleLine = true,
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email),
            modifier = Modifier.fillMaxWidth()
        )
        OutlinedTextField(
            value = phone, onValueChange = { phone = it },
            label = { Text("Phone (optional)") }, singleLine = true,
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Phone),
            modifier = Modifier.fillMaxWidth()
        )
        OutlinedTextField(
            value = password, onValueChange = { password = it },
            label = { Text("Password (min 6 chars)") }, singleLine = true,
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
            visualTransformation = PasswordVisualTransformation(),
            modifier = Modifier.fillMaxWidth()
        )
        OutlinedTextField(
            value = confirm, onValueChange = { confirm = it },
            label = { Text("Confirm password") }, singleLine = true,
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
            visualTransformation = PasswordVisualTransformation(),
            modifier = Modifier.fillMaxWidth()
        )

        Button(
            onClick = { scope.launch { doSignup() } },
            enabled = canSubmit() && !loading,
            modifier = Modifier.fillMaxWidth().height(52.dp)
        ) {
            if (loading) {
                CircularProgressIndicator(strokeWidth = 2.dp, modifier = Modifier.size(18.dp))
                Spacer(Modifier.width(12.dp))
```

```
                Text("Creating account...")
            } else {
                Text("Create account")
            }
        }
    }
  }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\Theme.kt

package com.example.androidbloodbank.ui.theme

import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.darkColorScheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.graphics.Color

// ------- Soothing rose palette (light) -------
private val LightColors = lightColorScheme(
    primary = Color(0xFFE56B6F),
    onPrimary = Color(0xFFFFFFFF),
    secondary = Color(0xFFF3A0A7),
    onSecondary = Color(0xFF3F2326),
    tertiary = Color(0xFF7AB6B1),
    onTertiary = Color(0xFF103C39),
    background = Color(0xFFFFFBFB),
    surface = Color(0xFFFFFFFF),
    surfaceVariant = Color(0xFFF2E7E9),
    onSurface = Color(0xFF1E1E1F),
    onSurfaceVariant = Color(0xFF514A4C),
)

// ------- Cozy dark (no pure black) -------
private val DarkColors = darkColorScheme(
    primary = Color(0xFFFF8C95),
    onPrimary = Color(0xFF2C0A0E),
    secondary = Color(0xFFF2B8BD),
    onSecondary = Color(0xFF2D1316),
    tertiary = Color(0xFFAEDDD8),
    onTertiary = Color(0xFF062927),
    background = Color(0xFF0F1113),
    surface = Color(0xFF131517),
    surfaceVariant = Color(0xFF22282B),
    onSurface = Color(0xFFE3E6E8),
    onSurfaceVariant = Color(0xFFB2B8BC),
)

/** App theme entry point — THIS is what MainActivity should import. */
@Composable
fun AndroidBloodBankTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    dynamicColor: Boolean = false, // keep false so brand colors are stable
    content: @Composable () -> Unit
) {
    MaterialTheme(
        colorScheme = if (darkTheme) DarkColors else LightColors,
        typography = Typography,
        content = content
    )
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\Type.kt

package com.example.androidbloodbank.ui.theme

import androidx.compose.material3.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp

// Set of Material typography styles to start with
val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    )
    /* Other default text styles to override
    titleLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 22.sp,
        lineHeight = 28.sp,
        letterSpacing = 0.sp
    ),
    labelSmall = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Medium,
        fontSize = 11.sp,
        lineHeight = 16.sp,
        letterSpacing = 0.5.sp
    )
    */
)
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\components\BloodGroupChips.kt

package com.example.androidbloodbank.ui.components

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.model.BloodGroup

@Composable
fun BloodGroupChips(
    selected: BloodGroup?,
    onSelected: (BloodGroup) -> Unit,
    modifier: Modifier = Modifier
) {
    // Order to match your mock
    val order = listOf(
        BloodGroup.A_POS, BloodGroup.A_NEG, BloodGroup.B_POS, BloodGroup.B_NEG,
        BloodGroup.O_POS, BloodGroup.O_NEG, BloodGroup.AB_POS, BloodGroup.AB_NEG
    )

    Column(modifier = modifier, verticalArrangement = Arrangement.spacedBy(12.dp)) {
        // Two tidy rows of 4 chips (no extra libs)
        Row(horizontalArrangement = Arrangement.spacedBy(12.dp), modifier = Modifier.fillMaxWidth()) {
            order.take(4).forEach { bg ->
                FilterChip(
                    selected = selected == bg,
                    onClick = { onSelected(bg) },
                    label = { Text(bg.label) }
                )
            }
        }
        Row(horizontalArrangement = Arrangement.spacedBy(12.dp), modifier = Modifier.fillMaxWidth()) {
            order.drop(4).forEach { bg ->
                FilterChip(
                    selected = selected == bg,
                    onClick = { onSelected(bg) },
                    label = { Text(bg.label) }
                )
            }
        }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\components\BloodGroupSelector.kt

package com.example.androidbloodbank.ui.components

import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import com.example.androidbloodbank.data.model.BloodGroup

@Composable
fun BloodGroupSelector(
    value: BloodGroup,
    onChange: (BloodGroup) -> Unit,
    modifier: Modifier = Modifier
) {
    Column(modifier = modifier) {
        Text("Blood Group:", style = MaterialTheme.typography.bodyLarge)
        Spacer(Modifier.height(8.dp))
        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            BloodGroup.values().forEach { bg ->
                val selected = bg == value
                Box(
                    modifier = Modifier
                        .padding(4.dp)
                        .clickable { onChange(bg) }
                        .background(
                            if (selected) MaterialTheme.colorScheme.primary else Color.LightGray,
                            RoundedCornerShape(8.dp)
                        )
                        .padding(horizontal = 12.dp, vertical = 8.dp)
                ) {
                    Text(
                        bg.label,
                        color = if (selected) Color.White else Color.Black
                    )
                }
            }
        }
    }
}
```

```
// File: app\src\main\java\com\example\androidbloodbank\ui\theme\components\BottomNavBar.kt

package com.example.androidbloodbank.ui.components

import androidx.compose.material3.NavigationBar
import androidx.compose.material3.NavigationBarItem
import androidx.compose.material3.Icon
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.navigation.NavHostController
import androidx.navigation.compose.currentBackStackEntryAsState
import androidx.compose.ui.unit.sp

data class NavBarItem(
    val route: String,
    val label: String,
    val icon: ImageVector
)

@Composable
fun BottomNavBar(navController: NavHostController, items: List<NavBarItem>) {
    val navBackStackEntry by navController.currentBackStackEntryAsState()
    val currentRoute = navBackStackEntry?.destination?.route

    NavigationBar {
        items.forEach { item ->
            NavigationBarItem(
                selected = currentRoute == item.route,
                onClick = {
                    if (currentRoute != item.route) {
                        navController.navigate(item.route) {
                            // Pop up to the start to avoid building a huge back stack
                            popUpTo(navController.graph.startDestinationId) { saveState = true }
                            // Avoid multiple copies
                            launchSingleTop = true
                            restoreState = true
                        }
                    }
                },
                icon = { Icon(item.icon, contentDescription = item.label) },
                label = { Text(item.label, fontSize = 12.sp) }
            )
        }
    }
}
```

// File: app\src\main\java\com\example\androidbloodbank\ui\theme\components\PostRequestScreen.kt

```
// File: app\src\test\java\com\example\androidbloodbank\ExampleUnitTest.kt

package com.example.androidbloodbank

import org.junit.Test

import org.junit.Assert.*

/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
        assertEquals(4, 2 + 2)
    }
}
```