**1. TRANSACTIONS**

**1.1 TransactionService (services/transaction\_service\_impl.dart)**

**State & Helpers**

* **static const \_kLastPulledKey**  
  key in SharedPreferences for downstream sync timestamp.
* **final Box<TransactionModel> \_txnBox**  
  Hive box holding all local transactions.
* **final Box<SyncRecord> \_queueBox**  
  Hive box holding pending upstream writes.
* **final FirebaseFirestore \_firestore**  
  Firestore instance.
* **final String \_userId**  
  Current Firebase-Auth UID.
* **final SharedPreferences \_prefs**  
  For persisting lastPulledAt.

**CRUD + Queueing**

1. **Future<void> addTransaction(TransactionModel txn)**
   * Writes txn immediately to \_txnBox.
   * Enqueues SyncRecord.create(txn) into \_queueBox.
2. **Future<void> updateTransaction(TransactionModel txn)**
   * Same as add, but SyncRecord.update.
3. **Future<void> deleteTransaction(String id)**
   * Deletes locally and enqueues SyncRecord.delete(id).

**Local Reads (Hive-only)**

1. **Future<List<TransactionModel>> fetchAll()**
   * Returns all box values sorted by date desc.
2. **Stream<List<TransactionModel>> watchAll()**
   * Emits current snapshot + on every box change (for real-time UI).
3. **Future<List<TransactionModel>> searchByDescription(String query)**
   * Filters fetchAll() by description substring.
4. **Future<List<String>> recentDescriptions({int limit})**
   * Scans fetchAll() in date order, dedups descriptions, up to limit.
5. **Future<List<TransactionModel>> fetchByDateRange({start, end})**
   * Filters fetchAll() to the given date window.
6. **Future<Summary> getSummary({start, end})**
   * Totals expense vs income over optional date range.
7. **Future<Map<String,double>> getCategoryTotals({start, end})**
   * Sums amounts per categoryId over date range.

**Sync with Firestore**

1. **Future<void> syncUpstream()**
   * Loops queued SyncRecords: for each  
     • create/update → ref.set(..., merge: true)  
     • delete → ref.delete()
   * On success removes the record.
2. **Future<void> syncDownstream()**
   * Reads lastPulledAt from prefs.
   * Queries Firestore for docs with updatedAt > lastPulledAt, orders by updatedAt.
   * Merges each into \_txnBox, then updates lastPulledAt.
3. **Future<void> synchronize()**
   * Calls syncUpstream() then syncDownstream().
4. **Future<void> initialSeed()**
   * If \_txnBox empty: pages through **all** Firestore docs (100/page), writes them locally, sets lastPulledAt.

**Utility**

1. **Future<void> clearAll()**
   * Clears \_txnBox, \_queueBox, and lastPulledAt in prefs.

**1.2 TransactionProvider (state/transaction\_provider.dart)**

**Variables**

* **List<TransactionModel> \_transactions**  
  current in-memory list, updated via watchAll().
* **bool \_isLoading**  
  true during \_initialize() (seed + initial sync).
* **bool \_isProcessing**  
  true while any single add/update/delete/clear operation runs.
* **bool \_isSyncing**  
  true when manually calling synchronize().
* **String? \_error**  
  holds last exception message (for UI error displays).
* **StreamSubscription<List<TransactionModel>> \_sub**  
  subscription to service.watchAll().

**Lifecycle**

1. **\_initialize()** (called from ctor)
   * \_setLoading(true)
   * await service.initialSeed()
   * \_sub = service.watchAll().listen(...)
   * await synchronize()
   * \_setLoading(false)
2. **dispose()**
   * Cancels \_sub.

**Public Methods**

1. **Future<void> addTransaction(txn)**
   * \_runProcessing(() async { service.addTransaction(txn); service.syncUpstream(); })
2. **Future<void> updateTransaction(txn)**
   * Same pattern with \_runProcessing.
3. **Future<void> deleteTransaction(id)**
   * Ditto.
4. **Future<void> synchronize()**
   * \_setSyncing(true) → service.syncUpstream(), then service.syncDownstream() → \_setSyncing(false).
5. **Future<List<TransactionModel>> search(query)**  
   → service.searchByDescription(query) with try/catch.
6. **Future<List<String>> recentDescriptions(limit)**  
   → service.recentDescriptions(limit) with error handling.
7. **Future<List<TransactionModel>> fetchByDateRange(start, end)**  
   → service.fetchByDateRange.
8. **Future<Summary> getSummary({start,end})**  
   → service.getSummary with fallback Summary(0,0).
9. **Future<Map<String,double>> getCategoryTotals({start,end})**  
   → service.getCategoryTotals.
10. **Future<void> clearAll()**
    * \_runProcessing(() { service.clearAll(); \_transactions=[]; }).

**Internals**

* **\_runProcessing(Future fn())**  
  toggles \_isProcessing, clears \_error, runs fn, catches & sets \_error, clears \_isProcessing.
* **\_setLoading(bool), \_setSyncing(bool) & \_setError(Object)**  
  simple setters + notifyListeners.

**2. CATEGORIES**

**2.1 CategoryService (services/category\_service\_impl.dart)**

**State**

* **static const \_kLastPulledKey**
* **Box<CategoryModel> \_box**
* **FirebaseFirestore \_firestore**
* **String \_userId**
* **SharedPreferences \_prefs**

**Methods**

1. **Future<void> seedDefaults()**
   * If Hive empty, writes a small hard-coded list of 5 stock categories.
2. **Future<void> initialSeed()**
   * If Hive empty, pulls all Firestore docs (.orderBy('name')) into Hive and sets lastPulledAt.
3. **Future<List<CategoryModel>> fetchAll()**
   * Local one-off read, sorted by name.
4. **Stream<List<CategoryModel>> watchAll()**
   * Hive-box stream + initial snapshot (via startWith).
5. **Future<CategoryModel?> getById(id)**
6. **Future<List<CategoryModel>> fetchExpenses()**
   * filters fetchAll() by isExpense == true.
7. **Future<List<CategoryModel>> fetchIncome()**
   * isExpense == false.
8. **Future<List<CategoryModel>> searchByName(query)**
   * substring filter on name.
9. **Future<void> addCategory(cat)**
   * Hive put + Firestore set(merge: true).
10. **Future<void> updateCategory(cat)**

* same as add.

1. **Future<void> deleteCategory(id)**

* Hive delete + Firestore delete().

1. **Future<void> syncDownstream()**

* Queries Firestore for updatedAt > lastPulledAt, merges into Hive, updates prefs.

1. **Future<void> synchronize()**

* alias for syncDownstream.

1. **Future<void> clearAll()**

* Hive clear() + prefs remove lastPulledAt.

**2.2 CategoryProvider (state/category\_provider.dart)**

**Variables**

* **List<CategoryModel> \_categories**
* **bool \_loading**
* **bool \_processing**
* **bool \_syncing**
* **String? \_error**
* **StreamSubscription<List<CategoryModel>> \_sub**

**Lifecycle**

* **\_initialize()**  
  • \_setLoading(true)  
  • await service.seedDefaults()  
  • await service.initialSeed()  
  • \_sub = service.watchAll().listen(...)  
  • \_setLoading(false)
* **dispose()** → \_sub.cancel()

**Public Methods**

1. **addCategory(cat)**
2. **updateCategory(cat)**
3. **deleteCategory(id)**
   * All three wrap in \_runProcessing.
4. **Future<CategoryModel?> getById(id)**
5. **Future<List<CategoryModel>> fetchExpenses()**
6. **Future<List<CategoryModel>> fetchIncome()**
7. **Future<List<CategoryModel>> searchByName(query)**
8. **Future<void> syncDownstream()**  
   toggles \_isSyncing.
9. **Future<void> synchronize()** → alias.
10. **Future<void> clearAll()**  
    resets \_categories.

**Internals**

* \_runProcessing(), \_setLoading(), \_setSyncing(), \_setError() same pattern as Transactions.

**3. TEMPLATES**

**3.1 TemplateService (services/template\_service\_impl.dart)**

**State**

* **Box<TemplateModel> \_box**
* **FirebaseFirestore \_firestore**
* **String \_userId**
* **SharedPreferences \_prefs**
* **\_kLastPulledKey**

**Methods**

1. **Future<void> initialSeed()**
   * Hive empty → pull all remote docs sorted by nextRun, store locally, set lastPulledAt.
2. **Future<List<TemplateModel>> fetchAll()**
   * returns sorted by nextRun.
3. **Stream<List<TemplateModel>> watchAll()**
4. **Future<TemplateModel?> getById(id)**
5. **addTemplate(tpl)**, **updateTemplate(tpl)**, **deleteTemplate(id)**
   * Hive + Firestore set/delete (merge: true).
6. **Future<List<TemplateModel>> fetchDueTemplates({now})**
   * local filter: tpl.autoAdd && tpl.nextRun <= now.
7. **Future<void> bumpNextRun(tpl)**
   * calculates nextRun based on frequency: daily/weekly/monthly/custom → persists locally & remote.
8. **Future<void> syncDownstream()**
   * Firestore diffs by updatedAt, merge into Hive, update prefs.
9. **Future<void> synchronize()** → alias.
10. **Future<void> clearAll()**

**Helper**

* **\_toFirestoreMap(tpl)**  
  serializes fields + serverTimestamp for updatedAt.

**3.2 TemplateProvider (state/template\_provider.dart)**

**Variables**

* **List<TemplateModel> \_templates**
* **bool \_isLoading**, **\_isProcessing**, **\_isSyncing**
* **String? \_error**
* **StreamSubscription<List<TemplateModel>> \_sub**

**Lifecycle**

* **\_initialize()**
  + \_setLoading(true)
  + await service.initialSeed()
  + \_sub = service.watchAll().listen(...)
  + \_setLoading(false)
* **dispose()** → \_sub.cancel()

**Public Methods**

1. **addTemplate(tpl)**, **updateTemplate(tpl)**, **deleteTemplate(id)**  
   via \_runProcessing.
2. **Future<List<TemplateModel>> fetchDueTemplates({now})**
3. **Future<void> bumpNextRun(tpl)**
4. **Future<void> syncDownstream()**  
   toggles \_isSyncing.
5. **Future<void> synchronize()**
6. **Future<void> clearAll()**

**Internals**

* \_runProcessing(), \_setLoading(), \_setSyncing(), \_setError() as before.