Codex / Claude Prompt - BookWords Native iOS TestFlight MVP v3

Access-First, Pure Swift / SwiftUI iPhone App for TestFlight

Context
You are building BookWords as a real native iPhone app, not another web prototype. BookWords started as a React/Next.js clickable prototype, but the next phase is a native iOS beta that can be installed on an iPhone through TestFlight and later prepared for public App Store release.

BookWords is a parent-focused iPhone app that helps a parent capture an unknown printed word while reading a physical book with a child, then automatically creates a vocabulary flashcard. The anchor use case is a child reading Harry Potter, not knowing the word "craning," and the parent quickly opening the app, pointing the camera at the word, capturing it, seeing a small confirmation, and returning to reading.

The core problem

The parent-child reading moment is the real experience. The phone should appear for only a few seconds, capture the word, and disappear. The app must reduce friction, not become a second screen-time activity.

The product promise

Open app -> native camera is ready -> center one printed word in the horizontal target -> tap capture -> small confirmation chip -> flashcard is created automatically -> parent returns to reading.

Latest founder decisions

Use these decisions as final unless Colby explicitly changes them:
1. Build the first native beta as pure Swift / SwiftUI iOS. Do not use React Native, Expo, a WebView wrapper, or a PWA.
2. Give BookWords its own GitHub repository. Do not continue building it inside MINERVA unless explicitly asked later.
3. App Store display name: BookWords.
4. First TestFlight beta may include physical card ordering and payment as a polished demo flow, but it must not charge real money.
5. Everything outside ordering/payment should be as functional and real as practical: real native camera, real Apple text recognition/OCR, real dictionary lookup, real local storage, real validation, real review flow, real tests.
6. Do not ship hard-coded definitions in the runtime app. Test fixtures can use sample data, but the app's normal card creation path must fetch definitions from a real source or show a clear failure state.
7. Assume Colby will enroll in the Apple Developer Program as an Individual unless he later decides to create an organization account.
8. The first response from Codex/Claude must ask for safe GitHub connection/repo access and Apple Developer/TestFlight setup status before coding, without asking for passwords, tokens, private keys, or raw secrets.

First response from Codex/Claude - access and setup gate

Before writing code, the agent's first response must be a short setup/access checklist. Do not ask broad product questions first. Ask only what is needed to safely start the native iOS build.

Use this exact intent:
"Before I code, I need to confirm repo access and the Apple signing/TestFlight path. I will not ask for passwords, tokens, .p8 files, certificates, provisioning profiles, or API keys in chat."

The first response must ask Colby to confirm these items:
1. GitHub repo: create or connect a private repo named bookwords-ios, then authorize the coding agent to access only that repo through the official GitHub connector/app or through a local clone.
2. Branch: confirm the agent should work on feature/native-ios-testflight-mvp, not main.
3. Apple Developer status: confirm whether Colby's Apple Developer Program membership is active and whether it is Individual or Organization. Default assumption: Individual.
4. Apple signing route: confirm that Colby will sign in to Xcode on his own Mac with his own Apple ID and select his Developer Team under Signing & Capabilities.
5. Bundle identifier: ask whether to use the placeholder com.colbyfoster.bookwords until Colby confirms the final Bundle ID.
6. TestFlight: confirm that the first beta target is TestFlight and that Colby will handle the first manual archive/upload from Xcode unless automation is explicitly requested later.

7. Secrets: if external services are needed, ask only for service configuration and environment variable names, never for secret values in chat.

If the GitHub repo is already available in the coding environment, continue with inspection immediately after this setup note. If Apple Developer access is not ready yet, continue building the app locally with placeholder signing settings and document exactly what Colby must do later in Xcode/App Store Connect.

Do not block native app development just because App Store Connect automation is not configured. Build the app, tests, docs, and QA matrix first.

Non-negotiable product rules

1. Native iPhone app.
2. Pure Swift and SwiftUI first.
3. Camera-first app.
4. App opens directly to Camera.
5. No login before first capture.
6. Parent is the user, not the child.
7. Child remains focused on the physical book.
8. Capture must be extremely fast.
9. Camera stays live after capture.
10. Confirmation behaves like the iPhone Camera thumbnail/chip, not a blocking modal.
11. OCR must prioritize the small horizontal target rectangle around one printed word.
12. Use Apple-native text recognition first.
13. Google Cloud Vision is only a server-side fallback later; never put Google credentials in the app.
14. Manual entry is a fallback, not the primary workflow.
15. Empty manual entry must never save a mock word.
16. Definitions are fixed, sourced, and read-only.
17. Kid-friendly explanations are generated from the sourced definition.
18. Book-context meaning is used only when a full sentence is confidently extracted.
19. Local-first before sign-in.
20. Sync only after sign-in is configured.
21. No gamification: no points, badges, streaks, coins, mascots, avatars, mastery percentages, or noisy animations.
22. No notifications in MVP.
23. Photos are temporary and deleted after the card is resolved/ready.
24. Digital flashcard review is free.
25. Physical ordering/payment is allowed only as a TestFlight demo feature flag until real payment/fulfillment exists.

Repository strategy

Create a standalone GitHub repository for BookWords.

Preferred repository name:
bookwords-ios

Alternative acceptable names:
BookWords
bookwords-native-ios

Initial repository rules:
1. Keep the repo private at first.
2. Use a clean README that explains what BookWords does and how to run it.
3. Use a branch for agent work:
feature/native-ios-testflight-mvp
4. Do not commit directly to main unless Colby explicitly asks.
5. Use pull requests for review.
6. Include a .gitignore appropriate for Xcode/Swift.
7. Do not commit secrets, .p8 keys, provisioning profiles, private certificates, .env files, or API keys.
8. If using Codex, use the official GitHub connector/app or local folder access. Do not ask for a GitHub password or personal access token in chat.

9. If using Claude Code, use a local clone or the official GitHub integration/action. Do not ask for a GitHub password or personal access token in chat.
10. If the repo is empty, bootstrap a new SwiftUI iOS project.
11. If the repo already contains an iOS project, inspect it before changing files.

Required seven-step execution order

Use this order as the default plan:
1. Create a new private GitHub repo named bookwords-ios.
2. Give Codex/Claude access only to that repo through the official GitHub connector/app or a local clone.
3. Build the SwiftUI app and tests on a feature branch: feature/native-ios-testflight-mvp.
4. Open a pull request and summarize the files changed, test results, defects found/fixed, and remaining setup.
5. Colby reviews the PR, then opens the Xcode project locally on his Mac.
6. Colby uses his own Apple Developer login in Xcode for Signing & Capabilities, archives the app, and uploads to App Store Connect/TestFlight.
7. Add App Store Connect API/CI automation only after the native beta builds, runs, and passes core QA locally.

The agent must not skip directly to App Store Connect automation. Manual Xcode signing/upload by Colby is safer for the first TestFlight build.

Account and secret rules

Do not ask Colby to paste secrets into chat.

Never ask for:
- Apple ID password
- GitHub password
- App Store Connect API private key
- Apple Developer certificate private key
- Supabase service role key
- Google Cloud credentials JSON
- OpenAI API key
- Stripe/payment keys
- address autocomplete provider keys
- .env values
- any raw token or private credential

If a credential is required:
1. Stop only the blocked integration.
2. Finish all unblocked work.
3. Ask Colby to configure the needed service.
4. Give the exact environment variable names or Xcode build setting names.
5. Do not request the secret value itself.
6. Confirm the app still runs with a non-deceptive fallback state when the service is missing.

Apple Developer and App Store Connect access rules

Do not ask for Colby's Apple ID password.
Do not ask Colby to paste two-factor codes, certificates, provisioning profiles, .p8 private keys, App Store Connect API keys, or any Apple credential in chat.

Default enrollment assumption
- Colby will likely enroll as an Individual Apple Developer account.
- Individual enrollment is acceptable for the first BookWords TestFlight beta.
- Organization enrollment can wait unless Colby needs company ownership, multiple internal team members under a company, or business/legal separation.
- The Account Holder/Colby must handle legal agreements, membership renewal, banking/tax setup, and final App Store submission decisions.

Preferred early beta path - manual and safe
1. Agent builds the SwiftUI/Xcode project in GitHub.
2. Agent documents exact build and test commands.
3. Colby opens the project in Xcode on his Mac.
4. Colby signs into Xcode with his own Apple ID.
5. Colby selects his Apple Developer Team under Signing & Capabilities.
6. Colby confirms or updates the Bundle ID.
7. Colby runs the app on his iPhone or simulator.
8. Colby creates an archive in Xcode and uploads to App Store Connect/TestFlight.

What the agent should ask for at the beginning
Ask for status/confirmation only:
- "Have you created or connected the private GitHub repo bookwords-ios?"
- "Is your Apple Developer Program enrollment active? Individual is fine for the first beta."
- "Do you want me to use com.colbyfoster.bookwords as a placeholder Bundle ID until you confirm the final one?"
- "Will you handle the first Xcode signing/archive/TestFlight upload locally, with me providing step-by-step instructions?"

What the agent must not ask for
Never ask for:
- Apple ID password
- two-factor authentication code
- App Store Connect API private key content
- .p8 file upload into chat
- certificate private key
- provisioning profile private material
- screenshots of sensitive account pages showing secrets

Human developer access through App Store Connect
If Colby later hires a human developer:
1. Colby goes to App Store Connect -> Users and Access.
2. Colby invites the developer by Apple ID email.
3. Colby assigns the least role needed.
4. For coding/build support, start with Developer where sufficient.
5. For TestFlight/app metadata management, App Manager may be needed.
6. Limit app access to BookWords when possible.
7. Do not assign Account Holder unless ownership transfer is intentionally required.
8. Remove access when the work is complete.

Automation access through App Store Connect API - later only
Do not set this up for the first build unless Colby explicitly asks.
If later needed:
1. Colby creates an App Store Connect API key from App Store Connect.
2. Colby stores Key ID, Issuer ID, and the .p8 private key in GitHub Actions Secrets or the CI provider's secret store.
3. The repo references those values by secret names only.
4. The .p8 file is never committed to the repo and never pasted into chat.
5. CI upload is added only after local Xcode archive/TestFlight upload has worked at least once.

Recommended secret names if CI upload is later enabled
- APP_STORE_CONNECT_KEY_ID
- APP_STORE_CONNECT_ISSUER_ID
- APP_STORE_CONNECT_API_KEY_P8
- APPLE_TEAM_ID
- IOS_BUNDLE_ID

```
Developer notes to include in docs/app-store-readiness.md
- How Colby signs into Xcode.
- How to select the Apple Developer Team.
- How to change the Bundle ID.
- How to run on a physical iPhone.
- How to archive and upload to TestFlight.
- Which steps require Colby/Account Holder.
- Which steps can be done by a coding agent without Apple credentials.
```

Swift/iOS stack

Use this stack unless a documented technical reason requires a change:
- Swift 6 or current stable Swift supported by the installed Xcode.
- SwiftUI for app UI.
- NavigationStack for navigation.
- SwiftData for local-first persistence.
- AVFoundation for live camera preview and still capture.
- Vision framework / VNRecognizeTextRequest for OCR on captured/cropped images.
- VisionKit / DataScannerViewController where available for Live Text-style guidance.
- URLSession for dictionary/API calls.
- AuthenticationServices for Sign in with Apple later.
- StoreKit only if needed later; do not use In-App Purchase for the mocked physical goods demo.
- Keychain for tokens only after real auth exists.
- XCTest and XCUITest for unit and UI tests.
- Accessibility identifiers on all critical buttons and controls.

Minimum deployment target

Prefer iOS 17+ if it materially simplifies SwiftData and Vision/VisionKit implementation. If choosing iOS 18+ or iOS 16, document the reason and tradeoff. The first beta is iPhone-first. iPad support can be passive but is not a priority.

App identity

Display name: BookWords.

Bundle identifier is not finalized. Use a placeholder during local work such as:
com.colbyfoster.bookwords

Before TestFlight upload, ask Colby to confirm the final Bundle ID. Do not assume ownership of bookwords.com or any company domain unless confirmed.

Orientation/device

- iPhone portrait-first.
- Lock to portrait for MVP unless there is a strong reason not to.
- Support standard iPhone screen sizes.
- Design and test against modern iPhones first.

Xcode project expectations

Create a project that a nontechnical founder can open in Xcode.

```
Preferred layout:
BookWords/
BookWords.xcodeproj or BookWords.xcworkspace
BookWords/
App/
Models/
Views/
Services/
Resources/
BookWordsTests/
BookWordsUITests/
README.md
docs/
setup.md
app-store-readiness.md
qa-matrix.md
```

If direct .xcodeproj generation is unreliable, use XcodeGen and include project.yml plus exact setup commands. Do not leave the project in a state that requires undocumented manual fixes.

Architecture

Use a modular architecture with injectable services so tests can use fakes and the app can use real services.

Suggested folders:

```
App/
- BookWordsApp.swift
- AppRootView.swift
- AppEnvironment.swift
- FeatureFlags.swift
```

```
Models/
- WordCard.swift
- CardStatus.swift
- CaptureMethod.swift
- RecognitionProvider.swift
- OCRResult.swift
- DictionaryEntry.swift
- ChildProfile.swift
- PhysicalCardOrder.swift
```

```
Views/Camera/
- CameraView.swift
- CameraPreview.swift
- WordTargetOverlay.swift
- CaptureChip.swift
- ManualEntryFallbackView.swift
- CameraPermissionView.swift
```

```
Views/Library/
- LibraryView.swift
- CardRowView.swift
- EmptyLibraryView.swift
```

```
Views/CardDetail/
- CardDetailView.swift
- EditWordSheet.swift
- EditSentenceSheet.swift
- StatusMetadataSheet.swift
- RegenerateExplanationSheet.swift
```

```
Views/Review/
- FlashcardReviewView.swift
- FlashcardFaceView.swift
```

```
Views/Account/
- AccountView.swift
- ChildProfileView.swift
- SignInView.swift
- PrivacyInfoView.swift
```

```
Views/OrderDemo/
- OrderSelectCardsView.swift
- PhysicalCardPreviewView.swift
- ShippingAddressView.swift
- MockCheckoutView.swift
- OrderConfirmationView.swift
```

```
Views/Shared/
- BottomTabBar.swift
- PrimaryButton.swift
- NativeSheet.swift
- StatusPill.swift
- ToastView.swift
```

```
Services/
- CameraService.swift
- TextRecognitionService.swift
- DictionaryService.swift
- CardEnrichmentService.swift
- LocalCardStore.swift
- SyncService.swift
- AuthService.swift
- ImageRetentionService.swift
- OrderDemoService.swift
```

```
Tests/
- WordValidationTests.swift
- DictionaryServiceTests.swift
- CardStatusLogicTests.swift
- ImageRetentionTests.swift
- LocalCardStoreTests.swift
- BookWordsUITests.swift
```

Core navigation

Use exactly three bottom tabs:
1. Library
2. Camera - prominent center action
3. Account

Do not add Review, Cart, or Order as bottom tabs. Review and Order live inside Library and Card Detail.

The app opens to Camera on first launch. Camera remains the default main action.

Native iPhone interaction patterns

Use native-feeling iOS patterns:
- prominent center Camera action
- iOS-style bottom sheets
- large tap targets
- subtle haptics on capture/save
- rounded cards
- system typography
- SF Symbols where appropriate
- native swipe-to-delete in Library
- native text inputs and keyboards
- Dynamic Type support
- VoiceOver labels
- clean empty/error/loading states
- no browser-looking controls

Visual design direction

BookWords should feel calm, premium, and Apple-like. It should not feel like a school game or generic flashcard app.

Use:
- white, black, soft grays, restrained blue accent
- SF system font for UI
- a refined literary serif only for the main vocabulary word on flashcards and physical card previews
- generous spacing
- calm hierarchy
- minimal copy
- subtle feedback

Avoid:
- kid colors
- Harry Potter branding
- fantasy graphics
- dashboard tables
- cluttered edit forms
- badges/streaks/coins
- decorative animation
- fake onboarding
- login walls

Camera screen

The Camera screen is the heart of BookWords.

Camera permission:
- Add NSCameraUsageDescription:
"BookWords uses the camera to capture words from physical books and turn them into flashcards."
- Request camera permission only when needed.
- If permission is denied, show a clear permission state and keep manual entry available.
- Do not make the app useless when camera access is denied.

Active camera UI:
- full-screen live camera preview
- one large outer focus box
- one small horizontal word target rectangle inside it
- the inner rectangle must be horizontal, not square
- the target should match the real physical use case: one printed word inside a tight horizontal box
- shaded region outside the target
- minimal instruction: "Center one word, then tap Capture."
- large capture button at bottom center
- bottom tab bar visible
- no top X button
- no top Account button
- no top Cards button
- no Flash button unless it is fully functional and tested

Capture interactions:
- Tapping the capture button captures.
- Tapping the target rectangle also captures.
- Both actions call the same capture method.
- Haptic feedback on capture.
- Do not navigate away after capture.
- Do not show a blocking modal after capture.
- Keep the camera live after capture.
- Show a bottom-left chip like the iPhone Camera thumbnail.

Capture chip states:
1. "Captured"
2. "Reading word..."
3. "Saved: [word]"
4. "Needs Review"
5. "Could not read word"

Chip behavior:
- It must not cover the capture button.
- It must not hide manual fallback.
- It must be tappable.
- Tapping Saved opens Card Detail.
- Tapping Needs Review opens Card Detail/correction.

Native OCR/text recognition

Use Apple-native OCR first.

Recognition flow:
1. Capture still image from AVFoundation.
2. Map the small target rectangle from screen coordinates into image coordinates.
3. Crop or prioritize the target rectangle.
4. Run VNRecognizeTextRequest on that crop first.
5. Extract the best single-word candidate.
6. If the crop fails, optionally try a slightly larger crop around the same line.
7. Use DataScannerViewController/VisionKit where appropriate for live guidance/highlighting on supported devices.
8. Do not mark a word Ready unless OCR confidence, word shape, and dictionary lookup pass.

OCR confidence/status rules:
- Initial OCR confidence threshold: 80%.
- Ready only when OCR confidence >= 80%, word shape is clean, and dictionary lookup succeeds.
- Needs Review when OCR is low confidence, empty, suspicious, ambiguous, or dictionary lookup fails.

Clean word shape:
- letters only, or normal apostrophe/hyphen punctuation
- no random symbols
- no OCR fragments
- no numbers in MVP
- trim whitespace
- normalize case for lookup
- preserve display word cleanly

Important: Do not hard-code "craning" as the normal capture result. The word "craning" can appear in tests, sample screenshots, and controlled fixtures only.

Manual / Offline Fallback

Manual entry is always available but secondary.

Use one of these labels:
- Type a word instead
- Manual Entry
- Manual / Offline Fallback

Manual entry rules:
- Placeholder: "Enter a word"
- Save button disabled until at least one non-whitespace character is typed.
- Trim whitespace.
- Empty save must not be possible.
- Empty input must never create "craning" or any mock word.
- On Save, create a local card and run dictionary/enrichment.
- Manual entry should not require an extra spelling confirmation step.

Dictionary lookup

Definitions must be real in normal runtime use.

Use Free Dictionary API first for MVP.

Rules:
- Fetch definition by normalized word.
- Store official definition.
- Store source label and URL.
- Store pronunciation text if available.
- Store pronunciation audio URL if available.
- Store synonyms if available.
- Cache successful lookups locally.
- Official definition is read-only.
- Parent cannot directly edit the official definition.
- AI cannot invent or overwrite the official definition.
- If lookup fails, mark Needs Review and show: "Could not verify this word. Check spelling."
- Do not use hard-coded runtime definitions.

Dictionary failure behavior:
- Card can exist locally as Needs Review.
- Card should show the captured word and clear error state.
- User can edit the word and retry lookup.
- Do not pretend a definition is sourced when it is not.

Kid-friendly explanation

Preferred beta behavior:
- Generate a kid-friendly explanation from the sourced definition through a backend endpoint.
- Do not put OpenAI or other AI keys in the iOS app.
- If backend is not configured, show a clear "Explanation pending" state or use a deterministic local simplification that is labeled as generated from the official definition, not as an official definition.
- Do not hard-code explanations as the normal path.

Rules:
- Target age: 10 / 4th grade by default.
- Use child profile if available.
- Generate only from the sourced definition.
- Do not invent meaning.
- Do not invent a book sentence.
- Do not create book-context meaning unless a full sentence is confidently extracted.

Book context

Book context is secondary to capturing the correct word.

Only create book-context meaning when:
- a full sentence around the word is confidently extracted,
- the sentence contains the captured word,
- the context is coherent.

If not confident:
- leave bookContextMeaning blank, or
- show "Needs sentence context," but
- do not block card creation.

If using the "craning" example sentence, preserve subject/pronouns. If the sentence is "Mrs. Dursley was craning her neck over the garden fence, spying on the neighbors," the explanation should refer to Mrs. Dursley/she, not he.

Local-first storage

The app must work before sign-in.

Before sign-in, the parent can:
- open camera
- grant camera permission
- capture words
- type words manually
- create cards
- view cards in Library
- edit cards
- review flashcards
- build up 10+ cards locally

Use SwiftData unless a documented reason requires Core Data.

Suggested WordCard fields:
- id: UUID
- word: String
- displayWord: String
- pronunciationText: String?
- pronunciationAudioURL: URL?
- officialDefinition: String?
- definitionSourceLabel: String?
- definitionSourceURL: URL?
- kidFriendlyExplanation: String?

- extractedSentence: String?
- sentenceConfidence: Double?
- bookContextMeaning: String?
- synonyms: [String]
- status: Needs Review | Ready | Ordered
- captureMethod: camera | manual
- ocrConfidence: Double?
- recognitionProvider: appleVision | visionKit | googleCloudVision | manual | testFixture
- sourceImageLocalPath: String?
- sourceImageRemotePath: String?
- imageRetentionState: pending | deleted | retainedForReview
- localOnly: Bool
- synced: Bool
- createdAt: Date
- updatedAt: Date
- orderedAt: Date?

Image privacy and retention

BookWords is not a photo archive.

Rules:
- Images exist only to extract the word and resolve the card.
- Before sign-in, use local temporary storage if needed.
- If OCR and dictionary succeed and the card becomes Ready, delete the source image after card data is saved.
- If OCR fails or the card is Needs Review, retain the image only until the parent resolves the card.
- Delete the image after the parent resolves the card.
- Do not show page photos on finished flashcards.
- Do not upload photos to cloud storage unless explicitly needed for unresolved Needs Review after sign-in.
- Add debug-only logging for image deletion.

Card statuses

Use only these user-visible statuses:
- Needs Review
- Ready
- Ordered

Ready: usable for phone review and physical card ordering.
Needs Review: OCR/dictionary/context is uncertain or parent flagged it.
Ordered: included in a demo or real order flow.

Do not add extra user-facing statuses unless absolutely necessary. Internal processing states can exist but should not clutter the interface.

Library screen

The Library should feel like Apple Notes/Wallet quality, not a database.

Requirements:
- Clean empty state when there are no cards.
- Empty copy: "No cards yet. When your child finds a word they do not know, take a quick photo. BookWords will turn it into a flashcard."
- Empty primary action: "Open Camera"
- Group cards by status.
- Hide empty status groups.
- Do not repeat status labels inside each row when a section already communicates status.
- Row shows word, short definition preview, subtle chevron.
- Tapping row opens Card Detail.
- Native swipe-to-delete.
- After user confirms Delete from native swipe action, delete immediately; no second confirmation.

Library actions:
- Show "Review Ready Cards" when at least one Ready card exists.
- Show "Order 10-card pack - $9.99" only when 10+ Ready cards exist and demo ordering flag is enabled.
- Review order: newest Ready cards first.

Card Detail screen

Card Detail should feel like a polished flashcard detail page, not a form.

Main presentation:
- Large vocabulary word at top.
- Black word on white card/background.
- Literary serif for word only.
- Pronunciation below the word.
- The word is the main focus.
- Status is subtle and must not crowd the word.

Important interaction:
- Tapping the large word opens Flashcard Review with that word as the first review card.
- Add a subtle hint only if needed, such as "Tap to review."

Sections:
- Official Definition
- Kid-Friendly Explanation
- Book Context
- Pronunciation
- Synonyms
- Metadata / Source

Official Definition:
- read-only
- sourced
- not directly editable

Editable fields:
- captured word
- extracted sentence/context

Do not present every value as an editable input.

Edit Word:
- Opens iOS-style bottom sheet focused only on correcting the word.
- Word is required.
- Save disabled if empty.
- Trim whitespace.
- Show copy: "Changing the word will refresh the definition and explanation."
- Saving refreshes dictionary, pronunciation, synonyms, kid-friendly explanation, and context.
- No mock default word.

Edit Sentence:
- Opens iOS-style bottom sheet focused only on sentence/context.
- User can edit or remove sentence.
- Saving updates context.
- It should not regenerate the official definition unless word also changed.
- It may regenerate book-context explanation if sentence confidence/context changed.

Status:
- Status must not trigger the same action as Edit Word or Edit Sentence.
- Prefer read-only metadata or an iOS-style status sheet.
- Status sheet shows card status, OCR confidence, recognition provider, dictionary source, context confidence, sync state, and image retention state.

- If status can be changed manually, show Needs Review, Ready, Cancel. Ordered should be applied by the order flow.

Regenerate:
- Separate action named "Regenerate Explanation" or "Refresh Card."
- Do not make this the default action for every edit/status control.
- Explain what refreshes.
- Official definition remains sourced and read-only.

Flashcard Review

Review is parent-led and low pressure.

Rules:
- Use Ready cards, newest first.
- If launched from Card Detail by tapping a word, start with that word.
- No scores.
- No badges.
- No streaks.
- No mastery percentages.
- No correct/incorrect pressure.

Behavior:
- Front: vocabulary word only.
- Tap card to flip.
- Back: word, pronunciation, official definition, kid-friendly explanation, context if available, synonyms.
- Back button exits review.
- Next button moves to next card.
- Quiet secondary action: Mark Needs Review.
- Mark Needs Review opens iOS-style action sheet:
"Mark this card Needs Review? This will move it back so it can be corrected later."
Actions: Mark Needs Review / Cancel.

Account screen

Account is secondary. It should not be the first task.

Sections:
1. Sign In
2. Child Profile
3. Shipping
4. Payment / Checkout Preference
5. Backup / Export
6. Privacy

Sign-in rules:
- No login before first capture.
- Sign in with Apple later.
- Sign in with Google only if configured.
- No username/password login.
- If sign-in is not configured in the first beta, show setup-required/coming-soon state. Do not fake successful sync.
- Sign-in is for sync, backup, and order flow.
- Before public release, add account deletion if account creation exists.

Child profile:
- Child name optional.
- Child age optional.
- Child grade optional.
- Age and grade are separate fields.
- If age is first entered, suggest grade once.
- After parent manually changes grade, do not override it.

- For the first beta, keep child profile local-only unless sync/privacy setup is complete.

Privacy:
- Explain temporary image handling plainly.
- Explain that finished cards do not keep page photos.
- Explain local-first storage.

Sync

Sync is post-sign-in only.

Do not fake sync.

If Supabase or another backend is not configured:
- keep local-first app fully usable,
- show sign-in/sync as setup-required or disabled,
- document what is needed.

When sync is implemented:
- local cards remain usable offline,
- after sign-in, sync local cards to backend,
- handle duplicates,
- mark localOnly/synced accurately,
- do not require sync to review flashcards.

Physical card ordering and payment demo

Digital use is free. Parents pay only for mailed physical 4x6 flashcards in the future.

For first TestFlight beta:
- Ordering/payment may be a polished demo.
- It must be behind a feature flag.
- It must not charge money.
- It must not create a real order.
- It must be clear in developer notes that this is demo-only.
- Customer-facing UI can feel polished, but do not mislead testers into thinking money was charged.

Feature flag suggestion:
BookWordsFeatureFlags.demoOrderingEnabled = true for TestFlight demo builds.

For public App Store release:
- remove/disable demo checkout, or
- replace it with a real compliant physical-goods checkout and fulfillment path.

Order flow:
1. Select Cards
2. Preview Physical Cards
3. Shipping Address
4. Demo Payment / Checkout
5. Demo Order Confirmation

Order CTA:
- "Order 10-card pack - $9.99"
- Appears only when there are 10+ Ready cards and demo ordering is enabled.
- App suggests 10 cards but parent can adjust selection.
- Do not force newest 10 without review.

Physical card layout:
Front:
- vocabulary word only

Back:
- vocabulary word
- pronunciation
- short official definition
- kid-friendly explanation
- book-context meaning if available
- synonyms

Do not include:
- book title/page
- page photo
- copyrighted book images

App Store/TestFlight readiness

Required before internal TestFlight:
- Native iOS project builds in Xcode.
- Bundle identifier selected.
- App display name BookWords.
- App icon placeholder or real icon added.
- Launch screen added.
- Camera purpose string added.
- App does not crash if camera permission is denied.
- App works on a real iPhone.
- Local-first capture and card creation works.
- Dictionary lookup works over HTTPS.
- No runtime hard-coded definitions.
- Unit tests exist and pass.
- UI tests exist for critical flows.
- Build can be archived.

Required before public App Store release:
- Privacy policy URL.
- App Store privacy details completed.
- Age rating questionnaire completed.
- App Review notes written.
- Backend services live if needed for review.
- Account deletion if account creation exists.
- Demo payment removed/disabled or replaced with real compliant checkout.
- Third-party SDK privacy requirements handled.
- Accessibility QA pass.
- Crash-free QA pass.

QA role requirement

Act like a QA tester, not just a coder.

Every button/control must have:
- expected action
- expected destination
- disabled state
- loading state
- success state
- error state
- accessibility identifier
- test coverage or manual QA result

Run all practical tests available in the environment. If Xcode or iOS Simulator is unavailable, say so and provide exact commands for Colby to run locally.

Build/test commands to run where available:
xcodebuild -scheme BookWords -destination 'platform=iOS Simulator,name=iPhone 16' build
xcodebuild -scheme BookWords -destination 'platform=iOS Simulator,name=iPhone 16' test

If simulator names differ, list available simulators and choose the closest modern iPhone.

Unit tests to create

- clean word shape validation
- empty manual entry disabled/blocked
- word normalization
- OCR confidence threshold logic
- dictionary success -> Ready
- dictionary failure -> Needs Review
- source image deletion rules
- no hard-coded runtime definitions
- kid-friendly explanation uses official definition input
- local card creation before sign-in
- demo ordering feature flag logic

UI tests to create

1. App launches to Camera.
2. Camera permission denied shows fallback and manual entry.
3. Empty manual entry Save is disabled.
4. Typing one letter enables Save.
5. Manual Save creates a card and starts dictionary lookup.
6. Capture button shows chip and keeps camera screen active.
7. Tapping target rectangle triggers same capture flow.
8. Tapping chip opens Card Detail.
9. Library tab opens Library.
10. Empty Library Open Camera returns to Camera.
11. Card row opens Card Detail.
12. Tapping large word opens Flashcard Review.
13. Flashcard tap flips front/back.
14. Next advances review.
15. Mark Needs Review opens action sheet.
16. Edit Word opens only word edit sheet.
17. Edit Sentence opens only sentence edit sheet.
18. Status opens metadata/status sheet, not regeneration.
19. Regenerate Explanation is separate.
20. Swipe-to-delete deletes card.
21. Account tab opens Account.
22. Sign-in buttons do not block capture.
23. 10+ Ready cards shows order CTA only when demo ordering is enabled.
24. Order demo proceeds Select -> Preview -> Shipping -> Demo Checkout -> Confirmation.
25. Public-release flag hides demo checkout.

Button-by-button QA matrix

Produce a table in the final summary with columns:
- Screen
- Button/control
- Expected action
- Expected destination
- Disabled state
- Loading state
- Error state
- Accessibility identifier
- Test result

Defects to explicitly guard against

- App opens to account/login instead of camera.
- Camera permission denial breaks the whole app.
- Manual Save enabled on empty input.
- Empty manual Save creates "craning".
- Runtime app uses hard-coded definition for "craning" or any word.
- Capture navigates away and interrupts reading.
- Camera stops after capture.
- Captured word chip is hidden behind another panel.
- Random example sentence appears on Camera screen without functional purpose.
- Target rectangle is square instead of horizontal.
- OCR ignores the target rectangle and uses the whole page only.
- Word captured incorrectly but marked Ready.
- Definition is editable as if it were user text.
- AI explanation invents meaning not supported by dictionary source.
- Book context is generated from incomplete sentence.
- Edit Word, Edit Sentence, and Status all trigger the same action.
- Status tag crowds the main word.
- Tapping the big word does nothing.
- Login required before first capture.
- Sign-in pretends to sync when sync is not configured.
- Physical checkout appears real while still mocked.
- Source images retained after card is Ready.

Test fixtures

Do not commit copyrighted book-page images as normal app assets.

For tests:
- Use synthetic text images generated by test code or included as plain fixtures.
- It is acceptable for a test fixture to contain the word "craning".
- It is acceptable for tests to mock Free Dictionary API responses.
- It is not acceptable for the production runtime path to rely on hard-coded sample card data.

Deliverables

Produce:
1. Native SwiftUI iOS app source code.
2. Xcode project or documented XcodeGen setup.
3. Working Camera -> OCR -> Dictionary -> Card creation path.
4. Manual Entry -> Dictionary -> Card creation path.
5. Local-first Library using SwiftData or documented local persistence.
6. Card Detail screen.
7. Flashcard Review screen.
8. Account screen with honest setup states.
9. Optional demo Order/Payment flow behind feature flag.
10. Unit tests.
11. UI tests.
12. Nontechnical setup guide.
13. App Store/TestFlight readiness checklist.
14. QA defect report.
15. Button-by-button QA matrix.
16. What is real vs mocked.
17. What accounts/permissions are still needed.
18. Exact commands to build/run/test.

Minimum working beta definition

The first successful native beta is a real iPhone app that:
- installs on an iPhone through TestFlight,
- opens to the native camera,

- lets a parent capture or type a word,
- runs Apple-native OCR on the target area,
- fetches a real sourced definition,
- creates a local flashcard,
- lets the parent review flashcards on the phone,
- works without sign-in,
- uses no runtime hard-coded definitions,
- mocks only ordering/payment,
- passes the QA checklist.

Questions to ask Colby only after doing all unblocked work

1. Confirm final Bundle ID. Suggested placeholder: com.colbyfoster.bookwords.
2. Is the Apple Developer account Individual or Organization?
3. Who should own the GitHub repo: Colby's personal GitHub or a company/org GitHub?
4. Should child profile fields be local-only for beta?
5. Where should the privacy policy live?
6. Should AI enrichment backend be Supabase Edge Functions, a small hosted API, or added later after native OCR/dictionary beta works?
7. Should Google Cloud Vision fallback wait until after Apple Vision beta is tested?
8. Should TestFlight demo ordering be visible to all testers or only internal testers?

Final developer response format

When finished, respond with:
1. Summary
2. Files created/changed
3. How to open in Xcode
4. How to run on iPhone Simulator
5. How to run on a real iPhone
6. How to run tests
7. QA results and defects found/fixed
8. Button-by-button click-path matrix
9. What is real
10. What is mocked
11. TestFlight/App Store checklist
12. Account access/setup questions for Colby
13. GitHub access status and exact next action
14. Apple Developer/TestFlight status and exact next action
15. Known limitations
16. Recommended next prompt

Do not claim the app is App Store-ready unless it builds, runs, tests pass, privacy/setup requirements are documented, and demo payment/order flows are removed, disabled, or properly gated for public release.