Patrick Quinn  
CS499

2/9/2025

Database Narrative

1. Data Structure (WeightStats and WeeklyChange):

kotlin

Copy

data class WeightStats(

val startWeight: Double,

val currentWeight: Double,

*// ...other fields*

)

These data classes define the structure for storing weight statistics. WeightStats holds comprehensive information about a user's weight journey, while WeeklyChange tracks weekly averages.

1. Repository Pattern (WeightRepository):

kotlin

Copy

class WeightRepository(private val dbHelper: DatabaseHelper) {

fun getAllWeightsForUser(userId: Long): Flow<List<WeightEntry>> = flow {

emit(dbHelper.getAllWeightsForUser(userId))

}.flowOn(Dispatchers.IO)

*// ...other methods*

}

The repository acts as a single source of truth for data operations. It wraps database operations and ensures they run on background threads using Dispatchers.IO. The use of Flow enables reactive data streaming, meaning the UI automatically updates when data changes.

1. ViewModel Implementation:

kotlin

Copy

class WeightViewModel(private val repository: WeightRepository) : ViewModel() {

private val \_weights = MutableStateFlow<List<WeightEntry>>(emptyList())

val weights: StateFlow<List<WeightEntry>> = \_weights.asStateFlow()

*// ...other properties and methods*

}

The ViewModel manages UI state and business logic. It uses StateFlow for reactive state management, meaning any changes to the data automatically trigger UI updates. Key features include:

* Calculating statistics from weight data
* Managing data loading and updates
* Processing weekly weight changes
* Handling all database operations through the repository

1. Custom Chart View:

kotlin

Copy

class WeightChartView @JvmOverloads constructor(

context: Context,

attrs: AttributeSet? = null,

defStyleAttr: Int = 0

) : View(context, attrs, defStyleAttr) {

*// ...implementation*

}

This custom view creates a line graph visualization of weight data. It:

* Calculates point positions based on weight values
* Draws axes and graph lines using the Android Canvas API
* Automatically scales the graph based on min/max weights
* Updates the display when new data arrives

1. Main Activity:

kotlin

Copy

class activity\_weight\_database : AppCompatActivity() {

private lateinit var viewModel: WeightViewModel

*// ...other properties*

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_weight\_database)

*// ...initialization*

}

}

The activity coordinates the UI and user interactions. Key features include:

* Setting up the ViewModel and UI components
* Observing data changes using coroutines and Flow
* Handling user input for adding/editing weights
* Managing the date picker dialog
* Showing confirmation dialogs for deletions

Notable Implementation Details:

1. Coroutines Usage:

kotlin

Copy

lifecycleScope.launch {

viewModel.weights.collect { weights ->

adapter.updateData(weights)

findViewById<WeightChartView>(R.id.weightChart).setData(weights)

}

}

The code uses Kotlin coroutines for asynchronous operations, ensuring smooth UI performance by:

* Running database operations on background threads
* Using structured concurrency with lifecycleScope
* Collecting Flow updates safely within the activity's lifecycle

1. State Management:

kotlin

Copy

private val \_weights = MutableStateFlow<List<WeightEntry>>(emptyList())

val weights: StateFlow<List<WeightEntry>> = \_weights.asStateFlow()

The application uses StateFlow for state management, providing:

* Thread-safe state updates
* Reactive UI updates
* Proper encapsulation of mutable state

1. Error Handling:

kotlin

Copy

try {

val weight = weightStr.toDouble()

*// ...processing*

} catch (e: NumberFormatException) {

weightInput.error = "Invalid weight value"

}

The code includes comprehensive error handling for:

* Invalid input validation
* Database operation failures
* User ID validation
* Number format exceptions

1. UI Updates:

kotlin

Copy

private fun updateStatsUI(stats: WeightStats?) {

stats?.let {

findViewById<TextView>(R.id.statsContent).text = """

Start Weight: ${it.startWeight}

*// ...other stats*

""".trimIndent()

}

}

The UI updates are handled efficiently with:

* Null-safe operations using the safe call operator
* Formatted string templates for readable output
* Reactive updates through Flow collection