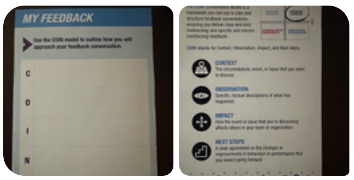

I need to create prompts for GitHub Copilot to build an iPhone app that records and shares COIN Feedback, based on the attached screenshot.

Requirements:

Sharing: Share feedback via email using native iOS features.

Syncing: Persist data in iCloud and sync it across all Apple devices.

History & Tracking: Store a list of past feedback. Indicate when it was shared. To record who the recipient was, the app should handle the recipient input internally before handing off to the email client.

Monetization: Implement a paywall. I need a recommendation on the best pricing model and price (Subscription vs. One-time).

Design: The interface should use calm colors and implement a 'Liquid Glass' (Glassmorphism) aesthetic. Note: Copilot lacks good references for this, so the prompts must be very specific.

Workflow: I need prompt templates to reuse for future enhancements or bug fixes.

Thought for 9s

Below is a “Copilot Prompt Pack” you can copy/paste into GitHub Copilot Chat inside Xcode. It’s designed to get you from zero → working COIN feedback app, with iCloud sync, email sharing, history tracking, and a glassmorphism (“Liquid Glass”) calm UI.

Notes:

- When I say “write code”, paste the prompt into Copilot Chat and let it generate the code, then iterate with the follow-up prompts.
- This assumes SwiftUI + SwiftData + CloudKit (iCloud sync) + StoreKit 2 (paywall).

==========

1. App blueprint prompt (foundation)
   ==========

Prompt:
You are building an iOS app called “COIN Feedback”. Tech stack:

- SwiftUI
- SwiftData for persistence
- iCloud sync via SwiftData + CloudKit
- StoreKit 2 for monetization
- Email sharing using native iOS email composer (MFMailComposeViewController) from SwiftUI
- Target iOS 17+

Core concept:

COIN model = Context, Observation, Impact, Next steps.
User writes feedback using 4 fields and saves entries. User can share an entry via email. App stores history of entries and shows if/when it was shared and who it was shared to.

Generate:

1. A high-level file structure (folders + key files).
2. Data model (SwiftData @Model) for:
   - FeedbackEntry: id, createdAt, updatedAt, context, observation, impact, nextSteps, recipientName, recipientEmail, sharedAt, shareCount
   - Optional: ShareEvent: id, entryId, sharedAt, recipientEmail (if you think it's needed)
3. Navigation + screens:
   - Home / History list
   - Create/Edit COIN entry screen
   - Detail screen with Share button
   - Paywall screen + gating logic
4. iCloud sync approach using SwiftData + CloudKit: exact project settings required, entitlements, and code-level configuration.
   Keep the answer actionable, with code scaffolds where necessary.

==========

2) SwiftData + iCloud CloudKit setup prompt (do this early)

Prompt:
Implement SwiftData persistence with iCloud sync using CloudKit.
Requirements:

- Provide the exact Swift code for creating the ModelContainer used by the app.
- Use a single schema containing FeedbackEntry (and ShareEvent only if needed).
- Ensure it works with iCloud across devices.

Also provide:

- The exact Xcode project steps and capabilities to enable:
  - iCloud
  - CloudKit
  - Any containers needed
- Required entitlements
- Common pitfalls and how to validate sync (ex: create on iPhone, verify on iPad)
  Keep it iOS 17+ and SwiftUI App lifecycle.

==========

3) Data model prompt (SwiftData models)

Prompt:
Write SwiftData models for this app using @Model.
Model: FeedbackEntry
Fields:

- id: UUID
- createdAt: Date
- updatedAt: Date
- context: String
- observation: String
- impact: String
- nextSteps: String

- recipientName: String (optional, stored even if email is used)
- recipientEmail: String (optional, stored before share)
- sharedAt: Date? (set when share completes)
- shareCount: Int
  Computed helpers:
- isShared: Bool
- coinSummary: String (short summary for list)
  Constraints / UX:
- The 4 COIN fields should be non-empty to enable "Share".
- Keep max lengths reasonable: each COIN field 800 chars; recipientEmail 254 chars.
  Provide code for model + simple validation helpers (non-UI).

==========

4) History list UI prompt (calm + glassmorphism)

Prompt:

Build the "History" screen in SwiftUI.

Data:

- Fetch FeedbackEntry list sorted by createdAt descending using SwiftData @Query.
  UI requirements:
- Calm colors, glassmorphism ("Liquid Glass") aesthetic:
  - Background: soft gradient (very subtle) from cool blue to violet at low saturation.
  - Cards: semi-transparent frosted glass effect with blur, 1px light border, subtle shadow.
  - Typography: large title, clean spacing, no clutter.
    List rows (cards) must show:
- A small COIN badge (C/O/I/N letters vertically or compact)
- Created date
- One-line preview (coinSummary)
- Shared status indicator:
  - If sharedAt exists: show "Shared" + date and a small icon
  - Else show "Not shared"
    Interaction:
- Tap row → Detail screen
- Floating + button or top-right + to create new entry
  Provide code for:
- HistoryView
- EntryCardView (glass card)
- A reusable GlassCard modifier or component (SwiftUI) that simulates "Liquid Glass".

Be very specific in how to implement glass:

- Use .background(.ultraThinMaterial) or .thinMaterial layered with a blurred gradient behind it
- Add an overlay stroke with low-opacity white
- Add a soft shadow
- Ensure good contrast in light and dark mode

==========

5) Create/Edit COIN screen prompt (matches the worksheet)

Prompt:

Create a "COIN Editor" screen in SwiftUI that mirrors the worksheet concept.

Layout:

- Header "MY FEEDBACK"

- Four sections labeled: C Context, O Observation, I Impact, N Next steps
- Each section has a multiline text editor with placeholder text and character counter.
- Calm, glassmorphism styling consistent with History screen.

Behavior:

- Support both create and edit.
- Autosave updatedAt.
- Save button:
  - Enabled when all 4 fields are non-empty
  - Saves to SwiftData ModelContext
- Optional recipient section:
  - Recipient name
  - Recipient email
  - Store these in the entry before sharing (the app must capture recipient internally)
- Provide accessibility labels.

Also implement:

- Placeholder behavior for TextEditor (custom overlay).
- Keyboard toolbar with “Done”.

Deliver code for:

- CoinEditorView
- Reusable CoinSectionView component

==========

6) Detail screen + Share flow prompt (email with recipient captured internally)

Prompt:
Build an EntryDetailView.
Show:

- The 4 COIN fields in read-only glass cards
- Recipient info if present
- Shared status:
  - “Not shared yet” if sharedAt is nil
  - “Shared on <date> to <recipientEmail>” if sharedAt exists
    Actions:
- Edit
- Share (email)

Email sharing requirements:

- Use MFMailComposeViewController (MessageUI) presented from SwiftUI.
- The app must capture recipientName + recipientEmail internally via a SwiftUI form before presenting the mail composer.
- The mail composer must pre-fill:
  - To: recipientEmail
  - Subject: “COIN Feedback”
  - Body: formatted text:
    Context:
    Observation:
    Impact:
    Next steps:
- When the email composer finishes with result = sent:
  - Set sharedAt to Date()

    - Increment shareCount
    - Persist changes
- If cancelled/failed: do not set sharedAt, but you may log an attempted share if you created ShareEvent.

Provide:

- SwiftUI wrapper for MFMailComposeViewController
- A "RecipientPromptSheet" that asks for name/email before opening email
- Validation for email format (basic)

==========

7) Paywall strategy recommendation (pricing model + what to gate)

Recommendation (simple and effective):

- Model: freemium + optional unlock
- Gate: "Sharing via email" + "Unlimited history"
- Free tier: create up to 10 entries, view history, edit entries
- Paid tier: unlimited entries + email sharing + share tracking + iCloud sync across devices (you can keep sync free, but many apps bundle it with Pro)

Best monetization option for a small, focused utility:

- Primary: subscription (low price) because it funds ongoing maintenance, iCloud/CloudKit overhead, and new features.
- Add a lifetime one-time purchase as an alternative for people who hate subscriptions.

Suggested prices (Canada/US typical utility app range):

- Subscription: $1.99/month
- Annual: $14.99/year (feels meaningfully discounted)
- Lifetime: $24.99 one-time

If you want simpler:

- Only annual + lifetime:
    - $14.99/year
    - $29.99 lifetime

==========

8) Paywall implementation prompt (StoreKit 2)

Prompt:
Implement a paywall using StoreKit 2 for iOS 17+ with SwiftUI.
Products:

- com.femsoftware.coinfeedback.pro.monthly (auto-renew)
- com.femsoftware.coinfeedback.pro.yearly (auto-renew)
- com.femsoftware.coinfeedback.pro.lifetime (non-consumable)

Requirements:

- Create a PurchaseManager (ObservableObject) that:
    - Loads products
    - Purchases product
    - Restores purchases
    - Listens for transaction updates
    - Exposes entitlement state: isProUser Bool
- Persist entitlement state reliably (based on current entitlements, not UserDefaults guesswork).
- Create PaywallView:

    - Glassmorphism UI consistent with app
    - Show benefits list (unlimited entries, email sharing, share tracking, iCloud sync)
    - Show monthly, yearly, lifetime options
    - Restore Purchases button
- Add gating:
    - If non-Pro and user tries to Share OR exceeds 10 entries → show PaywallView modally.

Provide full code for PurchaseManager and PaywallView and the gating logic.

==========

9) “Liquid Glass” style system prompt (design tokens + reusable components)

Prompt:

Create a small design system for calm “Liquid Glass” UI.

Define:

- AppColors: background gradient colors, primary accent, text colors for light/dark
- AppSpacing: standard paddings
- GlassStyle: a ViewModifier that applies:
    - .background(.ultraThinMaterial) with a subtle gradient behind
    - rounded corners 20–28
    - 1px border stroke with white opacity 0.15–0.25
    - shadow opacity 0.08–0.15, radius 12–20
- Buttons:
    - Primary glass button
    - Secondary text button
- Icons: use SF Symbols, keep them subtle

Deliver:

- AppTheme.swift
- GlassCard.swift (or GlassModifier.swift)
- Example usage in a sample view

Be explicit: do not use harsh neon colors. Use low saturation. Ensure legibility.

==========

10) Reusable prompt templates for future enhancements/bug fixes

Template A: Feature enhancement

Prompt:

You are working inside an existing SwiftUI + SwiftData + StoreKit2 app called “COIN Feedback”.

Goal: [describe feature]

Constraints:

- iOS 17+
- Must preserve iCloud sync with SwiftData
- Must respect paywall gating rules

  Acceptance criteria:

1. ...
2. ...
3. ...

  Edge cases:

- ...

  Deliverables:

- List the files you will create/modify

- Provide Swift code changes
- Provide any new models/migrations (if needed)
- Provide minimal unit tests or UI tests if applicable

Template B: Bug fix

Prompt:

Bug report:

- Expected: …
- Actual: …
- Steps to reproduce: …

  Context:
- App uses SwiftData + CloudKit sync
- Paywall via StoreKit 2

  Task:

1. Identify the most likely root cause
2. Propose the fix with code changes
3. Explain how to verify the fix (manual steps + any test)

Template C: Refactor / cleanup

Prompt:

Refactor request:

- Current problem: [messy view / duplicated logic / performance issue]
- Goal: [clean architecture / simpler view / better testability]

  Rules:
- No behavior changes
- Keep UI identical

  Deliverables:
- Proposed new structure
- Code diff style output (before/after snippets)
- Quick checklist to ensure nothing breaks

Template D: Performance / iCloud sync issues

Prompt:

Issue:

- Sync is slow / duplicates appear / entries missing across devices.

  Provide:

1. A diagnostic checklist for SwiftData + CloudKit
2. Code changes to reduce duplicates and ensure stable IDs
3. Logging points (what to log, where)
4. Verification steps on two devices

==========

11) One “all-in-one” Copilot prompt (if you want Copilot to generate everything incrementally)

Prompt:

Create the first working version of “COIN Feedback” iOS app (iOS 17+).

Must include:

- SwiftUI app structure
- SwiftData models (FeedbackEntry)
- SwiftData persistence + iCloud sync via CloudKit (include required project capability steps)
- History list + detail + create/edit
- Email sharing with MFMailComposeViewController:

    - capture recipientName + recipientEmail internally
    - prefill To/Subject/Body
    - on Sent: set sharedAt and increment shareCount
- Paywall using StoreKit 2 with monthly, yearly, lifetime products
    - Gate sharing and unlimited history
- Calm glassmorphism “Liquid Glass” design system with reusable GlassCard modifier

Deliver in chunks:

1. File structure + models + persistence container
2. Views (History, Editor, Detail)
3. Mail composer wrapper + share tracking
4. PurchaseManager + PaywallView + gating
5. Polish pass for glass UI + accessibility

Avoid placeholder pseudo-code; write real Swift code.

==========

If you want, paste what you’re using today (SwiftData or CoreData, iOS target, and whether you want iPad/macOS too), and I’ll tailor these prompts so Copilot generates fewer “wrong guess” paths and more “drop-in” code.