AI Personal Assistant App — Implementation Guide

1. Vision

A **chat-first personal assistant** mobile app that starts completely empty. The user talks to an AI, and the AI dynamically creates **modules** (reminders, budgets, habits, etc.), defines their data structure, builds custom screen layouts, sets up automations — all through natural language. The user can also manually tweak anything the AI creates.

Core principle: The app has no pre-built features. The AI and the user *together* build the app's functionality through conversation and manual editing.

```
Fresh install → Empty state → "Hey, help me set up reminders"
                            → AI creates a reminders MODULE
                            → Module includes: schema, screens, automations
                            → User tweaks it manually if needed
                            → Now the app "has" reminders
```

---

2. Tech Stack

Layer	Technology	Why
Mobile	Flutter (Dart)	Native feel, fast, single codebase (iOS + Android)
Database	Firebase Firestore	NoSQL, dynamic schemas, built-in offline sync
Auth	Firebase Auth	Simple, supports Google/Apple/email
AI Brain	Cloud Functions → Claude API	Tool use for modules, entries, screens, reminders
Notifications	Firebase Cloud Messaging (FCM)	Native push notifications
Scheduling	Cloud Scheduler → Cloud Functions	Cron jobs for recurring reminders

Screen System	JSON Blueprints + Widget Renderers	Safe, controlled, consistent dynamic UI
Escape Hatch	WebView (if ever needed)	For rare edge cases blueprints can't cover

Why Firebase over Supabase

- Firestore is **NoSQL by nature** — no JSONB workarounds needed. The AI writes whatever schema it wants directly as documents.
- **Offline-first out of the box** — Firestore's Flutter SDK has built-in offline persistence. Write on the subway with no signal, syncs when back online.
- **Everything under one roof** — Firestore, FCM, Auth, Cloud Functions, Cloud Scheduler. No stitching services together.

Why Flutter over React Native / Web

- We evaluated React + Capacitor (web app wrapped as native) for the ability to have AI generate real executable code at runtime.
- **Decision: Flutter + JSON Blueprints** is the right approach because:
  - AI-generated code is a security risk (`new Function()` can do anything)
  - Debugging AI-written broken components is a nightmare for users
  - Every screen looks inconsistent when AI styles things differently each time
  - 15–20 well-built widget types can compose thousands of different screens
  - Every tap, button, and action is **fully controlled by your code**

Why NOT offline-first then wire Firebase later

- Firestore's offline mode is **built into the SDK by default**. Every write already works offline and syncs automatically.
- Building a separate local DB (Hive, Isar, SQLite) first means designing a local model, building CRUD around it, then ripping it out or building a sync layer — weeks of throwaway work.
- **Go straight to Firebase from day one.** You get offline for free.

---

3. Flutter Dependencies

```
dependencies:
  firebase_core: ^3.0.0
  cloud_firestore: ^5.0.0          # DB + offline sync
  firebase_auth: ^5.0.0            # Auth
  firebase_messaging: ^15.0.0      # Push notifications
  cloud_functions: ^5.0.0          # Call Cloud Functions (Claude)
  flutter_local_notifications: ^18.0.0
  dash_chat_2: ^0.0.21             # Chat UI (or custom)
  flutter_riverpod: ^2.0.0         # State management
  go_router: ^14.0.0               # Navigation
  intl: ^0.19.0                    # Date formatting
```

---

4. Architecture Overview

```
┌───────────────────────────────────┐
│            Flutter App             │
│  ┌──────────┐  ┌───────────────┐  │
│  │ Chat UI  │  │ Module Views  │  │
│  └────┬─────┘  └──────┬────────┘  │
│       │               │            │
│       ▼               ▼            │
│      cloud_firestore SDK           │
└────────────┬───────────────────────┘
             │
             ▼
┌──────────────────────────────────────┐
│             Firebase                   │
│  ┌─────────────┐ ┌───────────────┐   │
│  │  Firestore  │ │Cloud Functions│   │
│  │  (NoSQL)    │ │ (Claude AI)   │   │
│  └─────────────┘ └───────┬───────┘   │
│  ┌─────────────┐         │           │
│  │Cloud Sched. │         ▼           │
│  └──────┬──────┘     Claude API      │
│         │           (tool use)       │
│         ▼                            │
│  Cloud Function → FCM push          │
└──────────────────────────────────────┘
```

---

5. Core Concept: Modules

A **module** is the top-level organizational unit. It bundles everything together: the data structure (schema), the UI (screens), the behavior (automations), and the actual data (entries).

```
MODULE = Schema + Screens + Automations + Entries
```

The app knows what capabilities to activate based on the module's **type**.

---

6. Firestore Data Model

Entries live as a **subcollection inside the module** — not in a separate top-level collection.

```
users/{userId}/

  modules/{moduleId}                          ← the module definition
  ├── name: "My Budget"
  ├── type: "budget"                          ← determines behavior
  ├── icon: "wallet"
  ├── color: "#4CAF50"
  ├── created_by: "ai" | "manual"
  ├── created_at: Timestamp
  ├── schema: {                               ← data structure
  │     fields: [
  │       { id: "description", label: "Description", type: "text", required: true
  │       { id: "amount", label: "Amount", type: "currency", required: true },
  │       { id: "category", label: "Category", type: "enum_single",
  │         options: ["food","transport","housing","entertainment","other"] },
  │       { id: "type", label: "Type", type: "enum_single",
  │         options: ["income","expense"] },
  │       { id: "date", label: "Date", type: "date", default: "today" },
  │       { id: "budgeting_rule", label: "50/30/20", type: "enum_single",
  │         options: ["needs","wants","savings"] }
  │     ]
  │   }
  ├── screens: [                              ← UI layouts (JSON blueprints)
  │     {
  │       title: "Dashboard",
  │       layout: [
  │         { type: "summary_cards", cards: [...] },
  │         { type: "pie_chart", group_by: "category", field: "amount" },
  │         { type: "entry_list", sort_by: "date", sort_order: "desc" },
  │         { type: "fab", action: "addEntry" }
  │       ]
  │     },
  │     {
  │       title: "Transactions",
  │       layout: [
  │         { type: "filter_bar", filters: ["category","type","date_range"] },
  │         { type: "entry_list", sort_by: "date", display_fields: [...] },
  │         { type: "fab", action: "addEntry" }
  │       ]
  │     }
  │   ]
  ├── capabilities: ["aggregations","charts","budgetRules","export"]
  ├── automations: [                        ← what happens automatically
  │     {
  │       trigger: "field:due_date",
  │       action: "send_notification",
  │       config: { message_template: "Reminder: {{title}}" }
  │     }
  │   ]
  ├── allowed_actions: ["addEntry","editEntry","deleteEntry","applyFilter","expor
  └── settings: {                           ← module-specific config
        default_view: "list",
        notification_sound: "default"
      }

  modules/{moduleId}/entries/{entryId}      ← the actual data (subcollection)
  ├── description: "Groceries"
  ├── amount: 45.00
  ├── category: "food"
  ├── type: "expense"
  ├── date: "2026-02-14"
  ├── budgeting_rule: "needs"
  ├── created_at: Timestamp
  └── created_by: "ai" | "manual"

  chat/{messageId}                          ← conversation history
  ├── role: "user" | "assistant"
  ├── content: "Create a reminders system for me"
  ├── tool_calls: [...]
  ├── timestamp: Timestamp
  └── context: { module_id: "xxx" }

  devices/{deviceId}                        ← FCM tokens
  ├── fcm_token: "abc123..."
  ├── platform: "ios" | "android"
  └── updated_at: Timestamp
```

Full Example with Multiple Modules

```
users/{userId}/

  modules/
  │
  ├── {budgetModuleId}/
  │    ├── name: "My Budget"
  │    ├── type: "budget"
  │    ├── schema: { fields: [...] }
  │    ├── screens: [...]
  │    └── entries/
  │         ├── {id1}: { description: "Groceries", amount: 45, category: "food", ..
  │         ├── {id2}: { description: "Rent", amount: 1200, category: "housing", ..
  │         └── {id3}: { description: "Salary", amount: 5000, type: "income", ... }
  │
  ├── {remindersModuleId}/
  │    ├── name: "My Reminders"
  │    ├── type: "reminders"
  │    ├── schema: { fields: [...] }
  │    ├── screens: [...]
  │    ├── automations: [...]
  │    └── entries/
  │         ├── {id1}: { title: "Clean room", recurrence: "weekly", ... }
  │         └── {id2}: { title: "Dentist", due_date: "2026-03-15", ... }
  │
  └── {habitsModuleId}/
       ├── name: "My Habits"
       ├── type: "habits"
       ├── schema: { fields: [...] }
       ├── screens: [...]
       └── entries/
            ├── {id1}: { habit: "Meditate", streak: 12, ... }
            └── {id2}: { habit: "Read", streak: 5, ... }
```

Why Entries Live Inside the Module

```
✅ Nested (clean):
   users/{userId}/modules/{moduleId}                ← definition
   users/{userId}/modules/{moduleId}/entries/        ← data right here

❌ Separate (messy):
   users/{userId}/modules/{moduleId}                ← definition here
   users/{userId}/data/{moduleId}/entries/           ← data over there
```

With Firestore subcollections:

- Deleting a module can cascade to its entries
- Security rules are simpler — one path to protect
- Queries are scoped automatically
- It's how Firestore is designed to be used

---

7. Module Type System

The module **type** tells the app what behavior to activate. The AI picks the type, and the app handles the rest.

Available Types

```
enum ModuleType {
  reminders,    // → scheduling, notifications, recurrence, completion
  budget,       // → aggregations, charts, budget rules, export
  habits,       // → daily tracking, streaks, progress
  journal,      // → date-based entries, rich text, mood
  inventory,    // → quantities, low-stock alerts
  contacts,     // → phone/email actions, search
  custom,       // → generic CRUD, no special behavior
}
```

Type → Capability Mapping

```
class FeatureTypeConfig {
  static Map<ModuleType, FeatureConfig> registry = {

    ModuleType.reminders: FeatureConfig(
      capabilities: [
        Capability.notifications,
        Capability.scheduling,
        Capability.recurrence,
        Capability.completion,
      ],
      defaultActions: [
        ScreenAction.addEntry,
        ScreenAction.editEntry,
        ScreenAction.deleteEntry,
        ScreenAction.markComplete,
        ScreenAction.snoozeReminder,
      ],
      systemFields: [
        FieldModel(id: 'completed', label: 'Done', type: FieldType.boolean),
        FieldModel(id: 'completed_at', label: 'Completed At', type: FieldType.dat
        FieldModel(id: 'snoozed_until', label: 'Snoozed Until', type: FieldType.d
      ],
      services: [
        ReminderSchedulingService,
        NotificationService,
        RecurrenceService,
      ],
    ),

    ModuleType.budget: FeatureConfig(
      capabilities: [
        Capability.aggregations,
        Capability.charts,
        Capability.budgetRules,
        Capability.export,
      ],
      defaultActions: [
        ScreenAction.addEntry,
        ScreenAction.editEntry,
        ScreenAction.deleteEntry,
        ScreenAction.applyFilter,
        ScreenAction.exportData,
      ],
      systemFields: [],
      services: [
        AggregationService,
        BudgetRuleService,
      ],
    ),

    ModuleType.habits: FeatureConfig(
      capabilities: [
        Capability.streaks,
        Capability.dailyTracking,
        Capability.progress,
      ],
      defaultActions: [
        ScreenAction.addEntry,
        ScreenAction.markComplete,
      ],
      systemFields: [
        FieldModel(id: 'streak', label: 'Streak', type: FieldType.number),
        FieldModel(id: 'last_completed', label: 'Last Done', type: FieldType.date
      ],
      services: [
        StreakService,
        DailyTrackingService,
      ],
    ),

    ModuleType.custom: FeatureConfig(
      capabilities: [Capability.basicCrud],
      defaultActions: [
        ScreenAction.addEntry,
        ScreenAction.editEntry,
        ScreenAction.deleteEntry,
      ],
      systemFields: [],
      services: [],
    ),
  };
}
```

Module Manager (activates services per type)

```
class ModuleManager {
  final ModuleModel module;
  late final FeatureConfig config;
  late final List<FeatureService> services;

  ModuleManager(this.module) {
    config = FeatureTypeConfig.registry[module.type]
        ?? FeatureTypeConfig.registry[ModuleType.custom]!;
    services = config.services.map((s) => s.init(module)).toList();
  }

  bool can(Capability capability) {
    return config.capabilities.contains(capability);
  }

  Future<void> onEntryCreated(EntryModel entry) async {
    for (final service in services) {
      await service.onEntryCreated(module, entry);
    }
  }
  Future<void> onEntryUpdated(EntryModel entry, Map<String, dynamic> changes) asy
    for (final service in services) {
      await service.onEntryUpdated(module, entry, changes);
    }
  }
}
```

What the AI Sees

```
MODULE TYPES AND THEIR BEHAVIORS:

- "reminders"  → scheduling, notifications, recurrence, completion tracking
- "budget"     → aggregations, charts, 50/30/20 rules, export
- "habits"     → daily tracking, streaks, progress visualization
- "journal"    → date-based entries, rich text, mood tracking
- "inventory"  → quantities, low-stock alerts
- "contacts"   → phone/email actions, search
- "custom"     → basic CRUD, no special behavior

When the user asks for something, pick the right type.
If nothing fits, use "custom".
The app will automatically activate the right services
based on the type you choose.
```

---

8. Field Type System

This is the contract between the AI and the UI. Both the AI and the manual schema editor use the same field types.

```
enum FieldType {
  text,         // simple string
  number,       // int or double
  boolean,      // toggle / checkbox
  datetime,     // date + time picker
  date,         // date only
  time,         // time only
  enumSingle,   // dropdown (single select)
  enumMulti,    // multi-select chips
  currency,     // number + currency symbol
  url,          // URL link
  image,        // file upload reference
  reference,    // link to another module's entry
}
```

Field Definition Model

```
class FieldModel {
  final String id;
  final String label;
  final FieldType type;
  final bool required;
  final dynamic defaultValue;
  final List<String>? options;           // for enum types
  final Map<String, dynamic>? visibleWhen; // conditional visibility
  final String? referenceModuleId;       // for reference type
}
```

---

9. Server-Driven UI: Screen Blueprints

The AI doesn’t generate Flutter code. It generates a **JSON screen definition** stored inside the module, and the app has renderers that turn any blueprint into a real screen.

Available Widget Types (build 15–20)

```
Layout:
  - summary_cards       → stat boxes at the top
  - section             → titled group of widgets
  - tabs                → tabbed sections
  - grid                → 2-3 column grid

Data Display:
  - entry_list          → sortable, filterable list
  - entry_table         → spreadsheet-like view
  - calendar_view       → entries on a calendar
  - kanban              → columns with draggable cards
  - timeline            → chronological feed

Charts:
  - pie_chart
  - bar_chart
  - line_chart
  - progress_bar
  - stat_comparison

Input:
  - quick_add_bar       → fast entry from the screen
  - filter_bar
  - search

Actions:
  - fab                 → floating action button
  - action_buttons      → edit, delete, share
```

Example: Budget Dashboard Blueprint

```
{
  "title": "Dashboard",
  "layout": [
    {
      "type": "summary_cards",
      "cards": [
        {
          "label": "Spent",
          "aggregate": "sum",
          "field": "amount",
          "filter": { "type": "expense" },
          "format": "currency",
          "color": "red"
        },
        {
          "label": "Income",
          "aggregate": "sum",
          "field": "amount",
          "filter": { "type": "income" },
          "format": "currency",
          "color": "green"
        },
        {
          "label": "Balance",
          "aggregate": "difference",
          "fields": ["income", "spent"],
          "format": "currency"
        }
      ]
    },
    {
      "type": "tabs",
      "tabs": [
        {
          "label": "Overview",
          "children": [
            { "type": "pie_chart", "group_by": "category", "field": "amount" },
            { "type": "bar_chart", "group_by": "budgeting_rule", "field": "amount
          ]
        },
        {
          "label": "Transactions",
          "children": [
            { "type": "filter_bar", "filters": ["category", "type", "date_range"]
            {
              "type": "entry_list",
              "sort_by": "date",
              "sort_order": "desc",
              "display_fields": ["description", "amount", "category"],
              "swipe_actions": [
                { "direction": "left", "action": "deleteEntry", "color": "red", "
                { "direction": "right", "action": "markComplete", "color": "green
              ]
            }
          ]
        },
        {
          "label": "Calendar",
          "children": [
            { "type": "calendar_view", "date_field": "date", "label_field": "desc
          ]
        }
      ]
    },
    { "type": "fab", "action": "addEntry" }
  ]
}
```

---

10. Action System (Security Layer)

The AI can only pick from a predefined menu of actions. It can never do something unexpected.

Action Registry

```
enum ScreenAction {
  addEntry,
  editEntry,
  deleteEntry,
  navigateToModule,
  navigateToScreen,
  openChat,
  applyFilter,
  sortBy,
  toggleView,
  shareEntry,
  exportData,
  markComplete,
  snoozeReminder,
}
```

How It Works

The AI outputs:

```
{ "type": "fab", "action": "addEntry" }
```

Your Flutter code handles it:

```
class ActionHandler {
  final BuildContext context;
  final String moduleId;

  void handle(ActionDefinition action, EntryModel? entry) {
    switch (action.type) {
      case ScreenAction.addEntry:
        context.push('/modules/$moduleId/entries/new');

      case ScreenAction.editEntry:
        context.push('/modules/$moduleId/entries/${entry!.id}');

      case ScreenAction.deleteEntry:
        showDialog(
          context: context,
          builder: (_) => AlertDialog(
            title: Text('Delete this entry?'),
            actions: [
              TextButton(
                onPressed: () => Navigator.pop(context),
                child: Text('Cancel'),
              ),
              TextButton(
                onPressed: () {
                  FirebaseFirestore.instance
```

```
                    .doc('users/$userId/modules/$moduleId/entries/${entry!.id}')
                    .delete();
                  Navigator.pop(context);
                },
                child: Text('Delete'),
              ),
            ],
          ),
        );

      case ScreenAction.markComplete:
        FirebaseFirestore.instance
          .doc('users/$userId/modules/$moduleId/entries/${entry!.id}')
          .update({'completed': true, 'completed_at': Timestamp.now()});

      case ScreenAction.exportData:
        exportToCsv(moduleId);
    }
  }
}
```

Per-Module Action Restrictions

```
{
  "name": "Budget",
  "allowed_actions": ["addEntry", "editEntry", "deleteEntry", "applyFilter", "exp
}
```

If the AI tries to put a `shareEntry` action on a module that doesn't allow it, the renderer ignores it.

Security Comparison

Scenario	AI Writes Code	JSON Blueprints
AI can delete user data	Yes	No — your handler decides
AI can make network calls	Yes	Impossible
AI can break the app	Yes	Unknown actions ignored
AI can create any layout	Unlimited	As many widgets as built
You control every tap	No	Yes, completely

11. App Navigation & UI Screens

Bottom Navigation

```
┌─────────────────────────────────────────────┐
│              Bottom Navigation               │
├───────────┬───────────┬───────────┬──────────┤
│   Home    │   Chat    │  Modules  │ Settings │
└───────────┴───────────┴───────────┴──────────┘
```

Screen Map

```
HOME (dashboard)
├── Module cards grid (quick access with summaries)
├── Recent entries across all modules
├── Upcoming reminders
└── Tap a card → Module View

CHAT
├── AI conversation
├── Tool result cards inline ("✅ Created Budget module")
└── Quick actions ("Create new module", "Add entry to...")

MODULES (list/management)
├── All modules list with metadata
├── Tap module → Module View
├── Long press → Edit / Delete
├── + button → Create Module (manual)
│
├── MODULE VIEW (inside a module)
│   ├── Tab: Screen 1 (e.g. "Dashboard")     ← rendered from blueprint
│   ├── Tab: Screen 2 (e.g. "Transactions")  ← rendered from blueprint
│   ├── FAB → Add Entry
│   └── ⚙️ → Module Settings
│       ├── Edit Module Info (name, icon, color)
│       ├── Edit Schema (add/remove/edit fields)
│       ├── Edit Screens (add/remove/reorder widgets)
│       ├── Edit Automations
│       └── Delete Module
│

SETTINGS
├── Account
├── Notifications
├── Theme
└── Export / Import
```

Screen 1: Home Screen

```
┌─────────────────────────────────┐
│  Good morning!                   │
│                                  │
│  ┌────────────┐  ┌────────────┐ │
│  │ 💰          │  │ 🔔          │ │
│  │ Budget     │  │ Reminders  │ │
│  │ $2,340     │  │ 3 due      │ │
│  └────────────┘  └────────────┘ │
│  ┌────────────┐  ┌────────────┐ │
│  │ 💪          │  │ ➕          │ │
│  │ Habits     │  │ Add New    │ │
│  │ 🔥 12       │  │ Module     │ │
│  └────────────┘  └────────────┘ │
│                                  │
│  Upcoming                        │
│  ────────────────────────────────│
│  🔔 Clean room — Wed 9am          │
│  🔔 Dentist — Mar 15              │
│                                  │
│  Recent                          │
│  ────────────────────────────────│
│  💰 Groceries — $45.00            │
│  💰 Coffee — $5.50                │
│  💪 Meditated ✓                   │
│                                  │
├────────┬────────┬─────────┬─────┤
│  Home  │  Chat  │ Modules │  ⚙️  │
└────────┴────────┴─────────┴─────┘
```

Each module card shows a **summary based on its type**:

- Budget → total spent this month
- Reminders → count of upcoming
- Habits → current streak

```
class ModuleSummaryCard extends ConsumerWidget {
  final ModuleModel module;
  Widget build(BuildContext context, WidgetRef ref) {
    final entries = ref.watch(entriesProvider(module.id));

    final summary = switch (module.type) {
      ModuleType.budget    => _budgetSummary(entries),
      ModuleType.reminders => _reminderSummary(entries),
      ModuleType.habits    => _habitSummary(entries),
      _                    => '${entries.length} entries',
    };

    return Card(
      color: Color(int.parse(module.color.replaceFirst('#', '0xFF'))),
      child: InkWell(
        onTap: () => context.push('/modules/${module.id}'),
        onLongPress: () => _showModuleOptions(context, module),
        child: Padding(
          padding: EdgeInsets.all(16),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Icon(module.iconData, size: 32),
              SizedBox(height: 8),
              Text(module.name, style: Theme.of(context).textTheme.titleMedium),
              Text(summary, style: Theme.of(context).textTheme.bodySmall),
            ],
          ),
        ),
      ),
    );
  }
}
```

Screen 2: Modules List Screen

```
┌──────────────────────────────────┐
│  My Modules                       │
│                                   │
│  ┌──────────────────────────────┐ │
│  │ 💰 My Budget            ⋮  │ │
│  │ 6 fields · 2 screens       │ │
│  │ 47 entries                   │ │
│  └──────────────────────────────┘ │
│                                   │
│  ┌──────────────────────────────┐ │
│  │ 🔔 Reminders          ⋮ │ │
│  │ 5 fields · 1 screen     │ │
│  │ 12 entries               │ │
│  └──────────────────────────┘ │
│                               │
│  ┌──────────────────────────┐ │
│  │ 💪 Habits             ⋮ │ │
│  │ 4 fields · 1 screen     │ │
│  │ 8 entries                │ │
│  └──────────────────────────┘ │
│                               │
│                        [+]    │
│                               │
├────────┬────────┬──────────┬───┤
│  Home  │  Chat  │ Modules  │ ⚙️ │
└────────┴────────┴──────────┴───┘
```

Screen 3: Module View (inside a module)

```
┌─────────────────────────────┐
│  ← My Budget            ⚙️  │
│                               │
│  ┌───────────┬─────────────┐ │
│  │ Dashboard │ Transactions│ │
│  └───────────┴─────────────┘ │
│                               │
│  ┌────────────┐ ┌───────────┐ │
│  │ Total Spent│ │  Income   │ │
│  │   $2,340   │ │  $5,000   │ │
│  └────────────┘ └───────────┘ │
│                               │
│    🥧 Pie Chart                │
│   [food 40%] [rent 30%]       │
│   [transport 20%] [other]     │
│                               │
│  Recent                       │
│  ─────────────────────────    │
│  Groceries             -$45.00 │
│  Coffee                 -$5.50 │
│  Freelance work       +$500.00 │
│                               │
│                        [+]    │
│                               │
└─────────────────────────────┘
```

```
class ModuleViewScreen extends ConsumerWidget {
  final String moduleId;

  Widget build(BuildContext context, WidgetRef ref) {
    final module = ref.watch(moduleProvider(moduleId));

    return Scaffold(
      appBar: AppBar(
        title: Text(module.name),
        actions: [
          IconButton(
            icon: Icon(Icons.settings),
            onPressed: () => context.push('/modules/$moduleId/settings'),
          ),
        ],
      ),
      body: module.screens.length > 1
          ? DefaultTabController(
              length: module.screens.length,
              child: Column(
                children: [
                  TabBar(
                    tabs: module.screens.map((s) => Tab(text: s.title)).toList(),
                  ),
                  Expanded(
                    child: TabBarView(
                      children: module.screens
                          .map((screen) => ScreenRenderer(
                                screen: screen,
                                moduleId: moduleId,
                                schema: module.schema,
                              ))
                          .toList(),
                    ),
                  ),
                ],
              ),
            )
          : ScreenRenderer(
              screen: module.screens.first,
              moduleId: moduleId,
              schema: module.schema,
            ),
    );
  }
}
```

Screen 4: Module Settings Screen

```
┌─────────────────────────────────────┐
│  ← Module Settings                   │
│                                      │
│  MODULE INFO                         │
│                                      │
│  ┌────────────────────────────────┐  │
│  │ Name: [My Budget            ]  │  │
│  │ Icon: 💰  [change]             │  │
│  │ Color: ███ [change]            │  │
│  │ Type: budget                   │  │
│  └────────────────────────────────┘  │
│                                      │
│  SCHEMA (FIELDS)                     │
│                                      │
│  ┌────────────────────────────────┐  │
│  │ ≡ description (text)      ✎ │  │
│  │ ≡ amount (currency)        ✎ │  │
│  │ ≡ category (enum)          ✎ │  │
│  │ ≡ type (enum)              ✎ │  │
│  │ ≡ date (date)              ✎ │  │
│  │ ≡ budgeting_rule (enum)    ✎ │  │
│  │                                │  │
│  │ [+ Add Field]                  │  │
│  └────────────────────────────────┘  │
│                                      │
│  SCREENS                             │
│                                      │
│  ┌────────────────────────────────┐  │
│  │ Dashboard                  ✎   │  │
│  │ Transactions               ✎   │  │
│  │                                │  │
│  │ [+ Add Screen]                 │  │
│  └────────────────────────────────┘  │
│                                      │
│  AUTOMATIONS                         │
│                                      │
│  ┌────────────────────────────────┐  │
│  │ 🔔 Notify on due_date         │  │
│  │                                │  │
│  │ [+ Add Automation]             │  │
│  └────────────────────────────────┘  │
│                                      │
│  [🗑️ Delete Module]                  │
│                                      │
└─────────────────────────────────────┘
```

Screen 5: Field Editor

```
┌─────────────────────────────────────┐
│ ← Edit Field                        │
│                                     │
│ Label: [Category                ]   │
│                                     │
│ Type: [▼ enum_single            ]   │
│                                     │
│ Required: [✓]                       │
│                                     │
│ Options:                            │
│ ┌─────────────────────────────────┐ │
│ │ food                        [×] │ │
│ │ transport                   [×] │ │
│ │ housing                     [×] │ │
│ │ entertainment               [×] │ │
│ │ [+ Add Option]                  │ │
│ └─────────────────────────────────┘ │
│                                     │
│ [Save]               [Delete]       │
│                                     │
└─────────────────────────────────────┘
```

Screen 6: Entry Form (dynamic, works for any module)

```
┌─────────────────────────────────────┐
│ ← New Expense                       │
│                                     │
│ Description                         │
│ ┌─────────────────────────────────┐ │
│ │ Groceries                       │ │
│ └─────────────────────────────────┘ │
│                                     │
│ Amount                              │
│ ┌─────────────────────────────────┐ │
│ │ $ 45.00                         │ │
│ └─────────────────────────────────┘ │
│                                     │
│ Category                            │
│ ┌─────────────────────────────────┐ │
│ │ ▼ food                          │ │
│ └─────────────────────────────────┘ │
│                                     │
│  Type                               │
│  [● expense] [○ income]             │
│                                     │
│  Date                               │
│  ┌───────────────────────────────┐  │
│  │ 📅 Feb 14, 2026               │  │
│  └───────────────────────────────┘  │
│                                     │
│  50/30/20 Category                  │
│  [needs] [wants] [savings]          │
│                                     │
│           [Save Entry]              │
│                                     │
└─────────────────────────────────────┘
```

```
class DynamicEntryForm extends ConsumerWidget {
  final SchemaDefinition schema;

  Widget build(BuildContext context, WidgetRef ref) {
    return Column(
      children: schema.fields.map((field) {
        return switch (field.type) {
          FieldType.text       => TextFormField(
                                    decoration: InputDecoration(labelText: field.
          FieldType.number     => TextFormField(
                                    keyboardType: TextInputType.number,
                                    decoration: InputDecoration(labelText: field.
          FieldType.currency   => CurrencyField(label: field.label),
          FieldType.boolean    => SwitchListTile(title: Text(field.label)),
          FieldType.datetime   => DateTimePicker(label: field.label),
          FieldType.date       => DatePicker(label: field.label),
          FieldType.enumSingle => DropdownButtonFormField(
                                    items: field.options!.map((o) =>
                                      DropdownMenuItem(value: o, child: Text(o))
                                    ).toList(),
                                    decoration: InputDecoration(labelText: field.
          FieldType.enumMulti  => MultiSelectChips(
                                    options: field.options!, label: field.label),
          _                    => Text('Unsupported: ${field.type}'),
        };
      }).toList(),
    );
  }
}
```

12. Key Flutter Widgets

Screen Renderer

```
class ScreenRenderer extends StatelessWidget {
  final ScreenDefinition screen;
  final String moduleId;
  final SchemaDefinition schema;

  Widget build(BuildContext context) {
    return ListView(
      padding: EdgeInsets.all(16),
      children: screen.layout.map((widgetDef) {
        return WidgetRenderer(
          config: widgetDef,
          moduleId: moduleId,
          schema: schema,
        );
      }).toList(),
    );
  }
}
```

Widget Renderer

```
class WidgetRenderer extends StatelessWidget {
  final Map<String, dynamic> config;
  final String moduleId;
  final SchemaDefinition schema;

  Widget build(BuildContext context) {
    return switch (config['type']) {
      'summary_cards'  => SummaryCardsWidget(config: config, moduleId: moduleId),
      'entry_list'     => EntryListWidget(config: config, moduleId: moduleId),
      'entry_table'    => EntryTableWidget(config: config, moduleId: moduleId),
      'pie_chart'      => PieChartWidget(config: config, moduleId: moduleId),
      'bar_chart'      => BarChartWidget(config: config, moduleId: moduleId),
      'line_chart'     => LineChartWidget(config: config, moduleId: moduleId),
      'calendar_view'  => CalendarViewWidget(config: config, moduleId: moduleId),
      'kanban'         => KanbanWidget(config: config, moduleId: moduleId),
      'timeline'       => TimelineWidget(config: config, moduleId: moduleId),
      'tabs'           => TabsWidget(config: config, moduleId: moduleId),
      'filter_bar'     => FilterBarWidget(config: config, moduleId: moduleId),
      'quick_add_bar'  => QuickAddBarWidget(config: config, moduleId: moduleId),
      'fab'            => FabWidget(config: config, moduleId: moduleId),
      'progress_bar'   => ProgressBarWidget(config: config, moduleId: moduleId),
      _                => SizedBox.shrink(),
    };
  }
}
```

---

13. Cloud Function: The AI Brain

```
import { onCall } from 'firebase-functions/v2/https';
import Anthropic from '@anthropic-ai/sdk';

const anthropic = new Anthropic();

export const processMessage = onCall(async (request) => {
  const { userId, message } = request.data;
  const db = admin.firestore();

  // 1. Load user's current modules
  const modules = await db.collection(`users/${userId}/modules`).get();
  const moduleContext = modules.docs.map(d => ({ id: d.id, ...d.data() }));

  // 2. Load recent chat history
  const history = await db
    .collection(`users/${userId}/chat`)
    .orderBy('timestamp', 'desc')
    .limit(20)
    .get();

  // 3. Call Claude with full context
  const response = await anthropic.messages.create({
    model: 'claude-sonnet-4-5-20250929',
    max_tokens: 4096,
    system: `You are a personal assistant that manages the user's data.
    You create modules, add entries, build screen layouts, and manage automations

    EXISTING MODULES:
    ${JSON.stringify(moduleContext)}

    MODULE TYPES: reminders, budget, habits, journal, inventory, contacts, custom

    SUPPORTED FIELD TYPES:
    text, number, boolean, datetime, date, time,
    enum_single, enum_multi, currency, url, image, reference
    SUPPORTED WIDGET TYPES:
    summary_cards, section, tabs, grid,
    entry_list, entry_table, calendar_view, kanban, timeline,
    pie_chart, bar_chart, line_chart, progress_bar, stat_comparison,
    quick_add_bar, filter_bar, search, fab, action_buttons

    SUPPORTED ACTIONS:
    addEntry, editEntry, deleteEntry, navigateToModule,
    navigateToScreen, openChat, applyFilter, sortBy,
    toggleView, shareEntry, exportData, markComplete, snoozeReminder`,

    messages: formatHistory(history, message),
    tools: [createModuleTool, updateModuleTool, deleteModuleTool,
            addEntryTool, updateEntryTool, deleteEntryTool, queryEntriesTool]
  });

  // 4. Execute tool calls
  const results = await executeToolCalls(userId, response);

  // 5. Save conversation
  await saveChat(userId, message, response, results);

  return { response: response.content, results };
});
```

Claude Tool Definitions

```
const createModuleTool = {
  name: 'create_module',
  description: 'Create a new module with schema, screens, and automations',
  input_schema: {
    type: 'object',
    properties: {
      name: { type: 'string' },
      type: { type: 'string', enum: ['reminders','budget','habits','journal',
              'inventory','contacts','custom'] },
      icon:  { type: 'string' },
      color: { type: 'string' },
      schema: {
        type: 'object',
        properties: {
          fields: { type: 'array', items: {
            type: 'object',
            properties: {
              id: { type: 'string' }, label: { type: 'string' },
              type: { type: 'string', enum: ['text','number','boolean','datetime'
                      'date','time','enum_single','enum_multi','currency',
                      'url','image','reference'] },
              required: { type: 'boolean' },
              options: { type: 'array', items: { type: 'string' } }
            },
            required: ['id', 'label', 'type']
          }}
        }
      },
      screens: { type: 'array', items: {
        type: 'object',
        properties: {
          title: { type: 'string' },
          layout: { type: 'array' }
        }
      }},
      automations: { type: 'array' }
    },
    required: ['name', 'type', 'schema']
  }
};

const addEntryTool = {
  name: 'add_entry',
  description: 'Add a data entry to a module',
  input_schema: {
    type: 'object',
    properties: {
      module_id: { type: 'string' },
      data: { type: 'object' }
    },
    required: ['module_id', 'data']
  }
};

const updateModuleTool = {
  name: 'update_module',
  description: 'Update module schema, screens, or settings',
  input_schema: {
    type: 'object',
    properties: {
      module_id: { type: 'string' },
      add_fields: { type: 'array' },
      remove_fields: { type: 'array', items: { type: 'string' } },
      update_screens: { type: 'array' },
      add_automations: { type: 'array' },
      update_settings: { type: 'object' }
    },
    required: ['module_id']
  }
};
```

Executing Tool Calls

```
async function executeToolCalls(userId: string, response: any) {
  const results = [];
  for (const block of response.content) {
    if (block.type !== 'tool_use') continue;

    switch (block.name) {
      case 'create_module': {
        const ref = await db.collection(`users/${userId}/modules`).add({
          ...block.input,
          created_by: 'ai',
          created_at: admin.firestore.FieldValue.serverTimestamp(),
        });
        results.push({ tool: 'create_module', module_id: ref.id, success: true })
        break;
      }
      case 'add_entry': {
        const { module_id, data } = block.input;
        const ref = await db
          .collection(`users/${userId}/modules/${module_id}/entries`)
          .add({
            ...data,
            created_by: 'ai',
            created_at: admin.firestore.FieldValue.serverTimestamp(),
          });
        results.push({ tool: 'add_entry', entry_id: ref.id, success: true });
        break;
      }
      case 'update_module': {
        const { module_id, add_fields, update_screens } = block.input;
        const updates: any = {};
        if (add_fields) {
          const doc = await db.doc(`users/${userId}/modules/${module_id}`).get();
          const existing = doc.data()?.schema?.fields || [];
          updates['schema.fields'] = [...existing, ...add_fields];
        }
        if (update_screens) updates['screens'] = update_screens;
        await db.doc(`users/${userId}/modules/${module_id}`).update(updates);
        results.push({ tool: 'update_module', success: true });
```

```
          break;
        }
      }
    }
    return results;
  }
```

14. Firestore Access in Flutter

```
// Get all modules
FirebaseFirestore.instance
    .collection('users/$userId/modules')
    .snapshots();

// Get entries for a module
FirebaseFirestore.instance
    .collection('users/$userId/modules/$moduleId/entries')
    .orderBy('created_at', descending: true)
    .snapshots();

// Add an entry
FirebaseFirestore.instance
    .collection('users/$userId/modules/$moduleId/entries')
    .add({
      'description': 'Groceries',
      'amount': 45.00,
      'created_at': FieldValue.serverTimestamp(),
      'created_by': 'manual',
    });

// Send message to AI
final result = await FirebaseFunctions.instance
    .httpsCallable('processMessage')
    .call({'message': userMessage});
```

15. Conversation Flow Examples

Creating a Budget Module

```
User: "I want to track my budget"
```

```
AI → create_module({
  name: "My Budget",
  type: "budget",
  icon: "wallet",
  color: "#4CAF50",
  schema: { fields: [...] },
  screens: [
    { title: "Dashboard", layout: [...] },
    { title: "Transactions", layout: [...] }
  ]
})

AI: "Created a Budget module with a dashboard and transactions view."
```

Adding Data Through Chat

```
User: "Spent $45 on groceries"

AI → add_entry({
  module_id: "budget_abc",
  data: { description: "Groceries", amount: 45, category: "food",
          type: "expense", date: "2026-02-14", budgeting_rule: "needs" }
})
```

Modifying a Module

```
User: "Add a notes field to my budget"

AI → update_module({
  module_id: "budget_abc",
  add_fields: [{ id: "notes", label: "Notes", type: "text", required: false }]
})
```

Creating a Reminders Module

```
User: "Set up reminders. I need recurring and one-time events."

AI → create_module({
  name: "My Reminders",
  type: "reminders",
  icon: "bell",
  schema: { fields: [
    { id: "title", type: "text", required: true },
    { id: "due_date", type: "datetime" },
    { id: "recurrence", type: "enum_single", options: ["none","daily","weekly","m
    { id: "priority", type: "enum_single", options: ["low","medium","high"] }
  ]},
  screens: [{ title: "Reminders", layout: [...] }],
  automations: [{
    trigger: "field:due_date",
    action: "send_notification",
    config: { message_template: "Reminder: {{title}}" }
  }]
})
```

---

16. Notification Flow

1. Flutter app registers with FCM, stores token at `users/{userId}/devices/`
2. User creates reminder entries (via chat or manual)
3. Cloud Function trigger on entry creation checks module type
4. If type is "reminders" → registers Cloud Scheduler job
5. Cloud Scheduler fires → Cloud Function sends FCM push
6. User gets native notification

```
export const onEntryCreated = onDocumentCreated(
  'users/{userId}/modules/{moduleId}/entries/{entryId}',
  async (event) => {
    const entry = event.data?.data();
    const { userId, moduleId } = event.params;
    const moduleDoc = await db.doc(`users/${userId}/modules/${moduleId}`).get();
    const module = moduleDoc.data();

    if (module?.type === 'reminders' && entry?.due_date) {
      await scheduler.createJob({
        name: `reminder-${event.params.entryId}`,
        schedule: cronFromEntry(entry),
        httpTarget: {
          uri: `https://<region>-<project>.cloudfunctions.net/sendReminder`,
          body: Buffer.from(JSON.stringify({ userId, moduleId, entryId: event.par
        }
      });
    }
  }
);
```

17. Flutter Project Structure

```
lib/
├── main.dart
├── app/
│   ├── router.dart
│   └── providers.dart
│
├── features/
│   ├── home/
│   │   ├── home_screen.dart
│   │   └── widgets/
│   │       ├── module_summary_card.dart
│   │       ├── upcoming_section.dart
│   │       └── recent_section.dart
│   │
│   ├── chat/
│   │   ├── chat_screen.dart
│   │   ├── chat_provider.dart
│   │   └── widgets/
│   │       ├── message_bubble.dart
│   │       └── tool_result_card.dart
│   │
│   ├── modules/
│   │   ├── modules_list_screen.dart
│   │   ├── module_view_screen.dart
│   │   ├── module_settings_screen.dart
│   │   ├── module_provider.dart
│   │   └── widgets/
│   │       ├── module_list_tile.dart
│   │       └── module_type_picker.dart
│   │
│   ├── schema/
│   │   ├── schema_editor_screen.dart
│   │   ├── field_editor_screen.dart
│   │   └── widgets/
│   │       ├── field_list_tile.dart
│   │       ├── field_type_picker.dart
│   │       └── options_editor.dart
│   │
│   ├── screens/
│   │   ├── screen_renderer.dart
│   │   ├── screen_editor.dart
│   │   └── widgets/
│   │       ├── widget_renderer.dart
│   │       ├── summary_cards_widget.dart
│   │       ├── entry_list_widget.dart
│   │       ├── entry_table_widget.dart
│   │       ├── calendar_view_widget.dart
│   │       ├── kanban_widget.dart
│   │       ├── timeline_widget.dart
│   │       ├── pie_chart_widget.dart
│   │       ├── bar_chart_widget.dart
│   │       ├── line_chart_widget.dart
│   │       ├── progress_bar_widget.dart
│   │       ├── filter_bar_widget.dart
│   │       ├── quick_add_bar_widget.dart
│   │       ├── tabs_widget.dart
│   │       └── fab_widget.dart
│   │
│   ├── entries/
│   │   ├── entry_form_screen.dart
│   │   ├── entries_provider.dart
│   │   └── widgets/
│   │       └── dynamic_field.dart
│   │
│   └── settings/
│       └── settings_screen.dart
│
├── models/
│   ├── module_model.dart
│   ├── schema_definition.dart
│   ├── field_model.dart
│   ├── screen_definition.dart
│   ├── automation_model.dart
│   ├── entry_model.dart
│   └── chat_message_model.dart
│
├── services/
│   ├── ai_service.dart
│   ├── action_handler.dart
│   ├── notification_service.dart
│   └── module_manager.dart
│
└── core/
    ├── constants.dart
    ├── enums.dart
    ├── theme.dart
    └── utils.dart
```

18. Go Router Setup

```
final router = GoRouter(
  routes: [
    ShellRoute(
      builder: (context, state, child) => AppShell(child: child),
      routes: [
        GoRoute(path: '/',         builder: (_, __) => HomeScreen()),
        GoRoute(path: '/chat',     builder: (_, __) => ChatScreen()),
        GoRoute(path: '/modules',  builder: (_, __) => ModulesListScreen()),
        GoRoute(path: '/settings', builder: (_, __) => SettingsScreen()),
      ],
    ),
    GoRoute(
      path: '/modules/:moduleId',
      builder: (_, state) => ModuleViewScreen(
        moduleId: state.pathParameters['moduleId']!,
      ),
      routes: [
        GoRoute(path: 'settings',
          builder: (_, state) => ModuleSettingsScreen(
            moduleId: state.pathParameters['moduleId']!)),
        GoRoute(path: 'schema',
          builder: (_, state) => SchemaEditorScreen(
            moduleId: state.pathParameters['moduleId']!)),
        GoRoute(path: 'entries/new',
          builder: (_, state) => EntryFormScreen(
            moduleId: state.pathParameters['moduleId']!)),
        GoRoute(path: 'entries/:entryId',
          builder: (_, state) => EntryFormScreen(
            moduleId: state.pathParameters['moduleId']!,
            entryId: state.pathParameters['entryId'])),
      ],
    ),
  ],
);
```

19. Quick Reference

What	Where
Module definition	`users/{userId}/modules/{moduleId}` document
Module's data entries	`users/{userId}/modules/{moduleId}/entries/` subcollection
Chat history	`users/{userId}/chat/` collection
Device tokens	`users/{userId}/devices/` collection
View all modules	**Home Screen** (cards) + **Modules List Screen**
Use a module	**Module View Screen** (renders blueprint tabs)
Edit a module	**Module Settings Screen**
Add data manually	**Entry Form Screen** (dynamic form)
Add data via AI	**Chat Screen** → AI calls `add_entry`

---

20. Competitive Landscape

App	What It Does	How Yours Is Different
Notion + AI Agent	Workspace with AI that builds databases/forms	Fixed UI — AI helps within Notion's structure
Airtable Cobuilder	AI builds tables + interfaces from prompts	Business-focused, no chat, no notifications
Baserow + Kuma	Open source DB with AI assistant	Web-only, no mobile-first, no personal assistant
NocoDB	Open source Airtable alternative	No AI, no chat interface

Your unique position: No existing app starts empty, is chat-first on mobile, uses a module system with type-based behavior activation, lets AI create both data structure AND UI, and allows manual editing of everything.

---

21. Development Phases

Phase 1: Foundation (Week 1–2)

- Firebase project setup (Firestore, Auth, Cloud Functions)
- Firebase Auth (Google + Apple sign-in)
- Basic chat UI, Cloud Function → Claude API
- Bottom navigation shell

Phase 2: Module System (Week 3–4)

- `create_module` tool, module creation via chat
- Module list screen, module view screen (basic entry list)
- Dynamic entry form, `add_entry` tool
- Manual entry creation

Phase 3: Screen Blueprints (Week 5–7)

- 5 core widget renderers (summary_cards, entry_list, pie_chart, bar_chart, fab)
- Screen renderer, blueprint tabs in module view
- Action handler system
- Home screen with module summary cards

Phase 4: Module Settings & Manual Editing (Week 8–9)

- Module settings screen, schema editor, field editor
- `update_module` tool
- Manual module creation (without chat)

Phase 5: Notifications & Type System (Week 10–11)

- FCM setup, device token registration
- Module type → capability mapping, service activation
- Cloud Scheduler for reminders, push delivery

Phase 6: Polish (Week 12–14)

- Remaining widget types (kanban, calendar, timeline)

- Screen editor, query/filter via chat
- Onboarding, error handling
- App Store / Play Store prep

---

22. Future Enhancements

- **WebView escape hatch** — for the 1% of screens blueprints can't handle
- **Module relationships** — reference fields linking entries across modules
- **Import/Export** — CSV import, data export
- **Shared modules** — share a module with other users
- **Home screen widgets** — Android/iOS widgets showing module data
- **Voice input** — talk to the assistant instead of typing
- **Smart suggestions** — AI proactively suggests modules based on usage
- **Templates** — pre-built module configs installable with one tap
- **Cross-module queries** — "How much did I spend on days I didn't exercise?"