**Help Sheet for lab 1**

Lab 1 Android project skeleton in Java using an Empty Views Activity. It includes:

* Patient Registration (with NHS number validation, Room DB)
* An Admin Portal (staff registration + list)
* RBAC (only Admins can access the Admin Portal; first-run bootstrap allows you to create the first Admin)
* Simple “current user” session via SharedPreferences

Project/app name: HospiManagmenetApp (Please use the same name)

Create a new Empty Views Activity project in Android Studio called HospiManagmenetApp, then replace/add files as below. Some of the files may require changing. Pay close attention to the imports as they give you information regarding this project skeleton’ s file structure.

**settings.gradle (Project)**

pluginManagement { // Configure how Gradle locates plugins used in the build

repositories { // Define the repositories that will host Gradle plugins

google { // Use Google's Maven repository for Android/Google plugins

content { // Restrict what groups are fetched from this repo (for speed and correctness)

includeGroupByRegex("com\\.android.\*") // Only include plugin groups starting with 'com.android'

includeGroupByRegex("com\\.google.\*") // Only include plugin groups starting with 'com.google'

includeGroupByRegex("androidx.\*") // Only include plugin groups starting with 'androidx'

}

}

mavenCentral() // Also search Maven Central for plugins

gradlePluginPortal() // And the official Gradle Plugin Portal

}

}

dependencyResolutionManagement { // Control how project dependencies (not plugins) are resolved

repositoriesMode.set(RepositoriesMode.FAIL\_ON\_PROJECT\_REPOS) // Disallow module-level repos; enforce centralised repos here

repositories { // The only repositories allowed for dependency resolution

google() // Google's Maven repo for Android/Google libraries

mavenCentral() // Maven Central for the wider JVM ecosystem

}

}

rootProject.name = "HospiManagmenetApp" // The name shown for the root project in Gradle/IDE

include(":app") // Include the ':app' module in this multi-module build

**app/build.gradle (Module) (Note that Lab 2 dependencies are included. Change sdk in the context of your project)**

plugins { // Gradle plugins applied to this module

alias(libs.plugins.android.application) // Use the Android Application plugin via the version catalogue alias

}

android { // Android-specific build configuration for the app module

namespace = "com.example.hospimanagmenetapp" // Package namespace used for R classes & manifest

compileSdk = 36 // Compile against Android API level 36 (must match an installed SDK)

defaultConfig { // Default settings for all build variants

applicationId = "com.example.hospimanagmenetapp" // Unique app ID used on devices/Play Store

minSdk = 24 // Minimum Android version the app supports (Android 7.0)

targetSdk = 36 // Target behaviour for API level 36 (opt-in to latest platform changes)

versionCode = 1 // Internal version (integer) used for updates

versionName = "1.0" // Human-readable version shown to users

testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner" // Runner for instrumented (device) tests

}

buildTypes { // Definitions of build variants like debug/release

release { // Settings for the release (shipping) build

isMinifyEnabled = false // Disable code shrinking/obfuscation (set true for production usually)

proguardFiles( // ProGuard/R8 rules used when shrinking/optimising

getDefaultProguardFile("proguard-android-optimize.txt"), // Standard optimised rules supplied by the Android plugin

"proguard-rules.pro" // Your custom rules file

)

}

}

buildFeatures { // Toggle optional Android build features

viewBinding = true // Generate binding classes for views to avoid findViewById

}

compileOptions { // Java language level the compiler targets/accepts

sourceCompatibility = JavaVersion.VERSION\_11 // Allow Java 11 language features in source

targetCompatibility = JavaVersion.VERSION\_11 // Compile bytecode compatible with Java 11

}

}

dependencies { // Libraries this module depends on

implementation(libs.appcompat) // AndroidX AppCompat for backwards-compatible UI components

implementation(libs.material) // Material Components for Android

implementation(libs.activity) // AndroidX Activity KTX utilities

implementation(libs.constraintlayout) // ConstraintLayout for flexible UI layouts

// Lab 1 UI lists

implementation("androidx.recyclerview:recyclerview:1.3.2") // RecyclerView for efficient scrolling lists/grids

// Room (Java -> annotationProcessor)

implementation("androidx.room:room-runtime:2.6.1") // Room runtime for SQLite ORM

annotationProcessor("androidx.room:room-compiler:2.6.1") // Annotation processor generating Room DAOs/entities

// Lifecycle (already had runtime + livedata) + ViewModel for Lab 2

implementation("androidx.lifecycle:lifecycle-runtime:2.8.6") // Lifecycle-aware components base runtime

implementation("androidx.lifecycle:lifecycle-livedata:2.8.6") // LiveData for observable data holders

implementation("androidx.lifecycle:lifecycle-viewmodel:2.8.6") // ViewModel for UI-related state

// SQLCipher (Lab 1 basic-at-rest security placeholder)

implementation("net.zetetic:android-database-sqlcipher:4.5.4") // Encrypted SQLite via SQLCipher

implementation("androidx.sqlite:sqlite:2.4.0") // AndroidX SQLite wrappers/utilities

// --- Lab 2: Network stack (Retrofit/OkHttp/Gson) ---

implementation("com.squareup.retrofit2:retrofit:2.11.0") // Retrofit HTTP client for type-safe APIs

implementation("com.squareup.retrofit2:converter-gson:2.11.0") // Gson converter for JSON <-> objects

implementation("com.squareup.okhttp3:okhttp:4.12.0") // OkHttp core HTTP client

implementation("com.squareup.okhttp3:logging-interceptor:4.12.0") // Interceptor for request/response logging

// --- Lab 2: Biometric authentication ---

implementation("androidx.biometric:biometric:1.2.0-alpha05") // BiometricPrompt APIs for fingerprint/face auth

// Tests (version catalogue)

testImplementation(libs.junit) // JUnit 4 for local unit tests

androidTestImplementation(libs.ext.junit) // AndroidX JUnit extensions for instrumented tests

androidTestImplementation(libs.espresso.core) // Espresso UI testing framework

}

**Manifest file (some features require commenting out as they are required only for Lab 2)**

<?xml version="1.0" encoding="utf-8"?>

<manifest

xmlns:android="http://schemas.android.com/apk/res/android" <!-- XML namespace for Android attributes -->

package="com.example.hospimanagmenetapp"> <!-- App’s package name (also the default appId in manifests); -->

<!-- Allows all outbound network access for Retrofit/OkHttp -->

<uses-permission android:name="android.permission.INTERNET" />

<!-- Biometric permission (normal, auto-granted on API 28+). Not strictly required for BiometricPrompt, but harmless. -->

<uses-permission android:name="android.permission.USE\_BIOMETRIC" />

<application

android:allowBackup="false" <!-- Disables ADB/auto backup of app data; good for privacy -->

android:icon="@mipmap/ic\_launcher" <!-- Launcher icon resource -->

android:label="HospiManagmenetApp" <!-- App name shown to users -->

android:roundIcon="@mipmap/ic\_launcher\_round" <!-- Round icon for devices that use it -->

android:supportsRtl="true" <!-- Enable right-to-left layout mirroring -->

android:theme="@style/Theme.HospiManagmenetApp" <!-- Global app theme -->

android:networkSecurityConfig="@xml/network\_security\_config"> <!-- Custom network security (e.g., allow debug cleartext, custom CAs) -->

<!-- Launcher activity: entry point shown on the home screen -->

<activity

android:name=".MainActivity" <!-- Fully qualified class via relative package path -->

android:exported="true"> <!-- Required true for activities with intent filters on API 31+ -->

<intent-filter> <!-- Declares this activity as a launcher target -->

<action android:name="android.intent.action.MAIN" /> <!-- Main entry action -->

<category android:name="android.intent.category.LAUNCHER" /> <!-- Puts icon in the launcher -->

</intent-filter>

</activity>

<!-- Patient registration screen; no external entry (internal only) -->

<activity

android:name=".ui.PatientRegistrationActivity"

android:exported="false" <!-- Not launchable by other apps -->

android:parentActivityName=".MainActivity" /> <!-- Enables Up navigation back to MainActivity -->

<!-- Admin login screen; internal only -->

<activity

android:name=".ui.AdminLoginActivity"

android:exported="false"

android:parentActivityName=".MainActivity" />

<!-- Admin portal; returns “Up” to AdminLogin by default -->

<activity

android:name=".ui.AdminPortalActivity"

android:exported="false"

android:parentActivityName=".ui.AdminLoginActivity" />

<!-- To return directly to Main instead, set parentActivityName=".MainActivity" -->

<!-- Appointments host; contains list/booking fragments; internal only -->

<activity

android:name=".ui.AppointmentActivity"

android:exported="false"

android:parentActivityName=".MainActivity" />

</application>

</manifest>

**Theme (created by Android Studio)**

**Layouts (XML)**

**activity\_main.xml**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" <!-- XML namespace for Android view attributes -->

android:orientation="vertical" <!-- Stack children top-to-bottom -->

android:padding="24dp" <!-- Inner spacing around the layout edges -->

android:layout\_width="match\_parent" <!-- Fill the available width -->

android:layout\_height="match\_parent"> <!-- Fill the available height -->

<TextView

android:id="@+id/tvWelcome" <!-- View ID for code/view binding -->

android:text="Welcome" <!-- Text displayed to the user -->

android:textStyle="bold" <!-- Bold styling -->

android:textSize="18sp" <!-- Font size in sp (scales with user preference) -->

android:layout\_width="wrap\_content" <!-- Size to fit the text -->

android:layout\_height="wrap\_content"/> <!-- Size to fit the text -->

<Button

android:id="@+id/btnPatientRegistration" <!-- ID to reference this button -->

android:text="Patient Registration" <!-- Label shown on the button -->

android:layout\_marginTop="24dp" <!-- Space above this button -->

android:layout\_width="match\_parent" <!-- Make the button full-width -->

android:layout\_height="wrap\_content"/> <!-- Height adjusts to content -->

<Button

android:id="@+id/btnAdminPortal"

android:text="Admin Portal"

android:layout\_marginTop="12dp" <!-- Slightly smaller gap than first button -->

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<Button

android:id="@+id/btnLogout"

android:text="Clear Session" <!-- Clearer action label than “Logout” for demos -->

android:layout\_marginTop="12dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

</LinearLayout>

**activity\_patient\_registration.xml (You can add as many fields as you wish here)**

<?xml version="1.0" encoding="utf-8"?>

<ScrollView xmlns:android="http://schemas.android.com/apk/res/android" <!-- Allows the form to scroll on smaller screens -->

android:fillViewport="true" <!-- Let content expand to full height before scrolling -->

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<LinearLayout

android:orientation="vertical" <!-- Stack fields top-to-bottom -->

android:padding="24dp" <!-- Comfortable form padding -->

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<EditText

android:id="@+id/etNhs" <!-- NHS number input -->

android:hint="NHS Number (10 digits)" <!-- Placeholder text shown when empty -->

android:inputType="number" <!-- Numeric keyboard (digits only) -->

android:maxLength="14" <!-- Room for optional spaces; pure 10 digits still fits -->

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<EditText

android:id="@+id/etFullName" <!-- Patient full name -->

android:hint="Full Name"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<EditText

android:id="@+id/etDob" <!-- Date of birth -->

android:hint="Date of Birth (yyyy-MM-dd)" <!-- Desired format hint -->

android:inputType="date" <!-- Date-friendly keyboard; consider a DatePicker -->

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<EditText

android:id="@+id/etPhone" <!-- Phone number -->

android:hint="Phone"

android:inputType="phone" <!-- Phone keypad (includes + and # etc.) -->

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<EditText

android:id="@+id/etEmail" <!-- Email address -->

android:hint="Email"

android:inputType="textEmailAddress" <!-- Email-optimised keyboard and validation hints -->

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<Button

android:id="@+id/btnSavePatient" <!-- Submit/save action -->

android:text="Save Patient"

android:layout\_marginTop="16dp" <!-- Space above the button -->

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

</LinearLayout>

</ScrollView>

**activity\_admin\_login.xml**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:padding="24dp"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView

android:text="Admin Login"

android:textStyle="bold"

android:textSize="18sp"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<EditText

android:id="@+id/etAdminEmail"

android:hint="Admin Email"

android:inputType="textEmailAddress"

android:layout\_marginTop="16dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<EditText

android:id="@+id/etAdminPin"

android:hint="Admin PIN"

android:inputType="numberPassword"

android:layout\_marginTop="8dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<Button

android:id="@+id/btnAdminLogin"

android:text="Login"

android:layout\_marginTop="16dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<Button

android:id="@+id/btnOpenAdminSetup"

android:text="Open Admin Portal (Setup Mode)"

android:layout\_marginTop="8dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:enabled="false"/>

</LinearLayout>

**activity\_admin\_portal.xml**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:padding="16dp"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView

android:text="Admin Portal — Staff Registration (RBAC)"

android:textStyle="bold"

android:textSize="18sp"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<EditText

android:id="@+id/etStaffName"

android:hint="Full Name"

android:layout\_marginTop="12dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<EditText

android:id="@+id/etStaffEmail"

android:hint="Email"

android:inputType="textEmailAddress"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<Spinner

android:id="@+id/spRole"

android:layout\_marginTop="8dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<EditText

android:id="@+id/etAdminSetupPin"

android:hint="Admin PIN (required if role = ADMIN)"

android:inputType="numberPassword"

android:layout\_marginTop="8dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<Button

android:id="@+id/btnRegisterStaff"

android:text="Register Staff"

android:layout\_marginTop="12dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<Button

android:id="@+id/btnRefreshList"

android:text="Refresh List"

android:layout\_marginTop="8dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

<TextView

android:text="Registered Staff"

android:textStyle="bold"

android:layout\_marginTop="16dp"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<androidx.recyclerview.widget.RecyclerView

android:id="@+id/rvStaff"

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_marginTop="8dp"

android:layout\_weight="1"/>

</LinearLayout>

**item\_staff.xml**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:padding="12dp"

android:background="?android:attr/selectableItemBackground"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<TextView

android:id="@+id/tvStaffName"

android:textStyle="bold"

android:text="Name"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<TextView

android:id="@+id/tvStaffEmail"

android:text="email@example.com"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<TextView

android:id="@+id/tvStaffRole"

android:text="ROLE"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

</LinearLayout>

**Strings (optional for the time being)**

**Java classes to implement the above** (Edit and change according to your requirements. Ensure you follow your project structure)

**AppDatabase.java**

package com.example.hospimanagmenetapp.data; // Package for data-layer classes

import androidx.room.Database; // Room annotation to define the DB schema

import androidx.room.Room; // Factory for creating Room databases

import androidx.room.RoomDatabase; // Base class for Room databases

import android.content.Context; // Needed to build the DB with an app Context

import com.example.hospimanagmenetapp.data.dao.PatientDao; // DAO for Patient operations

import com.example.hospimanagmenetapp.data.dao.StaffDao; // DAO for Staff operations

import com.example.hospimanagmenetapp.data.entities.Patient; // Entity mapped to a table

import com.example.hospimanagmenetapp.data.entities.Staff; // Entity mapped to a table

@Database(entities = {Patient.class, Staff.class}, version = 1, exportSchema = false)

// Declares the Room database: which entities it manages, the schema version,

// and whether to export the schema as JSON for tooling (false = do not export).

public abstract class AppDatabase extends RoomDatabase { // Concrete DB extends RoomDatabase

// Singleton instance (volatile ensures visibility across threads)

private static volatile AppDatabase INSTANCE;

// Room generates the implementation; these expose your DAOs to callers

public abstract PatientDao patientDao();

public abstract StaffDao staffDao();

// Thread-safe double-checked locking to get/create the singleton DB

public static AppDatabase getInstance(Context context) {

if (INSTANCE == null) { // Fast path: already created?

synchronized (AppDatabase.class) { // Serialise creation across threads

if (INSTANCE == null) { // Second check inside the lock

INSTANCE = Room.databaseBuilder(

context.getApplicationContext(), // Use app Context to avoid Activity leaks

AppDatabase.class, // The RoomDatabase subclass to create

"hms\_db" // On-device filename for the DB

)

.fallbackToDestructiveMigration() // Wipes & rebuilds on version change if no migration (dev-friendly, data-loss risk)

.build(); // Build the database instance

}

}

}

return INSTANCE; // Return the shared database

}

}

**Patient.java (For dates ISO formatting has been applied so, change to British format)**

package com.example.hospimanagmenetapp.data.entities; // Entity lives in the data.entities package

import androidx.annotation.NonNull; // Annotation to mark fields that must not be null

import androidx.room.Entity; // Marks this class as a Room table

import androidx.room.Index; // Allows creating DB indices for faster lookups/uniqueness

import androidx.room.PrimaryKey; // Identifies the primary key column

@Entity(

tableName = "patients", // Actual SQLite table name

indices = {@Index(value = {"nhsNumber"}, unique = true)} // Unique index so each NHS number appears only once

)

public class Patient {

@PrimaryKey(autoGenerate = true) // Auto-incremented surrogate key

public long id; // Local DB identifier

@NonNull // Must not be null; Room will enforce at runtime

public String nhsNumber; // NHS number (store digits only; format/validate in code)

public String fullName; // Patient’s full name (consider @NonNull if mandatory)

public String dateOfBirth; // ISO yyyy-MM-dd for simplicity; a TypeConverter to Date is cleaner

public String phone; // Contact number (normalised/validated in code)

public String email; // Contact email (basic pattern check in code/UI)

public long createdAt; // Unix epoch millis when the record was created

public long updatedAt; // Unix epoch millis when the record was last updated

}

**PatientDao.java**

package com.example.hospimanagmenetapp.data.dao; // DAO (Data Access Object) package

import androidx.room.Dao; // Marks this interface as a Room DAO

import androidx.room.Insert; // Annotation for insert operations

import androidx.room.OnConflictStrategy; // How Room should behave on key/constraint conflicts

import androidx.room.Query; // Annotation for custom SQL queries

import com.example.hospimanagmenetapp.data.entities.Patient; // Entity this DAO operates on

@Dao // Tells Room to generate the implementation at compile time

public interface PatientDao {

@Insert(onConflict = OnConflictStrategy.ABORT) // Insert a Patient; fail if a constraint (e.g., unique NHS) is violated

long insert(Patient patient); // Returns the new row ID (or -1 if ignored, depending on strategy)

@Query("SELECT COUNT(\*) FROM patients WHERE nhsNumber = :nhsNumber") // Parameterised SQL; :nhsNumber is bound from the method arg

int countByNhs(String nhsNumber); // Quick existence check (0 = none, >0 = exists)

}

**StaffDao.java**

package com.example.hospimanagmenetapp.data.dao; // DAO (Data Access Object) package for Room

import androidx.room.Dao; // Marks this interface as a Room DAO

import androidx.room.Insert; // Annotation for insert operations

import androidx.room.OnConflictStrategy; // Policy for handling conflicts (e.g., unique constraints)

import androidx.room.Query; // Annotation for custom SQL queries

import com.example.hospimanagmenetapp.data.entities.Staff; // Entity this DAO operates on

import java.util.List; // Used for returning multiple results

@Dao // Instructs Room to generate the implementation at compile time

public interface StaffDao {

@Insert(onConflict = OnConflictStrategy.ABORT) // Insert a Staff record; fail if a constraint is violated

long insert(Staff staff); // Returns the new row ID (throws on conflict with ABORT)

@Query("SELECT \* FROM staff ORDER BY fullName ASC") // Fetch all staff ordered alphabetically by full name

List<Staff> getAll(); // Returns a snapshot list (call from background thread)

@Query("SELECT COUNT(\*) FROM staff WHERE role = 'ADMIN'") // Count staff whose role is ADMIN

int countAdmins(); // Useful for gating admin features/bootstrapping

@Query("SELECT \* FROM staff WHERE email = :email LIMIT 1") // Look up a single staff member by email

Staff findByEmail(String email); // Returns null if not found (handle in caller)

}

**ValidationUtils.java**

package com.example.hospimanagmenetapp.util; // Utility classes for the app live here

public class ValidationUtils { // Simple holder for validation helper methods

// NHS number validation (exactly 10 digits, with Mod 11 checksum)

public static boolean validateNhsNumber(String nhs) { // Returns true if the NHS number is valid

if (nhs == null) return false; // Null input cannot be valid

String digits = nhs.replaceAll("\\s+", ""); // Strip all whitespace so "123 456 7890" still works

if (!digits.matches("\\d{10}")) return false; // Must be exactly 10 numeric digits

int sum = 0; // Accumulator for weighted sum across first 9 digits

for (int i = 0; i < 9; i++) { // Process digits 0..8 (the 10th is the check digit)

int d = digits.charAt(i) - '0'; // Convert character to its integer value

sum += d \* (10 - i); // Weighting: 10 for first digit down to 2 for ninth

}

int check = 11 - (sum % 11); // Compute raw check value per Mod 11 rule

if (check == 11) check = 0; // If remainder produced 11, check digit is 0

if (check == 10) return false; // A result of 10 is invalid per NHS rules

int provided = digits.charAt(9) - '0'; // Extract the provided check digit (last digit)

return check == provided; // Valid only if calculated and provided digits match

}

}

**SessionManager.java**

package com.example.hospimanagmenetapp.util; // Utility classes (non-UI helpers) live here

import android.content.Context; // Needed to obtain SharedPreferences

import android.content.SharedPreferences; // Key–value storage for simple app state

public class SessionManager { // Simple wrapper around SharedPreferences for session data

private static final String PREF = "hms\_prefs"; // Preferences file name

private static final String KEY\_ROLE = "current\_role"; // Key for stored user role

private static final String KEY\_EMAIL = "current\_email"; // Key for stored user email

public static void setCurrentUser(Context ctx, String role, String email) {

SharedPreferences sp = ctx.getSharedPreferences(PREF, Context.MODE\_PRIVATE); // Private prefs for this app only

sp.edit() // Begin an edit transaction

.putString(KEY\_ROLE, role) // Save the user’s role (e.g., "ADMIN", "STAFF")

.putString(KEY\_EMAIL, email) // Save the user’s email address

.apply(); // Apply asynchronously (non-blocking)

}

public static String getCurrentRole(Context ctx) {

SharedPreferences sp = ctx.getSharedPreferences(PREF, Context.MODE\_PRIVATE); // Open prefs

return sp.getString(KEY\_ROLE, ""); // Return role or empty string if none set

}

public static String getCurrentEmail(Context ctx) {

SharedPreferences sp = ctx.getSharedPreferences(PREF, Context.MODE\_PRIVATE); // Open prefs

return sp.getString(KEY\_EMAIL, ""); // Return email or empty string if none set

}

public static void clear(Context ctx) {

SharedPreferences sp = ctx.getSharedPreferences(PREF, Context.MODE\_PRIVATE); // Open prefs

sp.edit().clear().apply(); // Remove all stored keys for a full “logout/reset”

}

}

**MainActivity.java (Driver Class)**

package com.example.hospimanagmenetapp; // App's root package

import androidx.appcompat.app.AppCompatActivity; // Base class for modern Activities with ActionBar support

import android.content.Intent; // Used to navigate between Activities

import android.os.Bundle; // Holds saved instance state for lifecycle

import android.widget.Button; // UI widget: Button

import android.widget.TextView; // UI widget: TextView

import com.example.hospimanagmenetapp.ui.AdminLoginActivity; // Screen for admin sign-in

import com.example.hospimanagmenetapp.ui.AdminPortalActivity; // Screen for admin features (opened after login)

import com.example.hospimanagmenetapp.ui.PatientRegistrationActivity; // Screen to register patients

import com.example.hospimanagmenetapp.util.SessionManager; // Helper for simple session storage

public class MainActivity extends AppCompatActivity { // Entry Activity shown at app launch

private TextView tvWelcome; // Header showing session state

private Button btnPatientRegistration, btnAdminPortal, btnLogout; // Main menu buttons

@Override

protected void onCreate(Bundle savedInstanceState) { // Lifecycle: called when Activity is created

super.onCreate(savedInstanceState); // Always call the superclass first

setContentView(R.layout.activity\_main); // Inflate the layout defined in activity\_main.xml

// Bind views from the layout to fields

tvWelcome = findViewById(R.id.tvWelcome);

btnPatientRegistration = findViewById(R.id.btnPatientRegistration);

btnAdminPortal = findViewById(R.id.btnAdminPortal);

btnLogout = findViewById(R.id.btnLogout);

refreshHeader(); // Show current sign-in state immediately

// Navigate to the Patient Registration screen

btnPatientRegistration.setOnClickListener(v ->

startActivity(new Intent(this, PatientRegistrationActivity.class)));

// Navigate to Admin login first (RBAC gate). AdminPortal follows after successful login.

btnAdminPortal.setOnClickListener(v -> {

// RBAC gate: ADMIN only. If no admin exists yet, AdminLogin can handle bootstrap.

Intent i = new Intent(this, AdminLoginActivity.class);

startActivity(i);

});

// Clear session and update the header (acts like a simple "log out")

btnLogout.setOnClickListener(v -> {

SessionManager.clear(this); // Remove stored role/email

refreshHeader(); // Reflect the logged-out state in the UI

});

}

// Update the welcome header with the current session info

private void refreshHeader() {

String role = SessionManager.getCurrentRole(this); // Read stored role (e.g., "ADMIN")

String email = SessionManager.getCurrentEmail(this); // Read stored email

if (role == null || role.isEmpty()) { // No session present

tvWelcome.setText("Welcome (not signed in)"); // Guest view

} else {

tvWelcome.setText("Signed in: " + email + " (" + role + ")"); // Show who is signed in

}

}

@Override

protected void onResume() { // Lifecycle: called when Activity comes to the foreground

super.onResume(); // Always call the superclass

refreshHeader(); // Re-read session in case it changed while away

}

}

**PatientRegistrationActivity.java**

package com.example.hospimanagmenetapp.ui; // UI layer package for Activities

import androidx.appcompat.app.AppCompatActivity; // Base class for Activities with AppCompat support

import android.os.Bundle; // Lifecycle state bundle

import android.text.TextUtils; // Utility for simple string emptiness checks

import android.widget.Button; // UI widget: Button

import android.widget.EditText; // UI widget: text input

import android.widget.Toast; // Lightweight user notifications

import com.example.hospimanagmenetapp.R; // Resource IDs (layouts, strings, etc.)

import com.example.hospimanagmenetapp.data.AppDatabase; // Room database singleton

import com.example.hospimanagmenetapp.data.entities.Patient; // Entity to persist

import com.example.hospimanagmenetapp.util.ValidationUtils; // NHS number validator

import java.util.concurrent.Executors; // For running DB work off the main thread

public class PatientRegistrationActivity extends AppCompatActivity { // Screen to capture and save a patient

private EditText etNhs, etFullName, etDob, etPhone, etEmail; // Form inputs

private Button btnSave; // Save action

@Override

protected void onCreate(Bundle savedInstanceState) { // Activity creation lifecycle

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_patient\_registration); // Inflate the registration form layout

// Bind views to fields

etNhs = findViewById(R.id.etNhs);

etFullName = findViewById(R.id.etFullName);

etDob = findViewById(R.id.etDob);

etPhone = findViewById(R.id.etPhone);

etEmail = findViewById(R.id.etEmail);

btnSave = findViewById(R.id.btnSavePatient);

btnSave.setOnClickListener(v -> savePatient()); // When tapped, validate and persist the patient

}

// Validate inputs and insert the patient into Room on a background thread

private void savePatient() {

// Read and trim user input

String nhs = etNhs.getText().toString().trim();

String name = etFullName.getText().toString().trim();

String dob = etDob.getText().toString().trim();

String phone = etPhone.getText().toString().trim();

String email = etEmail.getText().toString().trim();

// Basic required-field validation

if (TextUtils.isEmpty(nhs) || TextUtils.isEmpty(name) || TextUtils.isEmpty(dob)) {

Toast.makeText(this, "NHS number, name, and DOB are required.", Toast.LENGTH\_SHORT).show();

return; // Stop here; user must complete the required fields

}

// Validate the NHS number using Mod 11 rules

if (!ValidationUtils.validateNhsNumber(nhs)) {

Toast.makeText(this, "Invalid NHS number.", Toast.LENGTH\_SHORT).show();

return; // Do not proceed with invalid identifiers

}

// Run database I/O off the main thread to keep the UI responsive

Executors.newSingleThreadExecutor().execute(() -> {

try {

AppDatabase db = AppDatabase.getInstance(getApplicationContext()); // Get the Room singleton

// Enforce uniqueness by NHS number before inserting

if (db.patientDao().countByNhs(nhs) > 0) {

runOnUiThread(() ->

Toast.makeText(this, "Patient with this NHS number already exists.", Toast.LENGTH\_SHORT).show());

return; // Abort insert; duplicate detected

}

// Map form inputs to a new Patient entity

Patient p = new Patient();

p.nhsNumber = nhs;

p.fullName = name;

p.dateOfBirth = dob; // Consider normalising/validating format upstream

p.phone = phone;

p.email = email;

long now = System.currentTimeMillis(); // Timestamp fields in epoch millis

p.createdAt = now;

p.updatedAt = now;

db.patientDao().insert(p); // Persist to the local database

// Notify success and close the screen

runOnUiThread(() -> {

Toast.makeText(this, "Patient saved.", Toast.LENGTH\_SHORT).show();

finish(); // Return to the previous screen

});

} catch (Exception e) {

// Generic error path (e.g., SQLite constraint, I/O issues)

runOnUiThread(() ->

Toast.makeText(this, "Error saving patient.", Toast.LENGTH\_SHORT).show());

}

});

}

}

**AdminLoginActivity.java**

package com.example.hospimanagmenetapp.ui; // UI layer package for Activities

import androidx.appcompat.app.AppCompatActivity; // Base class for Activities with AppCompat features

import android.content.Intent; // For navigating between Activities

import android.os.Bundle; // Lifecycle state bundle

import android.text.TextUtils; // Simple string checks (e.g., isEmpty)

import android.widget.Button; // UI widget: Button

import android.widget.EditText; // UI widget: text input

import android.widget.Toast; // Lightweight on-screen notifications

import com.example.hospimanagmenetapp.R; // Resource references (layouts, IDs, strings)

import com.example.hospimanagmenetapp.data.AppDatabase; // Room database singleton

import com.example.hospimanagmenetapp.data.entities.Staff; // Staff entity (contains role and PIN)

import com.example.hospimanagmenetapp.util.SessionManager; // Simple session storage (SharedPreferences)

import java.util.concurrent.Executors; // Run DB work off the main thread

public class AdminLoginActivity extends AppCompatActivity { // Screen for admin authentication

private EditText etEmail, etPin; // Inputs for admin email/PIN

private Button btnLogin, btnOpenSetup; // Actions: login or open bootstrap/setup

@Override

protected void onCreate(Bundle savedInstanceState) { // Called when the Activity is created

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_admin\_login); // Inflate the admin login layout

// Bind views

etEmail = findViewById(R.id.etAdminEmail);

etPin = findViewById(R.id.etAdminPin);

btnLogin = findViewById(R.id.btnAdminLogin);

btnOpenSetup = findViewById(R.id.btnOpenAdminSetup);

// Attach handlers

btnLogin.setOnClickListener(v -> doLogin()); // Attempt sign-in with provided email/PIN

btnOpenSetup.setOnClickListener(v -> openAdminPortal()); // Allow bootstrap into Admin Portal (first admin registration)

}

@Override

protected void onResume() { // Called when the Activity returns to foreground

super.onResume();

// Check if any ADMIN exists; if none, enable the setup button so you can create the first admin

Executors.newSingleThreadExecutor().execute(() -> {

int admins = AppDatabase.getInstance(getApplicationContext()).staffDao().countAdmins(); // Background DB call

runOnUiThread(() -> btnOpenSetup.setEnabled(admins == 0)); // UI update must be on main thread

});

}

// Navigate directly to the Admin Portal in "bootstrap" mode

private void openAdminPortal() {

Intent i = new Intent(this, AdminPortalActivity.class); // Intent to open Admin Portal

i.putExtra("bypassCheck", true); // Flag so the portal can skip normal prechecks for first-time setup

startActivity(i); // Launch the portal

finish(); // Close login so back won’t return here

}

// Validate inputs and perform admin login using background DB lookup

private void doLogin() {

String email = etEmail.getText().toString().trim(); // Read/trim email

String pin = etPin.getText().toString().trim(); // Read/trim PIN

// Basic required-field checks

if (TextUtils.isEmpty(email) || TextUtils.isEmpty(pin)) {

Toast.makeText(this, "Email and PIN are required.", Toast.LENGTH\_SHORT).show();

return; // Don’t proceed without both fields

}

// Run the lookup off the main thread (Room requirement / UI responsiveness)

Executors.newSingleThreadExecutor().execute(() -> {

Staff s = AppDatabase.getInstance(getApplicationContext()).staffDao().findByEmail(email); // Fetch staff by email

// Validate: must exist, be ADMIN role, have a stored PIN, and it must match

if (s == null || s.role != Staff.Role.ADMIN || s.adminPin == null || !s.adminPin.equals(pin)) {

runOnUiThread(() -> Toast.makeText(this, "Invalid admin credentials.", Toast.LENGTH\_SHORT).show()); // Show error on UI thread

} else {

// Persist session details and proceed into the Admin Portal

SessionManager.setCurrentUser(this, "ADMIN", s.email); // Store role/email for later checks

Intent i = new Intent(this, AdminPortalActivity.class); // Navigate to the portal proper

startActivity(i);

finish(); // Close login screen after success

}

});

}

}

**AdminPortalActivity.java**

package com.example.hospimanagmenetapp.ui; // UI layer package for Activities

import androidx.appcompat.app.AppCompatActivity; // Base Activity with AppCompat features

import androidx.recyclerview.widget.LinearLayoutManager; // Lays out RecyclerView items in a vertical list

import androidx.recyclerview.widget.RecyclerView; // Efficient scrolling list/grid container

import android.os.Bundle; // Lifecycle state bundle

import android.text.TextUtils; // Simple string checks (e.g., isEmpty)

import android.widget.ArrayAdapter; // Adapter to back the Spinner with enum values

import android.widget.Button; // UI widget: Button

import android.widget.EditText; // UI widget: text input

import android.widget.Spinner; // UI widget: drop-down selection

import android.widget.Toast; // Lightweight user notifications

import com.example.hospimanagmenetapp.R; // Resource references (layouts, IDs)

import com.example.hospimanagmenetapp.data.AppDatabase; // Room database singleton

import com.example.hospimanagmenetapp.data.dao.StaffDao; // DAO for Staff operations

import com.example.hospimanagmenetapp.data.entities.Staff; // Staff entity (has Role enum, email, PIN)

import com.example.hospimanagmenetapp.ui.adapters.StaffAdapter; // RecyclerView adapter to render staff list

import com.example.hospimanagmenetapp.util.SessionManager; // Simple session storage for RBAC checks

import java.util.Arrays; // Utility to turn arrays into Lists

import java.util.List; // List interface for collections

import java.util.concurrent.Executors; // Run DB work off the main thread

public class AdminPortalActivity extends AppCompatActivity { // Admin portal: manage staff accounts

private EditText etName, etEmail, etPin; // Inputs for staff name/email and admin PIN (if role is ADMIN)

private Spinner spRole; // Role picker (ADMIN/STAFF/etc.)

private Button btnRegisterStaff, btnRefresh; // Actions to register and refresh the list

private RecyclerView rvStaff; // Displays the current staff members

@Override

protected void onCreate(Bundle savedInstanceState) { // Activity creation lifecycle

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_admin\_portal); // Inflate the admin portal layout

// Bind views from XML

etName = findViewById(R.id.etStaffName);

etEmail = findViewById(R.id.etStaffEmail);

etPin = findViewById(R.id.etAdminSetupPin);

spRole = findViewById(R.id.spRole);

btnRegisterStaff = findViewById(R.id.btnRegisterStaff);

btnRefresh = findViewById(R.id.btnRefreshList);

rvStaff = findViewById(R.id.rvStaff);

rvStaff.setLayoutManager(new LinearLayoutManager(this)); // Vertical list for the RecyclerView

// Populate the role Spinner with all Staff.Role enum values using a simple built-in layout

spRole.setAdapter(new ArrayAdapter<>(

this,

android.R.layout.simple\_spinner\_dropdown\_item,

Arrays.asList(Staff.Role.values())

));

// Wire up button actions

btnRegisterStaff.setOnClickListener(v -> registerStaff()); // Validate inputs and insert staff

btnRefresh.setOnClickListener(v -> loadStaff()); // Reload the staff list from DB

// RBAC guard unless explicitly bypassed for first-admin bootstrap

boolean bypass = getIntent().getBooleanExtra("bypassCheck", false); // True if launched from setup flow

if (!bypass) {

String role = SessionManager.getCurrentRole(this); // Read stored role

if (!"ADMIN".equals(role)) { // Only admins may enter

Toast.makeText(this, "Admin access required.", Toast.LENGTH\_SHORT).show();

finish(); // Close and return to previous screen

return;

}

}

loadStaff(); // Populate list on first load

}

// Read inputs, validate, and insert a new Staff record (background thread)

private void registerStaff() {

String name = etName.getText().toString().trim(); // Staff full name

String email = etEmail.getText().toString().trim(); // Staff email (should be unique)

Staff.Role role = (Staff.Role) spRole.getSelectedItem(); // Selected role from Spinner

String pin = etPin.getText().toString().trim(); // Admin PIN (required only for ADMIN)

// Basic required-field checks

if (TextUtils.isEmpty(name) || TextUtils.isEmpty(email)) {

Toast.makeText(this, "Name and email are required.", Toast.LENGTH\_SHORT).show();

return; // Don’t proceed without essentials

}

// Enforce a PIN for ADMIN role

if (role == Staff.Role.ADMIN && TextUtils.isEmpty(pin)) {

Toast.makeText(this, "Admin PIN is required for ADMIN role.", Toast.LENGTH\_SHORT).show();

return;

}

// Do DB I/O off the main thread

Executors.newSingleThreadExecutor().execute(() -> {

try {

StaffDao dao = AppDatabase.getInstance(getApplicationContext()).staffDao(); // DAO handle

Staff s = new Staff(); // Create new entity

s.fullName = name; // Map inputs to fields

s.email = email;

s.role = role;

s.adminPin = (role == Staff.Role.ADMIN) ? pin : null; // Store PIN only for admins

dao.insert(s); // Persist to Room (unique constraints may throw)

// On success, clear form and refresh list on the UI thread

runOnUiThread(() -> {

Toast.makeText(this, "Staff registered.", Toast.LENGTH\_SHORT).show();

etName.setText(""); // Reset inputs

etEmail.setText("");

etPin.setText("");

loadStaff(); // Refresh the RecyclerView with latest data

});

} catch (Exception e) { // Likely a uniqueness violation on email (if enforced)

runOnUiThread(() ->

Toast.makeText(this, "Error: email may already exist.", Toast.LENGTH\_SHORT).show());

}

});

}

// Fetch all staff from the DB and display them in the RecyclerView

private void loadStaff() {

Executors.newSingleThreadExecutor().execute(() -> {

List<Staff> list = AppDatabase.getInstance(getApplicationContext()).staffDao().getAll(); // Read from Room

runOnUiThread(() -> rvStaff.setAdapter(new StaffAdapter(list))); // Bind adapter on UI thread

});

}

}

**StaffAdapter.java**

package com.example.hospimanagmenetapp.ui.adapters; // Adapter classes for RecyclerView live here

import android.view.LayoutInflater; // Inflates XML layouts into View objects

import android.view.View; // Base class for all UI components

import android.view.ViewGroup; // Container that holds child views

import android.widget.TextView; // UI widget for displaying text

import androidx.annotation.NonNull; // Hint for null-safety contracts

import androidx.recyclerview.widget.RecyclerView; // Support library RecyclerView

import com.example.hospimanagmenetapp.R; // Resource IDs (layouts, strings, etc.)

import com.example.hospimanagmenetapp.data.entities.Staff; // Model class rendered by this adapter

import java.util.List; // List interface for holding items

public class StaffAdapter extends RecyclerView.Adapter<StaffAdapter.VH> { // Adapter bridges Staff data to RecyclerView rows

private final List<Staff> data; // Backing data set rendered by the list

public StaffAdapter(List<Staff> data) { // Constructor takes the data to display

this.data = data; // Store the list for later binding

}

@NonNull

@Override

public VH onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

// Inflate a single row (item\_staff.xml) and wrap it in a ViewHolder

View v = LayoutInflater.from(parent.getContext())

.inflate(R.layout.item\_staff, parent, false); // false = don’t attach yet; RecyclerView will handle it

return new VH(v); // Return a new ViewHolder instance bound to this row view

}

@Override

public void onBindViewHolder(@NonNull VH h, int position) {

// Bind the Staff item at 'position' to the views in the holder

Staff s = data.get(position); // Fetch the model for this row

h.tvName.setText(s.fullName); // Show staff member’s name

h.tvEmail.setText(s.email); // Show staff member’s email

h.tvRole.setText(s.role.name()); // Show role (enum name as text)

}

@Override

public int getItemCount() {

// RecyclerView asks how many rows to display

return data == null ? 0 : data.size(); // Null-safe size (0 if data not set)

}

// ViewHolder caches view references for each row to avoid repeated findViewById calls

static class VH extends RecyclerView.ViewHolder {

TextView tvName, tvEmail, tvRole; // Row widgets for name, email, and role

VH(@NonNull View itemView) { // Constructed with the inflated row view

super(itemView); // Pass to base class

tvName = itemView.findViewById(R.id.tvStaffName); // Bind TextView for name

tvEmail = itemView.findViewById(R.id.tvStaffEmail); // Bind TextView for email

tvRole = itemView.findViewById(R.id.tvStaffRole); // Bind TextView for role

}

}

}