Apollo Capture — Full Build Spec

What This Doc Is

This is the complete build spec for Apollo Capture, an STR property onboarding web app. It's broken into sections you can copy-paste directly into your build tools. Each section tells you WHICH tool to use and WHAT to paste into it.

TABLE OF CONTENTS

STEP 1: GOOGLE AI STUDIO

Build the Core Recording Web App

Copy-paste the following prompt into Google AI Studio:

PROMPT FOR AI STUDIO:

```
Build me a mobile-first web application called "Apollo Capture" for recording property walkthrough videos. This is a Progressive Web App (PWA) that runs in a mobile browser.

TECH STACK:
- Frontend: HTML/CSS/JavaScript (vanilla or lightweight framework like Preact)
- Must work on mobile Safari and Chrome
- No backend needed yet — just the frontend recording interface
```

- Use the MediaRecorder API for video
- Use the Web Speech API (SpeechRecognition) for real-time transcription

CORE FEATURES:

1. START SCREEN
  - App title "Apollo Capture" at top
  - Text input field for "Property Name" (required)
  - Text input field for "Property Address" (optional)
  - Big prominent "Start Recording" button
  - Cannot start recording without property name filled in

2. RECORDING SCREEN
  - Full-screen camera viewfinder (rear camera by default)
  - Small button to flip between front/rear camera
  - Recording timer in top-left showing elapsed time (MM:SS)
  - Red pulsing dot next to timer indicating active recording
  - Real-time transcription text scrolling at the bottom of the screen in a semi-transparent dark overlay (last 2-3 lines of what's being said, auto-scrolling)
  - "Mark Photo" floating button on the right side (large, easy to tap as backup to voice trigger)
  - "Stop Recording" button at the bottom center (red)

3. VOICE TRIGGER — "MARK PHOTO"
  - The Web Speech API runs continuously during recording
  - When the phrase "mark photo" or "take photo" is detected in the transcript:
    a. Capture the current video frame as a JPEG image
    b. Play an audible camera shutter click sound (use an Audio element with a short mp3/wav)
    c. Flash a white overlay on the screen for 100ms (simulating camera flash)
    d. Show a thumbnail of the captured photo in the bottom-right corner for 2 seconds, then fade out
    e. Store the photo with its timestamp (seconds since recording started)
  - The manual "Mark Photo" button does the same thing

4. ROOM DETECTION
  - While transcription runs, detect when the speaker names a room
  - Common room keywords to listen for: "bedroom", "bathroom", "kitchen", "living room", "dining room", "garage", "laundry", "closet", "patio", "backyard", "front yard", "pool", "office", "den", "hallway", "entryway", "master", "guest room", "loft", "basement", "attic"
  - When a room name is detected:
    a. Show a brief toast notification at the top: "📍 [Room Name] detected"
    b. Log the room boundary with its timestamp
    c. The toast should be non-intrusive and auto-dismiss after 1.5 seconds
  - Store room boundaries as: { roomName: string, timestampSeconds: number }

5. STOP & REVIEW SCREEN

- After hitting "Stop Recording", show a summary screen:
  a. Property name and address at top
  b. Total recording duration
  c. Number of photos captured
  d. Number of rooms detected (list them)
  e. Scrollable grid of all captured photos (thumbnails) — tap to view full size
  f. Each photo shows its timestamp and which room it was captured in
  g. Allow deleting individual photos (tap photo → "Delete" option)
  h. "Upload" button at the bottom
  i. "Re-record" button (warning: this will discard current recording)

6. UPLOAD SCREEN
- When "Upload" is tapped:
  a. Show upload progress bar
  b. Package everything into a payload:
    - Video file (webm or mp4 depending on browser support)
    - Full transcript as JSON with timestamps: [{ text: string, timestampSeconds: number }]
    - Photos as JPEG files with metadata: [{ photoBlob: Blob, timestampSeconds: number, associatedRoom: string | null }]
    - Room boundaries: [{ roomName: string, timestampSeconds: number }]
    - Property info: { name: string, address: string }
  c. For now, just save everything to IndexedDB locally (we'll add server upload later)
  d. Show "Upload Complete ✓" with a "Done" button that returns to start screen

7. DATA STORAGE (LOCAL FOR NOW)
- Use IndexedDB to store all recordings locally
- Each recording is a "capture session" with a unique ID
- Store: video blob, transcript array, photos array, room boundaries, property info, created timestamp
- On the start screen, show a list of previous captures below the "Start Recording" button
- Each previous capture shows: property name, date, duration, number of photos
- Tapping a previous capture opens the review screen for it

TECHNICAL REQUIREMENTS:
- The app MUST request camera and microphone permissions on first use
- Handle permission denied gracefully with a message explaining why they're needed
- The Web Speech API should use continuous mode and interim results
- Video recording should be at the highest resolution the device supports
- Photos captured from video frames should be at least 1920x1080 if possible
- All data stays on-device until explicitly uploaded
- The app should work offline after first load (service worker for PWA)
- Responsive design but OPTIMIZE FOR MOBILE — this will primarily be used on phones

UI STYLE:
- Dark theme (dark gray/black backgrounds)

- Accent color: #FF6B35 (warm orange — Apollo brand)
- Clean, minimal interface
- Large touch targets (minimum 44x44px)
- System font stack for performance
- Rounded corners on cards and buttons (8-12px border radius)

Please build the complete application with all HTML, CSS, and JavaScript.

WHAT YOU SHOULD HAVE AFTER STEP 1:

A working web app that:

- ✅ Records video from phone camera
- ✅ Runs real-time speech-to-text
- ✅ Detects "mark photo" voice command and captures frames
- ✅ Plays shutter sound and shows thumbnail preview
- ✅ Detects room names from speech
- ✅ Shows review screen with all captured data
- ✅ Stores everything locally on the phone

KNOWN LIMITATIONS TO EXPECT:

- Web Speech API may not work in all browsers (Chrome is best, Safari is iffy)
- Video quality varies by device
- Transcription accuracy depends on mic quality and background noise
- Some room names might not be detected if spoken unclearly

IF SOMETHING DOESN'T WORK:

- If Web Speech API fails on Safari: add a fallback that just records audio and transcribes later in the backend (Step 3)
- If MediaRecorder doesn't support mp4: use webm, the backend will handle conversion
- If frame capture quality is low: that's okay for v1, we'll improve later

---

STEP 2: GOOGLE STITCH

Design the Full UI

Take what AI Studio built and paste it into Stitch with this prompt:

PROMPT FOR STITCH:

I have a web app called "Apollo Capture" — it's a mobile-first tool for recording property walkthrough videos for short-term rental management. I need you to redesign the UI to look professional and polished. Here's the current functionality (don't change any functionality, just make it look amazing):

BRAND:
- App name: "Apollo Capture"
- Primary color: #FF6B35 (warm orange)
- Secondary color: #1A1A2E (deep navy/dark)
- Accent: #E8E8E8 (light gray for text on dark backgrounds)
- Dark theme throughout
- Modern, clean, professional feel — think "premium tool for property managers"
- Subtle gradients are okay
- Use Inter or system font stack

SCREENS TO DESIGN:

1. START / HOME SCREEN
   - Apollo Capture logo/wordmark at top (text-based is fine, make it stylish)
   - Property name input with a clean floating label
   - Property address input with floating label
   - Big "Start Capture" button — rounded, gradient from #FF6B35 to #FF8C42, white text, subtle shadow
   - Below the button: "Previous Captures" section
   - Each previous capture is a card showing: property name, date, duration, photo count
   - Cards should have a subtle border, rounded corners, and a right-arrow icon
   - If no previous captures, show a friendly empty state: camera icon + "No captures yet"

2. RECORDING SCREEN
   - Camera viewfinder takes up full screen
   - Top bar (semi-transparent dark gradient overlay):

- Left: Recording timer (MM:SS) with red pulsing dot
- Right: Camera flip button (icon only)

- Bottom area (semi-transparent dark gradient overlay):
- Live transcription text (last 2-3 lines, white text, auto-scrolling)
- Below transcription: room detection badge — when a room is detected, show a small pill/badge that says " 📍 Kitchen" or " 📍 Bedroom 1" etc, with a subtle slide-in animation

- Right side floating: " 📸 " button (Mark Photo) — circular, orange, with a subtle glow/pulse
- Bottom center: Stop button — large red circle with white square inside (standard stop icon)
- When a photo is captured:
- White flash overlay (100ms)
- Thumbnail appears in bottom-right with a polaroid-style frame, slides in from right, stays 2 seconds, fades out

3. REVIEW / SUMMARY SCREEN

- Header: Property name + address
- Stats row: three cards side by side
- Duration (clock icon + MM:SS)
- Photos (camera icon + count)
- Rooms (door icon + count)

- "Detected Rooms" section: horizontal scrollable pills/tags showing each room name
- "Captured Photos" section: scrollable grid (2 columns)
- Each photo card shows the image thumbnail, timestamp, and associated room name as a small badge
- Tap to view full size in a modal/lightbox
- Long-press or tap delete icon to remove

- Bottom fixed bar:
- "Re-record" button (outlined/secondary style, left side)
- "Upload & Process" button (filled orange, right side, prominent)

4. UPLOAD / PROCESSING SCREEN

- Centered content with progress animation
- Circular progress indicator (orange) with percentage
- Status text below: "Uploading video..." → "Processing transcript..." → "Analyzing rooms..." → "Complete!"
- Each step gets a checkmark when done
- When complete: big green checkmark animation
- "View in Notion" button (if Notion link is available)
- "New Capture" button to return to home

5. PHOTO LIGHTBOX (Modal)

- Full screen dark overlay
- Photo centered and scaled to fit
- Top bar: room name badge + timestamp
- Bottom: "Delete Photo" button (red, outlined)
- Tap outside or X button to close
- Swipe left/right to navigate between photos

ANIMATIONS & MICRO-INTERACTIONS:
- Buttons should have subtle scale-down on press (transform: scale(0.97))
- Cards should have subtle hover/active states
- Screen transitions should slide (left to right for forward, right to left for back)
- The recording dot should pulse with a CSS animation
- Toast notifications should slide down from top and auto-dismiss
- The upload progress should animate smoothly

RESPONSIVE:
- Designed for phones (375px - 430px width primarily)
- Should still look okay on tablets
- NOT optimized for desktop (that's fine)

Please redesign the UI with all CSS and update the HTML structure. Keep all the JavaScript functionality exactly the same — just make it beautiful.

WHAT YOU SHOULD HAVE AFTER STEP 2:

A beautiful, professional-looking web app that:

- ✅ Looks like a real product, not a prototype
- ✅ Has consistent branding and color scheme
- ✅ Feels smooth with animations and transitions
- ✅ Is easy to use with one hand on a phone
- ✅ All original functionality still works

STEP 3: GOOGLE ANTIGRAVITY (CLAUDE OPUS)

Build the AI Processing Backend

Paste the following into Antigravity:

PROMPT FOR ANTIGRAVITY — BACKEND AI PIPELINE:

Build a backend API service for "Apollo Capture" — a property walkthrough recording tool. This service receives a video recording, transcript, and photos from a mobile web app and processes them using AI into structured property data.

TECH STACK:
- Node.js with Express (or Python with FastAPI — your choice based on what's simpler)
- Anthropic Claude API (model: claude-sonnet-4-20250514) for AI processing
- FFmpeg for video processing
- Hosted on whatever is easiest (Railway, Vercel, or just a simple server)
- File storage: Cloudflare R2, AWS S3, or Google Cloud Storage (cheapest option)

API ENDPOINTS:

POST /api/capture/upload
Receives the capture data from the mobile app.

Request (multipart/form-data):
- video: video file (webm or mp4)
- transcript: JSON string — array of { text: string, timestampSeconds: number }
- photos: array of JPEG files
- photoMetadata: JSON string — array of { timestampSeconds: number, associatedRoom: string | null }
- roomBoundaries: JSON string — array of { roomName: string, timestampSeconds: number }
- propertyName: string
- propertyAddress: string

Response:
```
{
  "captureId": "uuid",
  "status": "processing",
  "message": "Upload received, processing started"
}
```

GET /api/capture/:captureId/status
Check processing status.

Response:
```
{
  "captureId": "uuid",
  "status": "processing" | "complete" | "failed",
  "progress": {
    "transcription": "complete",
    "roomSegmentation": "complete",
```

```
    "inventoryExtraction": "processing",
    "photoAssociation": "pending",
    "notionSync": "pending"
  },
  "result": null | { ... structured data ... }
}

### GET /api/capture/:captureId/result
Get the final processed result.

PROCESSING PIPELINE (runs async after upload):

STEP 1: TRANSCRIPT ENHANCEMENT
- Take the raw transcript from the Web Speech API
- If transcript quality is poor or missing, use Whisper API or AssemblyAI to re-transcribe from the video audio
- Clean up the transcript: fix common speech-to-text errors, normalize room names

STEP 2: ROOM SEGMENTATION
- Send the full transcript to Claude with this system prompt:

"""
You are an AI assistant that processes property walkthrough transcripts. The user has recorded a video tour of a short-term rental property, narrating as they go through each room.

Your job is to segment this transcript into rooms.

INPUT: A timestamped transcript of a property walkthrough.

OUTPUT: A JSON object with the following structure:
{
  "propertyOverview": {
    "totalRooms": number,
    "propertyType": "house" | "apartment" | "condo" | "townhouse" | "other",
    "estimatedBedrooms": number,
    "estimatedBathrooms": number,
    "hasOutdoorSpace": boolean,
    "generalNotes": "string — any general property notes mentioned"
  },
  "rooms": [
    {
      "roomId": "room-1",
      "roomName": "Primary Bedroom",
      "roomType": "bedroom" | "bathroom" | "kitchen" | "living_room" | "dining_room" | "garage" | "laundry" | "outdoor" | "office" | "hallway" | "closet" | "other",
```

```
      "startTimestamp": number (seconds),
      "endTimestamp": number (seconds),
      "transcriptExcerpt": "the raw transcript text for just this room segment",
      "inventory": [
        {
          "item": "Queen bed",
          "quantity": 1,
          "notes": "with white duvet cover",
          "condition": "good" | "fair" | "needs_attention" | "not_mentioned"
        }
      ],
      "features": ["ceiling fan", "en-suite bathroom", "walk-in closet"],
      "quirksAndNotes": ["Light switch is behind the door", "Window sticks a little"],
      "accessInfo": ["Key code for bedroom door: 1234"],
      "cleaningNotes": ["Carpet needs deep clean between guests"]
    }
  ],
  "propertyAccess": {
    "wifiName": "string or null",
    "wifiPassword": "string or null",
    "lockboxCode": "string or null",
    "parkingInstructions": "string or null",
    "gateCode": "string or null",
    "otherAccess": ["any other access info mentioned"]
  },
  "systemsAndUtilities": {
    "hvac": "notes about heating/cooling",
    "waterHeater": "location and notes",
    "breakerBox": "location",
    "waterShutoff": "location",
    "trashDay": "if mentioned",
    "otherSystems": ["any other system notes"]
  }
}
```

RULES:

1. Be thorough — extract EVERY item mentioned, even small things like "there's an ironing board in the closet"
2. If the speaker mentions quantities, use exact numbers. If they don't specify, use 1.
3. Room names should be normalized (e.g., "master bedroom" → "Primary Bedroom", "the kitchen area" → "Kitchen")
4. If the speaker goes back to a room they already covered, merge that content into the existing room entry
5. Capture ALL access information (WiFi, codes, keys) even if mentioned casually
6. If something is unclear in the transcript, include it with a note like "unclear — verify"
7. For timestamps, use the closest timestamp from the input transcript
8. Extract condition notes only if the speaker explicitly mentions condition

"""

STEP 3: PHOTO ASSOCIATION
- Take the captured photos and their timestamps
- Match each photo to the room it was captured in based on timestamp overlap with room segments
- If a photo's timestamp falls between a room's startTimestamp and endTimestamp, associate it with that room
- If a photo falls outside any room boundary (e.g., in a hallway transition), associate it with "unassigned"

STEP 4: GENERATE ROOM CLIPS (OPTIONAL — can skip for v1)
- Using FFmpeg, split the video into clips per room based on the timestamps
- Save clips as individual files
- This is a nice-to-have — implement if time allows

STEP 5: COMPILE FINAL RESULT
- Combine everything into a final structured output:

```
{
  "captureId": "uuid",
  "propertyName": "string",
  "propertyAddress": "string",
  "captureDate": "ISO date string",
  "recordingDuration": number (seconds),
  "propertyOverview": { ... from Claude },
  "rooms": [
    {
      ... all fields from Claude output,
      "photos": [
        {
          "photoUrl": "url to stored photo",
          "timestamp": number,
          "thumbnailUrl": "url to thumbnail"
        }
      ],
      "videoClipUrl": "url to room video clip (if generated)"
    }
  ],
  "propertyAccess": { ... },
  "systemsAndUtilities": { ... },
  "fullTranscript": "complete cleaned transcript text",
  "rawData": {
    "videoUrl": "url to full video",
    "transcriptUrl": "url to full transcript JSON"
  }
}
```

ERROR HANDLING:
- If Claude fails to parse the transcript, retry once with a simplified prompt
- If video upload fails, return clear error and allow re-upload
- If a photo fails to upload, skip it and note it in the result
- Log all errors for debugging
- Set reasonable timeouts (Claude call: 120s, video processing: 300s)

ENVIRONMENT VARIABLES NEEDED:
- ANTHROPIC_API_KEY
- STORAGE_BUCKET_URL (for photo/video storage)
- NOTION_API_KEY (used in Step 4)
- PORT

Build the complete backend with all endpoints, the processing pipeline, error handling, and file storage integration.

WHAT YOU SHOULD HAVE AFTER STEP 3:

A backend that:

- ✅ Accepts video, transcript, and photo uploads
- ✅ Processes transcript through Claude to extract structured room data
- ✅ Associates photos with correct rooms by timestamp
- ✅ Stores all files in cloud storage
- ✅ Returns structured JSON with complete property data
- ✅ Has status endpoint for tracking processing progress

STEP 4: GOOGLE ANTIGRAVITY (CLAUDE OPUS)

Build the Notion Integration

Paste the following into Antigravity:

PROMPT FOR ANTIGRAVITY — NOTION INTEGRATION:

Build a Notion integration module for "Apollo Capture." This module takes structured property data (JSON) and creates a complete property documentation setup in Notion.

TECH STACK:
- Use the @notionhq/client npm package (official Notion SDK)
- This is a module that plugs into the existing backend from the previous step

NOTION WORKSPACE SETUP:
The integration should create or update the following structure:

DATABASE: "Apollo Properties"
This is the main database. If it doesn't exist, create it. If it exists, add to it.

Database properties:
- Property Name (title)
- Address (rich_text)
- Status (select: "Onboarding", "Active", "Inactive")
- Total Rooms (number)
- Bedrooms (number)
- Bathrooms (number)
- Property Type (select: "House", "Apartment", "Condo", "Townhouse", "Other")
- Has Outdoor Space (checkbox)
- Onboarding Date (date)
- Capture Video (url — link to full video)
- General Notes (rich_text)

PAGE CONTENT: For each property page
When a new property is created in the database, populate the page content with:

1. HEADER SECTION
   - H1: Property Name
   - Property address as subtitle text
   - Callout block with: " 📋 Onboarded on [date] via Apollo Capture"

2. PROPERTY OVERVIEW SECTION
   - H2: "Property Overview"
   - Table of basic info: bedrooms, bathrooms, property type, outdoor space
   - Any general notes

3. ACCESS INFORMATION SECTION
   - H2: " 🔑 Access Information"

- Callout block (important/warning style):
  - WiFi: [name] / [password]
  - Lockbox Code: [code]
  - Gate Code: [code]
  - Parking: [instructions]
  - Any other access info

4. ROOMS SECTION — For EACH room:
- H2: " 🚪 [Room Name]" (e.g., " 🚪 Primary Bedroom")
- Toggle block: "Inventory ([item count] items)"
  - Inside toggle: bullet list of every inventory item with quantity and notes
  - Format: "• [quantity]x [item] — [notes]" (e.g., "• 2x Pillows — standard size, white covers")
- Toggle block: "Room Features"
  - Bullet list of features
- Toggle block: "Notes & Quirks"
  - Bullet list of any quirks, cleaning notes, or special instructions
- Photos: Embed each photo associated with this room as an image block
  - Add a caption under each photo with the room name and timestamp

5. SYSTEMS & UTILITIES SECTION
- H2: " ⚙️ Systems & Utilities"
- Bullet list of all system information (HVAC, water heater, breaker box, etc.)

6. FULL TRANSCRIPT SECTION
- H2: " 📝 Full Walkthrough Transcript"
- Toggle block: "Click to expand full transcript"
  - Inside: the complete transcript text, broken into paragraphs by room

NOTION API IMPLEMENTATION:

```
function syncToNotion(captureResult, notionApiKey, databaseId) {
  // 1. Initialize Notion client
  // 2. Check if property already exists in database (by name + address)
  //    - If exists: update the existing page
  //    - If new: create new page
  // 3. Create the page with all properties
  // 4. Add all content blocks (overview, access, rooms, etc.)
  // 5. Upload and embed photos
  // 6. Return the Notion page URL
}
```

IMPORTANT NOTES:
- Notion API has a limit of 100 blocks per request — batch block creation if needed
- For photos: upload to your storage first, then embed using external image URLs in Notion

- The Notion API can't create toggle blocks directly in some versions — use the "toggle" block type or fall back to bulleted lists with indentation
- Handle rate limiting (Notion API: 3 requests/second)
- If the database doesn't exist yet, provide instructions for the user to:
  1. Create a new integration at https://www.notion.so/my-integrations
  2. Share a page/database with the integration
  3. Provide the database ID

INPUT: The structured JSON output from Step 3's processing pipeline.
OUTPUT: Notion page URL + confirmation of created/updated content.

ENVIRONMENT VARIABLES:
- NOTION_API_KEY: The Notion integration token
- NOTION_DATABASE_ID: The ID of the "Apollo Properties" database

Build the complete module with all Notion API calls, block creation, photo embedding, error handling, and batch management.

WHAT YOU SHOULD HAVE AFTER STEP 4:

A Notion integration that:

- ✅ Creates a new property page in your Notion database
- ✅ Populates it with all room data, inventory, photos, access info
- ✅ Uses toggle blocks for organized, collapsible sections
- ✅ Embeds all captured photos in the correct room sections
- ✅ Includes the full transcript
- ✅ Returns a Notion URL you can click to view the result

STEP 5: GOOGLE ANTIGRAVITY

Wire Frontend to Backend

Paste the following into Antigravity:

PROMPT FOR ANTIGRAVITY — CONNECT EVERYTHING:

I have three pieces of an app called "Apollo Capture" that need to be connected:

1. A mobile web app (frontend) that records video, captures photos, and generates transcripts
2. A backend API that processes uploads with AI (Claude) to extract structured property data
3. A Notion integration module that creates property pages from the structured data

Wire them together:

FRONTEND CHANGES NEEDED:
1. Replace the IndexedDB local storage with actual API upload
2. When the user taps "Upload & Process":
   a. Package the video, photos, transcript, and metadata into a multipart/form-data request
   b. POST to the backend: POST /api/capture/upload
   c. Show upload progress (track the XHR/fetch upload progress event)
   d. After upload completes, poll GET /api/capture/:id/status every 3 seconds
   e. Update the UI progress screen with each status change:
      - "Uploading..." → "Transcribing..." → "Analyzing rooms..." → "Creating Notion page..." → "Complete!"
   f. When status is "complete", show the result with a "View in Notion" button linking to the Notion page URL
   g. Also show a summary of what was found: rooms, total inventory items, photos processed

3. Add a settings screen (accessible from home screen, gear icon in top-right):
   - Backend API URL input (default to production URL)
   - Notion database ID input
   - "Test Connection" button that pings the backend health endpoint
   - Save settings to localStorage

BACKEND CHANGES NEEDED:
1. After the AI processing pipeline completes (Step 3 result), automatically trigger the Notion sync (Step 4)
2. Add the Notion page URL to the capture result

3. Add a health check endpoint: GET /api/health
4. Add CORS headers allowing requests from any origin (for now)
5. Make sure the status endpoint accurately reflects which stage the pipeline is in

DEPLOYMENT:
- Frontend: deploy as a static site (Vercel, Netlify, or Cloudflare Pages)
- Backend: deploy to Railway or Render
- Provide the deployment configuration files needed (vercel.json, Dockerfile, railway.toml, etc.)

ENVIRONMENT SETUP:
Provide a clear .env.example file with all needed variables:
- ANTHROPIC_API_KEY=
- NOTION_API_KEY=
- NOTION_DATABASE_ID=
- STORAGE_TYPE=s3|r2|gcs
- STORAGE_BUCKET=
- STORAGE_ACCESS_KEY=
- STORAGE_SECRET_KEY=
- STORAGE_REGION=
- FRONTEND_URL=
- PORT=3000

Build the connection layer, update both frontend and backend, and provide deployment instructions.

WHAT YOU SHOULD HAVE AFTER STEP 5:

A fully connected application where:

- ✅ User records walkthrough on phone
- ✅ Taps upload → data goes to backend
- ✅ Backend processes with AI → extracts structured data
- ✅ Automatically creates Notion page with everything
- ✅ User gets a link to view the Notion page
- ✅ Deployed and accessible from any phone via URL

STEP 6: TEST ON A REAL PROPERTY

Testing Checklist

Pre-Test Setup:

- [ ] Backend is deployed and running
- [ ] Frontend is deployed and accessible on phone
- [ ] Notion integration is connected (API key, database ID configured)
- [ ] Settings screen shows green "Connected" status

Test at an Apollo Property:

1. **Open the app on your phone**
   - [ ] App loads correctly
   - [ ] Can enter property name and address
2. **Start recording**
   - [ ] Camera activates (rear camera)
   - [ ] Transcription starts (you see text appearing at bottom)
   - [ ] Timer is running
3. **Walk through the property**
   - [ ] Say "This is the living room" → room detected toast appears
   - [ ] Say "mark photo" → hear shutter click, see thumbnail
   - [ ] Verify thumbnail shows the right thing
   - [ ] Walk to kitchen, say "Now we're in the kitchen"
   - [ ] Say "mark photo" a few times for different items
   - [ ] Narrate: "There are 4 bar stools at the island"
   - [ ] Continue through bedrooms, bathrooms, etc.
   - [ ] Mention WiFi password, lockbox code if applicable
   - [ ] Point out any quirks: "The shower handle needs to be pulled out first"
4. **Stop and review**
   - [ ] Recording stops cleanly
   - [ ] Summary shows correct photo count
   - [ ] Summary shows detected rooms
   - [ ] Photos are viewable and correct

- [ ] Can delete a bad photo

5. **Upload and process**
   - [ ] Upload progress shows correctly
   - [ ] Processing stages update in real-time
   - [ ] No errors or timeouts
   - [ ] "Complete" screen shows with Notion link
6. **Check Notion**
   - [ ] Property page exists in database
   - [ ] All rooms are listed with correct names
   - [ ] Inventory is accurate (or close — note any misses)
   - [ ] Photos are embedded in correct room sections
   - [ ] Access info is captured
   - [ ] Transcript is included

Common Issues to Watch For:

Issue	Likely Cause	Fix
Transcription is garbage	Bad mic, background noise	Use a lapel mic, or speak louder/closer to phone
"Mark photo" not detected	Speech API missed it	Tap the manual photo button as backup
Wrong room assignment	Didn't clearly state room name	Be more explicit: "Now entering bedroom 2"
Photos are blurry	Phone moved during capture	Hold steady for a beat when saying "mark photo"
Upload timeout	Video file too large	Reduce recording quality in settings, or compress before upload
Notion page is messy	AI misinterpreted transcript	Review and edit in Notion, note patterns for prompt improvement
Room missed entirely	Speaker didn't name the room	App should prompt if no room name detected in X seconds (v2 feature)

DATA MODEL REFERENCE

Capture Session Object

json

```
{
 "captureId": "uuid-v4",
 "propertyName": "Apollo Beach House",
 "propertyAddress": "123 Ocean Drive, Malibu, CA 90265",
 "captureDate": "2026-02-14T10:30:00Z",
 "recordingDuration": 847,
 "status": "complete",

 "propertyOverview": {
  "totalRooms": 8,
  "propertyType": "house",
  "estimatedBedrooms": 3,
  "estimatedBathrooms": 2,
  "hasOutdoorSpace": true,
  "generalNotes": "Two-story beach house, recently renovated kitchen"
 },

 "rooms": [
  {
   "roomId": "room-1",
   "roomName": "Primary Bedroom",
   "roomType": "bedroom",
   "startTimestamp": 45,
   "endTimestamp": 128,
   "inventory": [
    { "item": "King bed", "quantity": 1, "notes": "Tempur-Pedic mattress", "condition": "good" },
    { "item": "Pillow", "quantity": 4, "notes": "2 firm, 2 soft", "condition": "good" },
    { "item": "Nightstand", "quantity": 2, "notes": "with USB charging ports", "condition": "good" },
    { "item": "Lamp", "quantity": 2, "notes": "touch-activated", "condition": "good" },
    { "item": "Dresser", "quantity": 1, "notes": "6-drawer, walnut finish", "condition": "good" },
    { "item": "Ironing board", "quantity": 1, "notes": "in closet", "condition": "fair" },
    { "item": "Iron", "quantity": 1, "notes": "in closet with ironing board", "condition": "good" },
    { "item": "Hangers", "quantity": 20, "notes": "wooden, in closet", "condition": "good" }
   ],
   "features": ["ceiling fan", "en-suite bathroom", "walk-in closet", "ocean view"],
   "quirksAndNotes": ["Closet light switch is inside on the left wall"],
   "accessInfo": [],
   "cleaningNotes": ["White duvet — check for stains after each guest"],
   "photos": [
    { "photoUrl": "https://storage.example.com/captures/abc/photo-001.jpg", "timestamp": 52 },
    { "photoUrl": "https://storage.example.com/captures/abc/photo-002.jpg", "timestamp": 78 },
    { "photoUrl": "https://storage.example.com/captures/abc/photo-003.jpg", "timestamp": 95 }
   ]
```

```
    }
  ],

  "propertyAccess": {
    "wifiName": "ApolloBeach_5G",
    "wifiPassword": "sunset2024!",
    "lockboxCode": "4589",
    "parkingInstructions": "Two-car garage, opener in kitchen drawer by fridge",
    "gateCode": null,
    "otherAccess": ["Pool equipment shed key is on the keyring in the lockbox"]
  },

  "systemsAndUtilities": {
    "hvac": "Nest thermostat in hallway, set to 72°F for guests",
    "waterHeater": "Tankless, in garage, no adjustment needed",
    "breakerBox": "In garage, far left wall behind the surfboards",
    "waterShutoff": "Front yard, left side of house, green cover in ground",
    "trashDay": "Wednesday — bins go out Tuesday night",
    "otherSystems": [
      "Pool pump runs automatically 8am-2pm",
      "Sprinklers run Mon/Wed/Fri at 6am"
    ]
  },

  "fullTranscript": "Okay so we're at the Apollo Beach house...",
  "rawData": {
    "videoUrl": "https://storage.example.com/captures/abc/walkthrough.webm",
    "transcriptUrl": "https://storage.example.com/captures/abc/transcript.json"
  }
}
```

FUTURE ENHANCEMENTS (V2+)

For reference only — don't build these now:

1. **AI Photo Verification** — AI checks photos against narration ("you said 5 pillows, I count 4")
2. **Room Prompting** — App asks "which room are you in?" if no room detected in 60 seconds

3. **Recurring Inspections** — Same walkthrough flow but compares against baseline inventory
4. **Damage Documentation** — Flag and photograph damage with severity rating
5. **Team Management** — Assign onboarding tasks to team members, track completion
6. **Multi-language Support** — Transcription in Spanish (many cleaners/onboarders are bilingual)
7. **Offline Mode** — Full offline recording with sync when back online
8. **Listing Description Generator** — Use the inventory + photos to auto-generate Airbnb listings
9. **Cleaning Checklist Generator** — Auto-create room-by-room cleaning checklists from inventory
10. **Native App** — iOS/Android app for better camera control and offline support
11. **Video Clips Per Room** — Auto-cut video into per-room clips using FFmpeg
12. **Property Comparison** — Compare two captures of the same property over time (before/after guest stay)
13. **Integration with PMS** — Sync property data with Guesty, Hostaway, etc.
14. **Voice Commands Beyond Mark Photo** — "flag issue", "urgent note", "skip room", etc.
15. **AR Measurements** — Use phone AR to estimate room dimensions during walkthrough

---

QUICK REFERENCE: WHAT GOES WHERE

Step	Tool	What You're Building	Time Estimate
1	Google AI Studio	Core recording web app (camera, mic, speech-to-text, photo capture)	2-4 hours
2	Google Stitch	Professional UI redesign	1-2 hours
3	Antigravity + Opus	Backend AI pipeline (transcript → structured data)	3-5 hours
4	Antigravity + Opus	Notion integration (structured data → Notion pages)	2-3 hours
5	Antigravity	Wire frontend ↔ backend, deploy	2-3 hours
6	Your legs	Walk through a real property and test	1-2 hours

Total estimated build time: 11-19 hours across 2-3 days

Built for Apollo Property Management LLC. Created February 2026.