API SPECIFICATION

🔐 What Is Google Safe Browsing?

Google Safe Browsing is a security service by Google that maintains constantly updated lists of:

- ⚠️ **Malware-infected sites**
- 🎣 **Phishing sites**
- 💣 **Unwanted software distributors**
- 🧬 **Social engineering attacks**
- 🎭 **Sites that use deceptive ads or behaviors**

Google uses this service to protect users in Chrome, Android, Gmail, and other products — but developers can also access it.

⚙️ What Does It Actually Do?

When you use Safe Browsing in your project, you can **check whether a URL is dangerous** before letting your users access it.

It works like this:

1. Your app queries Google Safe Browsing with a URL.
2. Google checks if the URL matches known bad patterns or domains.
3. You get a response like:
    - `MALWARE`
    - `PHISHING`
    - `SOCIAL_ENGINEERING`
    - or `OK` (safe)

You can then decide:

- 🚫 Block the URL
- ⚠️ Warn the user
- ✅ Let them proceed

🧰 What You Can Do with Safe Browsing in Your Project

Here's a list of possible features and use cases if you integrate Safe Browsing:

✅ 1. Pre-scan URLs Submitted by Users

- Protect users from submitting malicious links in forms, chat, or forums.

✅ 2. QR Code Safety

- If you're decoding QR codes in your app, you can **run the extracted URL through Safe Browsing** before opening it.

✅ 3. Browser Extensions or URL Checkers

- Build a browser extension or tool that warns users about bad sites in real-time.

✅ 4. Parental Control or Content Filter Tools

- Automatically **block access to harmful or unsafe websites**.

✅ 5. Email Filtering

- Scan links in incoming emails/messages before displaying them or enabling clicks.

📡 How It Works (Tech Side)

You use the **Google Safe Browsing Lookup API** or the more efficient **Update API** (recommended for large-scale apps).

API Flow (Simplified):

1. **Send a hash of the URL prefix** (to preserve privacy).
2. **Google checks against its threat lists.**
3. **If there's a match**, you get back a threat type (like `MALWARE`).
4. **Then you choose how to respond in your app.**

📃 Limitations / Important Notes

- **Rate-limited**: Free tier has usage limits; you may need a key and quota for commercial projects.
- **Privacy-preserving**: You don’t send full URLs unless needed.
- It doesn't **block sites**, it just **informs you** about known threats — what you do with the result is up to you.

👉 Example Use Case

Let’s say you're building a **QR code scanner app**.
After decoding a QR, you:

1. Extract the URL.
2. Send it to Google Safe Browsing API.
3. If flagged: Warn the user with a red alert.
4. If safe: Let them open the link.

✅ **Free Usage Limits (Per Month):**

- **URL Lookups (check if a URL is safe):** 100,000 requests
- **URL Submissions (report unsafe URLs):** 100 submissions
- **Threat List Updates (for local storage):** Unlimited

🕒 **Request Frequency:**

- **URL Lookups:** No strict rate limit, but requests should be reasonable (e.g., a few per second)
- **Update Requests:** Google provides a `minimumWaitDuration` — usually ~30 minutes — to sync your app's local threat list
- **Back-off Mode:** Triggered after multiple failed requests → exponential wait time before retrying

📜 **Usage Type:**

- **Free version is for personal or non-commercial use only**
- For commercial/business use → use the **Web Risk API** (paid)

When you use the **Google Safe Browsing API** to check a URL, the **output is a JSON response** that tells you whether the URL matches any known threat types.

✅ If the URL is safe (no threat):

```
json
CopyEdit
{}
```

— An **empty JSON** means **no threat found** (the URL is clean).

❌ If the URL is unsafe (threat found):

```
json
CopyEdit
{
  "matches": [
    {
      "threatType": "MALWARE",
      "platformType": "WINDOWS",
      "threatEntryType": "URL",
      "threat": {
        "url": "http://example.com/bad"
      }
    }
  ]
}
```

🔑 Key Fields in the Output:

Field	Meaning
`threatType`	Type of threat: `MALWARE`, `PHISHING`, `SOCIAL_ENGINEERING`, `UNWANTED_SOFTWARE`, etc.
`platformType`	Affected platform (e.g., `WINDOWS`, `ANDROID`, `ANY_PLATFORM`)
`threatEntryType`	Usually `URL` (can also be `EXECUTABLE`)
`threat.url`	The URL that was matched (echoed back)

Here's a **clear step-by-step guide** to using the **Google Safe Browsing API (v4)** to check if a URL is safe.

🛠️ Step-by-Step: Using Google Safe Browsing API

✅ Step 1: Get API Access

1. Go to Google Cloud Console.
2. Create a **new project** (or select an existing one).
3. Go to **APIs & Services > Library**.
4. Search for **"Safe Browsing API"** and **enable it**.
5. Go to **APIs & Services > Credentials**:
    - Click **"Create Credentials" → API key**.
    - Copy your new **API key**.

✅ Step 2: Understand the Endpoint

Use the **Threat Matches endpoint**:

```
bash
CopyEdit
POST
https://safebrowsing.googleapis.com/v4/threatMatches:find?key=YOUR_API_KEY
```

✅ Step 3: Prepare the Request

This is a **POST request with JSON body**, like this:

```
json
CopyEdit
{
  "client": {
    "clientId": "yourcompanyname",
    "clientVersion": "1.0"
  },
  "threatInfo": {
    "threatTypes": ["MALWARE", "SOCIAL_ENGINEERING", "UNWANTED_SOFTWARE",
"POTENTIALLY_HARMFUL_APPLICATION"],
    "platformTypes": ["ANY_PLATFORM"],
    "threatEntryTypes": ["URL"],
    "threatEntries": [
      {"url": "http://example.com/suspicious"}
    ]
  }
}
```

✅ Step 4: Send the Request

Use your favorite HTTP tool or language:

🔧 Example in Python (using `requests`):

```
python
CopyEdit
import requests

api_key = "YOUR_API_KEY"
url = "http://example.com/suspicious"
endpoint =
f"https://safebrowsing.googleapis.com/v4/threatMatches:find?key={api_key}"

payload = {
    "client": {
        "clientId": "yourproject",
        "clientVersion": "1.0"
    },
    "threatInfo": {
        "threatTypes": ["MALWARE", "SOCIAL_ENGINEERING",
"UNWANTED_SOFTWARE"],
        "platformTypes": ["ANY_PLATFORM"],
        "threatEntryTypes": ["URL"],
        "threatEntries": [{"url": url}]
    }
}

response = requests.post(endpoint, json=payload)
print(response.json())
```

✅ Step 5: Handle the Response

- If the response is `{}`, the URL is **safe**.
- If there's a `"matches"` array, the URL is **flagged** — check `threatType` to know why.

END