Here is a comprehensive Markdown summary of your project state. You can save this as ReflectionsMatch_Status_Jan08.md to capture exactly where we left off.

It includes the **verified working code** for both the Extension and the Dashboard, along with the critical configuration fixes we discovered (CORS, Keys, and Model Versions).

Reflections Match - Project Status

Date: January 8, 2026

Status: Functional Prototype (Alpha)

1. Executive Summary

Reflections Match is a full-stack system allowing users to capture "reflections" (screenshots) from any webpage and automatically organize them with AI-generated summaries and tags.

- **Chrome Extension:** Captures visible tab, supports cropping, uploads image to Firebase Storage, and writes metadata to Firestore.
- **Web Dashboard:** Real-time feed of captures. Automatically detects new images, sends them to Gemini AI for analysis, and updates the card with a summary and tags.

---

2. Architecture & Flow

1. **User** clicks Extension Icon → Captures Screenshot.
2. **Extension** uploads image to Firebase Storage (/captures).
3. **Extension** writes document to Firestore (/reflections) with imageUrl and timestamp.
4. **Dashboard** listens to Firestore in real-time.
5. **Dashboard** detects a record *without* an aiSummary.
6. **Dashboard** sends image URL to **Gemini 1.5 Flash**.
7. **Gemini** returns JSON (Summary + Tags).
8. **Dashboard** updates Firestore, and the UI updates instantly.

---

3. The "Code Vault" (Current Working Versions)

A. Chrome Extension (reflectionsmatch-extension/src/App.tsx)

Features: Capture, Crop, Upload, Database Write, Auto-Reset.

TypeScript

```
import { useState, useRef } from 'react'
import ReactCrop, { type Crop, type PixelCrop } from 'react-image-crop'
import 'react-image-crop/dist/ReactCrop.css'
import { ref, uploadBytes, getDownloadURL } from 'firebase/storage'
import { collection, addDoc, serverTimestamp } from 'firebase/firestore'
import { storage, db } from './firebaseConfig'
import { canvasPreview } from './canvasPreview'
import './App.css'

function App() {
  const [screenshotUrl, setScreenshotUrl] = useState<string | null>(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [crop, setCrop] = useState<Crop>();
  const [completedCrop, setCompletedCrop] = useState<PixelCrop>();
  const imgRef = useRef<HTMLImageElement>(null);
  const [uploading, setUploading] = useState(false);
  const [uploadSuccess, setUploadSuccess] = useState(false);

  const captureTab = async () => {
    setLoading(true);
    setError(null);
    setUploadSuccess(false);
    try {
      chrome.tabs.captureVisibleTab({ format: 'png' }, (dataUrl: string) => {
        if (chrome.runtime.lastError) {
          setError(chrome.runtime.lastError.message || 'Failed to capture tab');
          setLoading(false);
          return;
        }
        if (dataUrl) setScreenshotUrl(dataUrl);
        else setError('No image data received');
        setLoading(false);
      });
    } catch (err: any) {
      setError(err.message || 'Unexpected error');
      setLoading(false);
    }
  };

  const clearScreenshot = () => {
    setScreenshotUrl(null);
    setCrop(undefined);
    setCompletedCrop(undefined);
    setError(null);
    setUploadSuccess(false);
  };

  const handleUpload = async () => {
    if (!screenshotUrl) return;
    setUploading(true);
    setError(null);

    try {
      let blob: Blob;
      if (completedCrop && completedCrop.width > 0 && completedCrop.height > 0 && imgRef.current) {
        blob = await canvasPreview(imgRef.current, completedCrop);
      } else {
        const response = await fetch(screenshotUrl);
        blob = await response.blob();
      }

      // 1. Upload to Storage
      const filename = `captures/${Date.now()}.png`;
      const storageRef = ref(storage, filename);
      const snapshot = await uploadBytes(storageRef, blob);
      const downloadUrl = await getDownloadURL(snapshot.ref);

      // 2. Write to Firestore
      await addDoc(collection(db, "reflections"), {
        imageUrl: downloadUrl,
        timestamp: serverTimestamp(),
        notes: "",
        userId: "test-user"
      });

      // 3. Reset and Close
      setUploadSuccess(true);
      setTimeout(() => {
        clearScreenshot(); // Fixes "Ghost Image" bug
        window.close();
      }, 2000);

    } catch (err: any) {
      console.error("Upload failed:", err);
      setError('Upload failed: ' + err.message);
```

```
    } finally {
      setUploading(false);
    }
  };

  // ... (JSX Return logic follows standard pattern) ...
  return ( /* UI Code */ );
}

export default App
```

B. Web Dashboard (reflectionsmatch-web/src/App.jsx)

Features: Real-time Feed, Auto-AI Analysis, Gemini Integration.

Critical Configuration:

- **Model:** Must use "gemini-1.5-flash-001" (Version suffix is required).
- **Key:** Must use the API Key restricted to "Reflections Match" project (not the generic Browser key).

JavaScript

```
import React, { useEffect, useState, useRef } from 'react';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import { collection, query, orderBy, onSnapshot, doc, updateDoc } from 'firebase/firestore';
import { db } from './firebase';
import { GoogleGenerativeAI } from "@google/generative-ai";
import ReflectionCard from './components/ReflectionCard';

const API_KEY = 'YOUR_REFLECTIONS_MATCH_PROJECT_KEY';
const genAI = new GoogleGenerativeAI(API_KEY);

const Dashboard = () => {
  const [reflections, setReflections] = useState([]);
  const [loading, setLoading] = useState(true);
  const [analyzingIds, setAnalyzingIds] = useState(new Set());
  const processingIds = useRef(new Set());

  // 1. Fetch Data
  useEffect(() => {
    const q = query(collection(db, 'reflections'), orderBy('timestamp', 'desc'));
    const unsubscribe = onSnapshot(q, (snapshot) => {
      const data = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
      setReflections(data);
      setLoading(false);
    });
    return () => unsubscribe();
  }, []);

  // 2. AI Logic
  const analyzeReflection = async (id, imageUrl) => {
    if (processingIds.current.has(id)) return;
    processingIds.current.add(id);
    setAnalyzingIds(prev => new Set(prev).add(id));

    try {
      // CRITICAL: Use the specific version -001
      const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash-001" });
      const imagePart = await urlToGenerativePart(imageUrl);
      const prompt = 'Analyze this image. Return a valid JSON object with a "summary" (max 2
sentences) and "tags" (array of 3 keywords). Do not include markdown code block syntax around the
JSON.';
```

```
            const result = await model.generateContent([prompt, imagePart]);
            const response = await result.response;
            const cleanText = response.text().replace(/```json/g, '').replace(/```/g, '').trim();
            const analysis = JSON.parse(cleanText);

            await updateDoc(doc(db, "reflections", id), {
                aiSummary: analysis.summary,
                tags: analysis.tags
            });
        } catch (error) {
            console.error("AI Analysis Failed:", error);
        } finally {
            processingIds.current.delete(id);
            setAnalyzingIds(prev => {
                const next = new Set(prev);
                next.delete(id);
                return next;
            });
        }
    };

    // 3. Auto-Trigger
    useEffect(() => {
        if (loading) return;
        reflections.forEach(reflection => {
            if (!reflection.aiSummary && (reflection.imageUrl || reflection.url) &&
!processingIds.current.has(reflection.id)) {
                analyzeReflection(reflection.id, reflection.imageUrl || reflection.url);
            }
```

```
        });
    }, [reflections, loading]);

    // ... (JSX for Dashboard Grid) ...
};
```

---

4. Configuration & Fixes Log

1. CORS Policy (The "Final Boss")

Issue: Dashboard was blocked from downloading images from Firebase Storage to send to Gemini.

Fix: Applied CORS configuration via Google Cloud Shell.

Bash

```
echo '[{"origin": ["*"],"method": ["GET"], "maxAgeSeconds": 3600}]' > cors.json
gsutil cors set cors.json gs://reflections-match.firebasestorage.app
```

2. The 404 Model Error

Issue: gemini-1.5-flash returned 404 Errors.

Cause: Generic aliases sometimes fail depending on Project region/permissions.

Fix: Explicitly used version gemini-1.5-flash-001.

3. API Key Mismatch

Issue: Using the default "Browser Key" created by Firebase caused 404s.

Fix: Created/Used a new API Key specifically within the "Reflections Match" project in Google Cloud Console, with access to the Generative Language API.

---

5. Next Steps

- **User Auth:** Implement proper Google Login (Path B) so userId isn't hardcoded to "test-user".
- **Extension Auth:** Connect the extension to the same Auth system.
- **Search:** Add a search bar to filter reflections by the AI-generated tags.