CS-499-11684-M01

Professor Fitzroy Nembhard

Fernanda Coghlan

02 June 2025

# CS 499 Module Five

# 5-2Milestone Four:

# Enhancement Three: Databases

**PART I**

**Artifact Narrative**

# Structure

This narrative documents the **third enhancement** for my CS 499 ePortfolio project, focusing on **databases**. The purpose of this enhancement was to demonstrate my ability to design and implement a robust, persistent data management solution for a professional-quality Android application. I applied modern database engineering principles to enhance the **FLC Weight Tracker** app, originally developed during **CS 360: Mobile Architecture and Programming** (Spring 2024).

The original prototype lacked data persistence; all user data was stored in-memory and was lost when the app closed or the device restarted. Through this enhancement, I integrated the **Room database** to persist weight entries and used **SharedPreferences** to manage lightweight user preferences. The application now delivers a consistent, data-driven experience across user sessions, demonstrating key database engineering skills in Android development.

# Artifact Description

The selected artifact is an Android mobile application titled **FLC Weight Tracker**, which enables users to manage their personal weight tracking goals. The core functionality of the app includes:

* Account registration and secure login
* Adding daily weight entries
* Viewing weight trends over time
* Setting and managing weight goals
* Updating profile information
* Managing notification preferences (Yes/No for SMS)

Initially, the application was at a **prototype level**:

* Data was stored entirely in-memory
* No persistence layer was implemented
* Closing the app or restarting the device resulted in total data loss

The goal of this enhancement was to evolve the app into a **persistent, data-driven system** using best practices in Android database architecture.

# Justification for Inclusion

I selected this artifact for the **Databases** category because it presented a clear opportunity for significant improvement in persistent data management. Without a database or preferences layer, the app could not deliver a consistent user experience across sessions.

**Key enhancements implemented include:**

**Room Database Integration**  
To support structured persistence of weight entries:

* Created the WeightEntry entity class
* Implemented WeightEntryDao with insert, delete, and query methods
* Established AppDatabase as a singleton Room instance
* Integrated LiveData to ensure the UI reacts dynamically to database updates

**Persistence of User Preferences via SharedPreferences**  
For lightweight data storage across app sessions:

* Profile information (Name, Password, Email, Phone)
* Weight Goal
* Notification Yes/No preference

**Automatic Data Recovery**  
The application now automatically restores:

* Weight entries
* Profile data
* Weight goals
* Notification preferences

**UI Now Linked to Database**

* Weekly weight trend is calculated directly from Room data
* Weight entries are displayed dynamically using LiveData
* Profile fields and weight goal auto-fill from SharedPreferences
* Notification preference is visually maintained across sessions

**Robust Database Practices Demonstrated**

* Lifecycle-aware data persistence with Room ORM
* Lightweight settings management with SharedPreferences
* Consistent and smooth user experience
* Clear separation of structured vs. simple data storage

This enhancement significantly elevated the architectural quality of the app and aligns strongly with modern **Android persistence design patterns**.

# Course Outcomes and Updates

**Outcome Alignment**

This enhancement directly supports **Course Outcome #4**:

"Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals (software engineering/design/database)."

**Specific Achievements**

* Implemented Room ORM-based database
* Integrated LiveData for a reactive, responsive UI
* Persisted multiple data categories:
  + Weight Entries
  + Profile Information
  + Weight Goals
  + Notification Preferences

**Updates**  
No changes are needed to my **outcome coverage plan** from Module One. This enhancement fully meets my planned objectives. Additionally, this work contributes to **Course Outcome #5** in the area of **secure data handling**, by selecting appropriate mechanisms (Room for structured data and SharedPreferences for lightweight preferences).

# Reflection on Enhancement Process

Throughout this enhancement, I gained valuable experience in applying modern database techniques to Android development.

**What I Learned**

* How to integrate Room into an existing Android app architecture
* How to design effective Room entities and DAOs
* How to use LiveData to create a reactive and seamless UI
* How to persist user preferences securely with SharedPreferences
* How to coordinate multiple persistence layers (Room + SharedPreferences)

**Challenges Faced**

* Designing an optimal WeightEntry data schema to support trend calculations
* Ensuring thread-safe updates when observing LiveData
* Migrating from an in-memory architecture to a fully persistent model
* Maintaining responsive and intuitive UI during database interactions
* Coordinating ProfileActivity data across multiple screens (Login, Dashboard, Weight Tracker)

**Outcomes**  
Following an iterative process of testing and refinement, the FLC Weight Tracker app now offers:

* Fully data-driven architecture (Room + LiveData)
* Persistent profile, goals, and settings
* Appropriate separation of sensitive and lightweight data
* Enhanced user experience through smooth and consistent UI transitions
* A strong example of database engineering for mobile applications

### **Final Summary**

This milestone completes the transformation of **FLC Weight Tracker** from a prototype to a production-quality mobile application with robust, modern data handling capabilities. The app now features:

* Full **Room database integration** for structured weight data
* Correct use of **SharedPreferences** for lightweight preferences
* Reliable **persistence** of key user data
* Polished, reactive, and user-friendly UI

Through this work, I demonstrated my ability to:

* Architect database-driven Android applications
* Manage multiple layers of persistent data effectively
* Implement modern, reactive UI linked to database models
* Deliver a seamless and professional user experience

In conclusion, this enhancement reflects a comprehensive mastery of Android persistence techniques and strongly supports my ePortfolio for **Milestone Four** in the **Databases** category.

# Evidence of Enhancement

# Screenshots / Code Evidence

**WeightEntry.java (Room Entity)**

// CS-499-11684-M01  
// Professor Fitzroy Nembhard  
// Fernanda Coghlan  
// 02 June 2025  
  
package com.example.cs499trackerapp;  
  
import androidx.room.Entity;  
import androidx.room.PrimaryKey;  
import java.util.Date;  
  
*/\*\*  
 \* Entity class representing a row in the 'weight\_entries' table.  
 \* Used by Room to generate database schema.  
 \*/*@Entity(tableName = "weight\_entries")  
public class WeightEntry {  
  
 @PrimaryKey(autoGenerate = true)  
 private int id;  
  
 private float weight; // Weight in pounds  
 private Date date; // Date of entry  
 private String userId; // Username that owns this entry  
  
 // Constructor  
 public WeightEntry(float weight, Date date, String userId) {  
 this.weight = weight;  
 this.date = date;  
 this.userId = userId;  
 }  
  
 // --- Getters ---  
  
 public int getId() {  
 return id;  
 }  
  
 public float getWeight() {  
 return weight;  
 }  
  
 public Date getDate() {  
 return date;  
 }  
  
 public String getUserId() {  
 return userId;  
 }  
  
 // --- Setters ---  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public void setWeight(float weight) {  
 this.weight = weight;  
 }  
  
 public void setDate(Date date) {  
 this.date = date;  
 }  
  
 public void setUserId(String userId) {  
 this.userId = userId;  
 }  
}

**WeightEntryDao.java**

// CS-499-11684-M1  
// Professor Fitzroy Nembhard  
// Fernanda Coghlan  
// 02 June 2025  
  
package com.example.cs499trackerapp;  
  
import androidx.lifecycle.LiveData;  
import androidx.room.Dao;  
import androidx.room.Delete;  
import androidx.room.Insert;  
import androidx.room.Query;  
  
import java.util.List;  
  
// DAO = Data Access Object — Interface defining how we interact with the Room database  
@Dao  
public interface WeightEntryDao {  
  
 // Insert a new weight entry into the weight\_entries table  
 // Room handles this on a background thread when used via repository  
 @Insert  
 void insert(WeightEntry entry);  
  
 // Delete a specific weight entry from the table  
 // Typically used when supporting delete operations in the UI  
 @Delete  
 void delete(WeightEntry entry);  
  
 // Custom query to get all entries for a specific user sorted by date (most recent first)  
 // LiveData ensures automatic updates to observers when the table changes  
 @Query("SELECT \* FROM weight\_entries WHERE userId = :username ORDER BY date DESC")  
 LiveData<List<WeightEntry>> getEntriesForUser(String username);  
}

**AppDatabase.java**

// CS-499-11684-M01  
// Professor Fitzroy Nembhard  
// Fernanda Coghlan  
// 02 June 2025  
  
package com.example.cs499trackerapp;  
  
import android.content.Context;  
  
import androidx.room.Database;  
import androidx.room.Room;  
import androidx.room.RoomDatabase;  
import androidx.room.TypeConverters;  
  
// Defines the database with associated entities and type converters  
@Database(entities = {WeightEntry.class}, version = 1, exportSchema = false)  
@TypeConverters({Converters.class}) // Enables custom converters (e.g., for Date objects)  
public abstract class AppDatabase extends RoomDatabase {  
  
 // Abstract method to access DAO  
 public abstract WeightEntryDao weightEntryDao();  
  
 // Singleton instance to prevent multiple database objects  
 private static volatile AppDatabase *INSTANCE*;  
  
 // Provides access to the singleton database instance  
 public static AppDatabase getDatabase(final Context context) {  
 if (*INSTANCE* == null) {  
 synchronized (AppDatabase.class) {  
 if (*INSTANCE* == null) {  
 // Creates the database instance using Room builder  
 *INSTANCE* = Room.*databaseBuilder*(  
 context.getApplicationContext(),  
 AppDatabase.class,  
 "weight\_database" // Database file name  
 )  
 .fallbackToDestructiveMigration() // Automatically rebuilds DB on schema changes (dev use only)  
 .build();  
 }  
 }  
 }  
 return *INSTANCE*;  
 }  
}

**LiveData observer in WeightTrackerActivity**

// CS-499-11684-M01  
// Professor Fitzroy Nembhard  
// Fernanda Coghlan  
// 02 June 2025  
  
package com.example.cs499trackerapp;  
  
import android.app.AlertDialog;  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.util.Log;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.TableLayout;  
import android.widget.TableRow;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.lifecycle.ViewModelProvider;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.List;  
import java.util.Locale;  
  
*/\*\*  
 \* WeightTrackerActivity  
 \* Allows the user to:  
 \* - Add weight entries  
 \* - View historical weight data  
 \* - Set a weight goal (persisted)  
 \* - View weekly trend analysis  
 \* - Delete entries  
 \* - Navigate back to Dashboard  
 \*/*public class WeightTrackerActivity extends AppCompatActivity {  
  
 private static final String *TAG* = "WeightTrackerActivity";  
  
 // UI components  
 private TableLayout weightTable;  
 private TextView weightGoalValue;  
 private TextView weeklyTrendValue;  
 private WeightViewModel viewModel;  
 private String currentUser;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_weight\_tracker*);  
  
 Log.*d*(*TAG*, "onCreate started");  
  
 // Step 1: Retrieve username from Intent  
 currentUser = getIntent().getStringExtra("username");  
 Log.*d*(*TAG*, "Got from intent: " + currentUser);  
  
 // Step 2: Fallback to SharedPreferences if needed  
 if (currentUser == null || currentUser.isEmpty()) {  
 SharedPreferences prefs = getSharedPreferences("UserPrefs", Context.*MODE\_PRIVATE*);  
 currentUser = prefs.getString("loggedInUser", null);  
 Log.*d*(*TAG*, "Fallback to prefs: " + currentUser);  
 }  
  
 // Step 3: Redirect to login if username missing  
 if (currentUser == null || currentUser.isEmpty()) {  
 Log.*e*(*TAG*, "No username found. Redirecting.");  
 Toast.*makeText*(this, "No user found. Returning to login.", Toast.*LENGTH\_SHORT*).show();  
 startActivity(new Intent(this, LoginActivity.class));  
 finish();  
 return;  
 }  
  
 // Step 4: Initialize UI components  
 weightTable = findViewById(R.id.*weight\_table*);  
 weightGoalValue = findViewById(R.id.*weight\_goal\_value*);  
  
 // Load saved goal  
 SharedPreferences prefs = getSharedPreferences("UserPrefs", Context.*MODE\_PRIVATE*);  
 float savedGoal = prefs.getFloat("weightGoal", 0.0f);  
 weightGoalValue.setText(String.*format*(Locale.*US*, "%.1f lbs", savedGoal));  
  
 weeklyTrendValue = findViewById(R.id.*weekly\_trend\_value*);  
 EditText inputDate = findViewById(R.id.*input\_date*);  
 EditText inputWeight = findViewById(R.id.*input\_weight*);  
 Button addWeightButton = findViewById(R.id.*add\_weight\_button*);  
 Button setGoalButton = findViewById(R.id.*set\_goal\_button*);  
  
 // Step 5: Initialize ViewModel  
 try {  
 WeightViewModelFactory factory = new WeightViewModelFactory(getApplication(), currentUser);  
 viewModel = new ViewModelProvider(this, factory).get(WeightViewModel.class);  
 Log.*d*(*TAG*, "ViewModel created for user: " + currentUser);  
  
 // Bind LiveData observers  
 viewModel.bindTrendToOwner(this);  
 } catch (Exception e) {  
 Log.*e*(*TAG*, "ViewModel creation failed", e);  
 Toast.*makeText*(this, "Unable to load weight data.", Toast.*LENGTH\_LONG*).show();  
 finish();  
 return;  
 }  
  
 // Step 6: Observe LiveData  
 viewModel.getAllEntries().observe(this, this::updateTable);  
 viewModel.getWeeklyTrend().observe(this, trend -> weeklyTrendValue.setText(trend));  
  
 // Step 7: Add weight entry  
 addWeightButton.setOnClickListener(v -> {  
 String dateStr = inputDate.getText().toString().trim();  
 String weightStr = inputWeight.getText().toString().trim();  
  
 if (dateStr.isEmpty() || weightStr.isEmpty()) {  
 Toast.*makeText*(this, "Please fill both fields.", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
  
 try {  
 float weight = Float.*parseFloat*(weightStr);  
 SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy", Locale.*US*);  
 Date date = sdf.parse(dateStr);  
  
 if (date != null) {  
 // Insert entry  
 WeightEntry entry = new WeightEntry(weight, date, currentUser);  
 viewModel.insertEntry(entry);  
  
 // Check goal  
 String goalStr = weightGoalValue.getText().toString().replace(" lbs", "").trim();  
 if (!goalStr.isEmpty()) {  
 try {  
 float goal = Float.*parseFloat*(goalStr);  
 if (Float.*compare*(goal, weight) == 0) {  
 Toast.*makeText*(this, "🎉 You reached your goal!", Toast.*LENGTH\_LONG*).show();  
 }  
 } catch (NumberFormatException ignored) {}  
 }  
  
 // Reset fields  
 inputDate.setText("");  
 inputWeight.setText("");  
 } else {  
 Toast.*makeText*(this, "Invalid date format.", Toast.*LENGTH\_SHORT*).show();  
 }  
  
 } catch (Exception e) {  
 Toast.*makeText*(this, "Error adding weight.", Toast.*LENGTH\_SHORT*).show();  
 Log.*e*(*TAG*, "Weight entry error", e);  
 }  
 });  
  
 // Step 8: Set weight goal  
 setGoalButton.setOnClickListener(v -> {  
 final EditText input = new EditText(this);  
 input.setHint("Enter weight goal (lbs)");  
  
 new AlertDialog.Builder(this)  
 .setTitle("Set Weight Goal")  
 .setView(input)  
 .setPositiveButton("Set", (dialog, which) -> {  
 String goalStr = input.getText().toString().trim();  
 if (!goalStr.isEmpty()) {  
 try {  
 float goal = Float.*parseFloat*(goalStr);  
 weightGoalValue.setText(String.*format*(Locale.*US*, "%.1f lbs", goal));  
  
 // Save to SharedPreferences  
 prefs.edit().putFloat("weightGoal", goal).apply();  
  
 } catch (NumberFormatException e) {  
 Toast.*makeText*(this, "Invalid weight goal entered.", Toast.*LENGTH\_SHORT*).show();  
 }  
 } else {  
 Toast.*makeText*(this, "Weight goal cannot be empty.", Toast.*LENGTH\_SHORT*).show();  
 }  
 })  
 .setNegativeButton("Cancel", null)  
 .show();  
 });  
  
 // Step 9: Back to Dashboard  
 Button backToDashboard = findViewById(R.id.*button\_back\_to\_dashboard*);  
 backToDashboard.setOnClickListener(v -> {  
 Intent intent = new Intent(WeightTrackerActivity.this, DashboardActivity.class);  
 startActivity(intent);  
 finish(); // Close this activity  
 });  
 }  
  
 */\*\*  
 \* Dynamically updates the weight entry table.  
 \*  
 \* @param entries List of weight entries  
 \*/* private void updateTable(List<WeightEntry> entries) {  
 int rowCount = weightTable.getChildCount();  
 if (rowCount > 1) {  
 weightTable.removeViews(1, rowCount - 1);  
 }  
  
 SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy", Locale.*US*);  
  
 for (WeightEntry entry : entries) {  
 TableRow row = new TableRow(this);  
 TextView dateView = new TextView(this);  
 TextView weightView = new TextView(this);  
  
 // Populate row  
 dateView.setText(sdf.format(entry.getDate()));  
 weightView.setText(String.*format*(Locale.*US*, "%.1f", entry.getWeight()));  
  
 // Styling  
 dateView.setPadding(8, 8, 8, 8);  
 weightView.setPadding(8, 8, 8, 8);  
  
 row.addView(dateView);  
 row.addView(weightView);  
  
 // Long press to delete  
 row.setOnLongClickListener(v -> {  
 new AlertDialog.Builder(this)  
 .setTitle("Delete Entry")  
 .setMessage("Do you want to delete this entry?")  
 .setPositiveButton("Delete", (dialog, which) -> {  
 viewModel.deleteEntry(entry);  
 Toast.*makeText*(this, "Entry deleted", Toast.*LENGTH\_SHORT*).show();  
 })  
 .setNegativeButton("Cancel", null)  
 .show();  
 return true;  
 });  
  
 weightTable.addView(row);  
 }  
 }  
}

**Profile screen with saves the data until new update.**

A screenshot of a phone

AI-generated content may be incorrect.

**Weight data and Weight goal save the data until new update. Weekly Weight Trend brought to the top for better user experience. Back to Dashboard button added to Weight Tracker screen. Set Goal instructions added to support final user.**

A screenshot of a phone

AI-generated content may be incorrect.

**Notifications screen holds last choice until new choice is made.**

A screenshot of a phone

AI-generated content may be incorrect.

**PART II**

**Artifact Submission**

Technical artifact files were zipped and uploaded and I used the check list below to assure the enhancement proposals for Database were implemented:

**1. Room Database Integration**

To provide reliable, structured persistence of weight entries, I integrated Room, a modern Android ORM library. I defined a WeightEntry entity class with userId, date, and weight fields to model the core data structure. A WeightEntryDao interface was created to expose standard operations including insert, delete, and querying user-specific entries. The AppDatabase singleton Room instance ensures consistent access to the database across the app lifecycle.

**2. LiveData Observers**

To enable a reactive user interface, the WeightViewModel was extended to expose LiveData of weight entries retrieved from Room. The Weight Tracker screen observes this LiveData and automatically updates the displayed entries whenever database changes occur, without requiring manual refreshes. This implementation adheres to best practices for lifecycle-aware data handling.

**3. Weight Entry Persistence**

Weight entries are now fully persistent. When a user adds a new weight entry, it is inserted into Room through the DAO layer. On subsequent app launches, previously saved entries are retrieved from the database and displayed. All weekly trend calculations are based entirely on persisted Room data, ensuring accuracy and consistency across sessions.

**4. Persistent User Preferences**

In addition to Room for complex data, SharedPreferences is used for lightweight user preferences. I implemented persistence for the user’s profile information (username, password, email, and phone), the current weight goal, and the YES/NO preference for SMS notifications. These settings are saved in SharedPreferences and automatically restored when the user revisits the relevant screens.

**5. Automatic Data Recovery**

One of the critical improvements is automatic data recovery. When the app is relaunched, the following data is restored automatically: all weight entries from Room, the weight goal from SharedPreferences, the user’s profile fields, and the notifications preference. The user’s experience is now seamless, with no data loss even after closing and reopening the app.

**6. UI Binding to Persistent Data**

The user interface is now tightly bound to persisted data. The weekly weight trend is computed in real-time from Room data using LiveData observation. The weight entries table reflects the latest Room state. The weight goal is displayed based on the stored value from SharedPreferences, and profile fields are pre-populated with saved preferences. This ensures a consistent and intuitive experience for the user.

**7. Security and Best Practices**

Throughout the enhancement, I followed best practices for secure and efficient data handling. Sensitive information such as usernames and passwords are stored in private SharedPreferences. Room is used with ViewModel and LiveData to ensure thread-safe, lifecycle-aware database interactions. All data persistence mechanisms comply with Android architecture guidelines.

**8. Enhanced User Experience**

The user experience has been greatly improved with this enhancement. Navigation is consistent, with a “Back to Dashboard” button now available on key screens. The Notifications screen visually reflects the persisted preference, and selections remain consistent across sessions. The Profile screen now reliably persists user updates. The weight goal functionality allows for redefinition and maintains its state across app restarts.

**9. Testing and Validation**

Comprehensive testing was conducted to validate the enhancements. Weight entries persist correctly and display as expected. Weekly trend calculations update automatically in response to database changes. Profile data remains consistent across sessions. The Notifications preference accurately reflects user selections even after closing and reopening the app. All in-memory-only data handling has been eliminated in favor of persistent storage.