Basics

When using Puppeteer in an Electron + Vite project, you may encounter issues after building the app, especially with the latest versions of Puppeteer. Here's why:

1. Puppeteer Uses Chromium, but Electron Has Its Chromium

Puppeteer downloads its version of Chromium, but Electron also has its own embedded version.

When you build the Electron app, Puppeteer's Chromium may not be included in the final package, causing errors.

2. Native Dependencies and Build Issues

Puppeteer is meant for Node.js environments, while Electron runs in both Node.js and a browser-like renderer process.

Some Puppeteer modules might fail in the renderer process.

3. After Build: Possible Errors You May Face

If Puppeteer is not properly configured, you might see errors like:

:x: Error: Could not find expected browser (chrome) locally.

:x: Protocol error (Page.navigate): Target closed.

:x: Puppeteer is not usable in the packaged app.

4. How to Fix Puppeteer in Electron Build?

To prevent errors after building:

:white\_check\_mark: 1. Use Electron’s Built-in Chromium Instead of Puppeteer’s

Instead of using Puppeteer's Chromium, tell Puppeteer to use Electron's browser:

ts

import puppeteer from 'puppeteer-core'; // Use core version (no bundled Chromium)

import { app } from 'electron';

const browser = await puppeteer.launch({

executablePath: app.getPath('exe'), // Uses Electron's Chromium

headless: true,

});

:white\_check\_mark:

2. Ensure Puppeteer's Dependencies Are Included in the Build

If you still want to use Puppeteer’s Chromium, ensure it is included in your build by adding it to package.json:

JSON

"build": {

"files": [

"node\_modules/puppeteer",

"node\_modules/puppeteer/.local-chromium"

]

}

:white\_check\_mark:

3. Use Electron Forge or Electron Builder Configuration

If you're using electron-builder, add this to package.json:

JSON

"build": {

"extraResources": [

{

"from": "node\_modules/puppeteer/.local-chromium",

"to": "app.asar.unpacked/node\_modules/puppeteer/.local-chromium"

}

]

}

Final Answer:

:rocket: Yes, the latest Puppeteer version may cause errors after the Electron build unless you configure it properly.

To avoid issues, use Puppeteer-core and Electron's Chromium or ensure Puppeteer's Chromium is included in the build.

Would you like help debugging your specific error? :blush:

Proxy

**Proxy Configuration in Puppeteer + Electron**

## **Free Proxy Options**

* **Command-line Arguments**: Set proxy via --proxy-server flag
* **Page-level Configuration**: Configure using Page.authenticate()
* **Public Proxies**: Easy to find but unreliable, slow, often blacklisted
* **Rotating Free Proxies**: Better but requires maintenance

## **Paid Proxy Options**

* **Residential Proxies**: Look like real users, less likely to be blocked
* **Datacenter Proxies**: Faster but easier to detect
* **Dedicated Proxies**: Better performance and reliability
* **Premium Services**: BrightData, Oxylabs, and SmartProxy offer specialized solutions

## **Implementation Example**

// Puppeteer with proxy

const browser = await puppeteer.launch({

args: [

`--proxy-server=${proxyServer}:${proxyPort}`

]

});

// With authentication

const page = await browser.newPage();

await page.authenticate({

username: proxyUsername,

password: proxyPassword

});  
  
**Code Example:**  
const puppeteer = require("puppeteer");

async function run() {

const browser = await puppeteer.launch({

headless: false,

args: ["--proxy-server=200.73.128.156:3128"],

});

const page = await browser.newPage();

const pageUrl = "https://whatismyipaddress.com/";

await page.goto(pageUrl);

}

run();

## **Electron Integration**

// In Electron main process

app.commandLine.appendSwitch('proxy-server', `${proxyHost}:${proxyPort}`);

app.commandLine.appendSwitch('proxy-bypass-list', '<local>');

// With authentication

session.defaultSession.setProxy({

proxyRules: `http=${proxyHost}:${proxyPort}`,

proxyBypassRules: '<local>'

}).then(() => {

// Handle proxy authentication

});

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
**Proxy with webshare:  
  
 Why Use proxy-chain?**

1. Converts authenticated proxies into anonymous ones
   * Puppeteer does not support http://username:password@proxy-ip:port directly in --proxy-server.
   * proxy-chain converts it into a usable format.
2. Prevents proxy detection
   * Some websites block known proxy IPs.
   * proxy-chain makes them appear like normal requests.
3. Auto-authentication
   * Instead of manually calling page.authenticate(), proxy-chain handles proxy login automatically.

**Code Using Webshare + ProxyChain + Puppeteer:**

**const puppeteer = require("puppeteer");**

**const proxyChain = require("proxy-chain");**

**(async () => {**

***// Your Webshare proxy details***

**const oldProxyUrl = "http://ghvbaygi:jr0hg5dw32lt@38.154.227.167:5868";**

***// Convert to an anonymized proxy (avoids detection issues)***

**const newProxyUrl = await proxyChain.anonymizeProxy(oldProxyUrl);**

**const browser = await puppeteer.launch({**

**headless: false,**

**args: [`--proxy-server=${newProxyUrl}`], *// Set the proxy server***

**});**

**const page = await browser.newPage();**

**try {**

**await page.goto("https://whatismyipaddress.com/", { waitUntil: "domcontentloaded" });**

**console.log("Page loaded successfully!");**

**} catch (error) {**

**console.error("Failed to load page:", error);**

**}**

***//await browser.close();***

**await proxyChain.closeAnonymizedProxy(newProxyUrl);**

**})();**

reCAPTCHA

### **Handling reCAPTCHA in Puppeteer**

If you're automating a website and it presents a reCAPTCHA, you generally have a few options:

#### **1. Manually Solve the reCAPTCHA**

You can pause execution and wait for the user to solve it manually.

await page.waitForFunction(() => {

return document.querySelector('.g-recaptcha-response')?.value.length > 0;

});

This waits until the .g-recaptcha-response field is filled, which means the CAPTCHA is solved.

#### **2. Using waitForSelector to Detect reCAPTCHA**

You can wait for reCAPTCHA to appear and take action.

await page.waitForSelector('.g-recaptcha', { timeout: 10000 });

console.log("reCAPTCHA detected. Waiting for manual solving...");

await page.waitForFunction(() => {

return document.querySelector('.g-recaptcha-response')?.value.length > 0;

});

console.log("reCAPTCHA solved!");

#### **3. Using Third-Party Services to Solve reCAPTCHA**

If you need to automate solving, you can use services like 2Captcha or Anti-Captcha. For example, using puppeteer-extra-plugin-recaptcha:

const puppeteer = require('puppeteer-extra');

const RecaptchaPlugin = require('puppeteer-extra-plugin-recaptcha');

// Add the reCAPTCHA plugin

puppeteer.use(

RecaptchaPlugin({

provider: { id: '2captcha', token: 'YOUR\_2CAPTCHA\_API\_KEY' },

visualFeedback: true, // Shows visual progress

})

);

(async () => {

const browser = await puppeteer.launch({ headless: false });

const page = await browser.newPage();

await page.goto('https://example.com'); // Replace with your target page

// Solve the reCAPTCHA

await page.solveRecaptchas();

console.log("reCAPTCHA solved!");

})();

#### **4. Bypassing Cloudflare Turnstile or Other Challenges**

Some sites use Cloudflare's anti-bot measures. You may need Puppeteer Stealth:

const puppeteer = require('puppeteer-extra');

const StealthPlugin = require('puppeteer-extra-plugin-stealth');

puppeteer.use(StealthPlugin());

(async () => {

const browser = await puppeteer.launch({ headless: false });

const page = await browser.newPage();

await page.goto('https://example.com');

await page.waitForTimeout(5000); // Wait for Cloudflare to process

})();