ChatGPT

Guia Prático: Usando o endpoint /function do Browserless no n8n

1. Estrutura Correta do Código no Corpo da Requisição

O endpoint `/function` do Browserless espera receber um **código JavaScript em formato de módulo ECMAScript** (ESM) com uma função assíncrona *exportada como default*. Em outras palavras, o conteúdo do corpo da requisição deve ser um script iniciando com `export default async function` e **recebendo um objeto de contexto**. Por exemplo, a função recebe normalmente um objeto com propriedades como `{ page, context }`, onde `page` é o objeto Puppeteer (página do navegador) e `context` são dados extras opcionais fornecidos por você [1].

Dentro dessa função, você pode usar comandos do Puppeteer (como `page.goto`, `page.click`, etc.) e outras APIs do navegador. **Ao final, a função deve retornar um objeto contendo duas propriedades obrigatórias:** `data` **e** `type` [2]. - `data`: os dados de resultado que você quer receber (pode ser um objeto JSON, texto simples, Buffer binário etc.).
- `type`: uma string indicando o MIME type (Content-Type) apropriado para os dados retornados em `data`.

Conforme a documentação: *"Functions should return an object with two properties:* `data` *and* `type`. `data` *can be whatever you'd like (Buffer, JSON, or plain text) and* `type` *is a string describing the Content-Type of* `data` *"* [2]. Isso significa que o Browserless irá ler esses campos e definir a resposta HTTP de acordo com eles. Por exemplo, se `type` for `"application/json"`, o retorno será um JSON; se for `"text/html"`, retornará HTML; se for um PDF, você usaria `"application/pdf"`, etc.

Exemplo de código válido: a seguir um exemplo básico (adaptado da documentação) de função enviada no corpo da requisição:

```
export default async function ({ page }) {
  // Navega para uma página de exemplo
  await page.goto("https://example.com/");
  const titulo = await page.title();  // obtém o título da página

  // Gera cinco números aleatórios só para demonstrar lógica interna
  const numeros = [...Array(5)].map(() => Math.floor(Math.random() *
1000000));

  // Retorna os dados em formato JSON
  return {
    data: {
      titulo,
      numeros
    },
    type: "application/json"
```

```
  };
}
```

No exemplo acima, a função acessa `page`, extrai o título do site e gera uma lista de números aleatórios, então retorna um objeto com esses dados e especifica o tipo de conteúdo como JSON [3]. Esse é o formato correto esperado. **Sempre inclua o `return` com um objeto contendo `data` e `type`**, caso contrário o Browserless pode não saber como formatar a resposta.

> **Dica:** O código enviado **deve ser um módulo ECMAScript válido**. Isso significa que você pode utilizar sintaxe `import` para carregar bibliotecas externas se necessário (desde que sejam acessíveis via URL ESM) e não deve usar `require()` comum de Node. O ambiente de execução do Browserless para `/function` suporta top-level `await` e módulos ESM [4]. Certifique-se também de escrever uma função sintaticamente correta – um erro de sintaxe (por exemplo, esquecer de fechar chaves ou parênteses) resultará em erro ao processar o código.

2. Formato e Codificação do Corpo da Requisição

É fundamental enviar o conteúdo no **formato correto e com a codificação apropriada** para evitar erros como *"Unexpected end of input"*. Há **duas formas suportadas** de enviar o código pelo `/function`:

- **a) Como JavaScript puro (Content-Type: application/javascript):** você envia o código *diretamente* no corpo da requisição, como texto JavaScript. **Não envolva o código em JSON ou nenhum outro wrapper** – ele deve ser o script exatamente como seria escrito, iniciando com `export default async function` e terminando com o `}` de fechamento. Por exemplo, usando cURL isso seria: `-H "Content-Type: application/javascript" -d 'export default async function({ page }) { ... }'` [5] [6]. Nesse modo, o Browserless executa diretamente o código fornecido. **Atenção:** se optar por esse formato, você não poderá passar um `context` separado facilmente, pois o corpo é apenas o código JavaScript. (Você até pode acessar variáveis globais, mas a forma oficial de passar context é pelo formato JSON, explicado a seguir.)

- **b) Como JSON (Content-Type: application/json):** você envia um JSON no corpo contendo os campos `"code"` e opcionalmente `"context"`. Neste formato, o campo `"code"` é uma string com o código da função (em formato ESM, igual ao exemplo acima só que convertido para string), e `"context"` é um objeto JSON com quaisquer dados que você queira disponibilizar dentro da função [7] [8]. Por exemplo:

```
{
  "code": "export default async function({ context }) { return { data: context, type: 'application/json' }; }",
  "context": { "foo": "bar", "valor": 123 }
}
```

No exemplo JSON acima, estamos enviando uma função que simplesmente retorna o próprio objeto de contexto fornecido, e definimos `"context": {...}` no corpo. O Browserless vai inserir esse objeto como parâmetro na função (accessível via `context.foo`,

`context.valor`, etc.) [7]. Note que no JSON precisamos colocar todo o código JavaScript como uma única string – isso implica **escapar caracteres especiais** (por exemplo, em JSON, aspas dentro do código precisam ser `\"` ou usar aspas simples no código). Muitas vezes é conveniente **minificar ou colocar o código em uma única linha** para evitar problemas com quebras de linha e aspas no JSON [9]. A documentação oficial mostra um exemplo de código minificado dentro do JSON justamente por esse motivo [9].

Em ambos os casos acima, **use codificação UTF-8** (padrão para JSON e texto na web). Se você está enviando caracteres não ASCII dentro do código (por exemplo, strings em português), certifique-se de que o encoding da requisição esteja em UTF-8, o que normalmente é garantido pelo cabeçalho `Content-Type` correto.

Erro comum – "Unexpected end of input": Esse erro normalmente indica que o corpo enviado não pôde ser interpretado corretamente. As causas mais frequentes são: - **JSON malformado**: se você escolheu `application/json` mas o JSON está inválido (por exemplo, faltou fechar aspas ou chaves), resultando em erro de parse no servidor (HTTP 400). - **Código JavaScript truncado ou sintaticamente inválido**: por exemplo, esquecer de fechar `}` no final da função ou algum caractere especial quebrando a string, o que leva a um *SyntaxError: Unexpected end of input*. Nesses casos, o Browserless pode retornar um erro 400 informando que a requisição contém erros ou conteúdo não foi codificado corretamente [10].

Para evitar isso, sempre valide seu JSON (se usar esse formato) e teste o código JavaScript separadamente para garantir que não haja erros de sintaxe.

3. Cabeçalhos HTTP Apropriados

Configurar os cabeçalhos corretamente é essencial para o Browserless entender sua requisição: - **Content-Type:** defina de acordo com o formato escolhido: - Use `Content-Type: application/javascript` se estiver enviando o código bruto JavaScript no corpo [11] [12]. - Use `Content-Type: application/json` se estiver enviando um JSON com os campos `"code"` e `"context"` [8] [13]. - **Authorization/Token:** O Browserless **usa token de API para autenticação**. Se você estiver usando o serviço na nuvem ou uma instância protegida, inclua o token de API. A forma mais comum (conforme a documentação) é acrescentar o token na query string da URL (por exemplo `?token=SEU_TOKEN_AQUI`) [5]. Alternativamente, algumas instâncias podem aceitar `Authorization: Bearer SEU_TOKEN` no header (embora a documentação enfatize o uso do parâmetro na URL). Certifique-se de **não esquecer o token** se ele for exigido – uma requisição sem token ou com token inválido resultará em erro **401 Unauthorized** [10]. Em ambiente self-hosted do Browserless, se você não configurou nenhum token, então não é necessário autenticar (nesse caso, usar apenas a URL local sem token). - **Outros cabeçalhos:** Geralmente não é necessário nenhum outro header específico para o endpoint `/function`. Não use `Content-Length` manualmente (isso é gerenciado pelo cliente HTTP/n8n automaticamente). Também não configure `Accept` manualmente – o Browserless retornará o tipo correto baseado no resultado da função. Somente se sua função exigir algo especial (por exemplo, talvez definir um agente de usuário global, o que geralmente é feito via código `page.setUserAgent`, não via header HTTP da requisição), mas isso foge do escopo do endpoint em si.

Recapitulando, um exemplo mínimo de cabeçalhos para a requisição seria:

```
Content-Type: application/javascript
```

e se necessário:

```
Authorization: Bearer <SEU_TOKEN>
```

(ou token na URL, veja abaixo). Sempre cheque se o token está correto; caso contrário, você verá erros de autorização (401/403).

4. URL Correta com ou sem Token

A URL base a ser usada depende de onde seu Browserless está rodando: - **Instância em Nuvem (Browserless SaaS):** Utilize a URL fornecida pela Browserless. A documentação sugere escolher a região mais próxima, por exemplo: `https://production-sfo.browserless.io/function` (para US Oeste) ou `...-lon.browserless.io` (Europa), etc. [14] . **Você deve adicionar o parâmetro** `token` **na query string**, por exemplo:

```
https://production-sfo.browserless.io/function?token=YOUR_API_TOKEN_HERE
```

[5] . Sem o token, a chamada retornará 401/403. Esse token é aquele fornecido na sua conta Browserless (ou configurado na instância). **Não inclua o token em clientes web ou locais inseguros**, pois ele dá acesso ao seu serviço [15] . - **Instância Self-Hosted:** Se você está rodando o Browserless no seu próprio servidor ou localmente, a URL padrão geralmente é algo como `http://localhost:3000/function` (ou o host/porta que você definiu). Por padrão, instâncias self-hosted não exigem token **a menos que você tenha configurado um via variável de ambiente**. Se um token foi definido na instância self-host, então mesmo localmente você deve passar `?token=...` na URL ou usar a forma de autenticação apropriada. Caso contrário (nenhum token configurado), basta usar o endpoint diretamente sem parâmetros de autenticação.

Importante: Note que na documentação OpenAPI os endpoints são referenciados como `/chrome/function`, mas na prática a URL pública é apenas `/function` quando usando as URLs base fornecidas (como `...browserless.io/function`). O prefixo `/chrome` pode ser um detalhe interno do API, mas **a chamada correta é** `/function` **na maioria dos casos de uso externos**. Por exemplo, o cURL oficial usa `/function` direto na URL [5] . Certifique-se de não acidentalmente usar um caminho errado (um *404 Not Found* ocorrerá se o endpoint estiver incorreto ou se você errar o caminho [16]).

Resumindo, combine a URL base da sua instância + `/function`, e acrescente `?token=` se for necessário. Exemplos: - Cloud: `https://production-sfo.browserless.io/function?token=SEU_TOKEN` - Local (sem token): `http://meu-servidor:3000/function`

5. Exemplos de Código Válidos vs. Inválidos

Para clareza, vejamos alguns exemplos e armadilhas comuns:

- **Válido:** *(Já mostramos acima um exemplo válido básico.)* Outro exemplo válido, usando `context`: digamos que queremos gerar dados fake. Podemos enviar no corpo JSON:

```
{
  "code": "import { faker } from 'https://esm.sh/@faker-js/faker';
\nexport default async function({ context }) {\n  const name =
faker.name.fullName();\n  return { data: { nomeGerado: name, parametro:
context.param }, type: 'application/json' };\n}",
  "context": { "param": 42 }
}
```

  Neste exemplo, o código importou a biblioteca Faker via URL ESM e gerou um nome aleatório. Ele retorna um objeto JSON com o nome gerado e inclui também um valor do contexto (`context.param`). O retorno especifica `application/json`. Esse código segue todas as regras: é um módulo ESM válido, exporta função async default e retorna `{ data, type }`.

- **Válido (retorno não-JSON):** Você pode retornar outros tipos de conteúdo. Por exemplo, para tirar uma captura de tela (screenshot) e retornar a imagem, você pode fazer:

```
export default async function ({ page }) {
  await page.goto("https://google.com");
  const buffer = await page.screenshot({ fullPage: true });
  return { data: buffer, type: "image/png" };
}
```

  Aqui retornamos um Buffer binário de imagem e o tipo `"image/png"`. O Browserless então devolverá a imagem PNG diretamente na resposta HTTP com esse content-type. **Atenção:** Ao usar n8n, se você fizer isso, configure o nó HTTP Request para receber binário (veja seção de ajustes) ou então o n8n pode tentar interpretar o binário como texto e gerar erro.

- **Inválido: Não incluir `export default` ou usar a assinatura errada.** Por exemplo, se você enviar:

```
async function myFunc(page) {
  // ...
  return { data: "ok", type: "text/plain" };
}
```

  Isso **não vai funcionar**, pois o Browserless espera **uma função exportada como default**. Ele não executará uma função que não esteja exportada ou que não seja default. Além disso, a assinatura esperada deve usar destruturação `{ page }` no parâmetro (ou `{ page, context }`). No exemplo acima, usamos `page` diretamente como parâmetro posicional, o que não corresponde ao objeto passado. O correto seria `export default async`

`function({ page }) { ... }`. Se você esquecer o `export default`, o servidor provavelmente retornará erro 400 (código não executável) ou até um erro interno se tentar rodar e não encontrar a função esperada.

- **Inválido: Não retornar nada ou retornar valor incorreto.** Se sua função executa comandos mas não dá um `return`, o Browserless não terá um resultado para enviar. Frequentemente isso resultará em um timeout (erro 408) se a função nunca retorna [17] ou em uma resposta vazia inesperada. Certifique-se de sempre retornar algum resultado. Similarmente, se você retornar algo que **não seja serializável ou suportado**, pode causar erro. Retornar, por exemplo, um handle de elemento DOM ou um objeto complexo do contexto do navegador sem processá-lo pode causar *internal error*. Sempre extraia os dados necessários (por exemplo, use `page.evaluate` para retornar dados primitivos ou JSON do contexto da página, ao invés de retornar diretamente um objeto DOM). Um exemplo incorreto seria:

```
export default async function({ page }) {
  await page.goto("https://example.com");
  const handle = await page.$("h1");
  return { data: handle, type: "application/json" };
}
```

Isso é inválido porque `handle` (um ElementHandle do Puppeteer) não pode ser serializado em JSON. A chamada acima resultaria em erro interno (500) no Browserless ao tentar montar o JSON de resposta. O correto seria extrair algo do handle (como seu texto) antes de retornar, ou retornar uma indicação apropriada.

- **Inválido: Código com erro de sintaxe ou malformatação.** Por exemplo, esquecer um `}`:

```
export default async function({ page }) {
  await page.goto("https://example.com");
  // esqueci de fechar a função!
```

Isso resultará imediatamente em um erro 400 (*Unexpected end of input*), já que o parser de JS não consegue interpretar o código. Sempre teste seu código JavaScript num ambiente local ou linte o código para garantir que esteja sintaticamente correto antes de enviar.

Resumindo, siga os padrões dos exemplos válidos: **exportação default, função async, uso correto de `page` / `context`, e return de `{ data, type }`.** Quando esses critérios não são atendidos, você tende a receber erros do servidor ou resultados inesperados.

6. Ajustes no nó HTTP Request do n8n

Para chamar o endpoint `/function` a partir de um workflow n8n, é necessário configurar o nó HTTP Request de forma adequada:

- **Método e URL:** Selecione método **POST** e coloque a URL completa do seu endpoint (conforme explicado na seção da URL, incluindo o token se necessário). Ex: `https://production-sfo.browserless.io/function?token=SEU_TOKEN`.

- **Modo do Corpo da Requisição:** No nó HTTP Request do n8n, ative a opção **"JSON/RAW Parameters"** (nos paineis recentes do n8n, isso aparece como alternar **"Enviar corpo como JSON/RAW"**). Em seguida:

- Escolha **Content Type = Raw** (nas opções adicionais, definir *Body Content Type* como "raw").
- **Não** adicione pares de chave/valor no Body se estiver no modo raw. Em vez disso, forneça todo o conteúdo do corpo em uma única string.
- Se for enviar JavaScript puro: você pode simplesmente colar a função no campo de corpo. Certifique-se de incluí-la completa. Por exemplo, no campo de texto do corpo poderia ficar exatamente:

```
export default async function({ page }) {
  await page.goto("https://exemplo.com");
  return { data: await page.title(), type: "text/plain" };
}
```

  Não envolva em aspas ou JSON manualmente – o n8n enviará exatamente esse texto bruto.
- Se for enviar JSON com `code` / `context`: você pode optar por montar um objeto JSON no próprio n8n. Por exemplo, ainda em JSON/RAW, coloque algo como:

```
{
  "code": "export default async function({ context }) { return { data: context.msg, type: 'text/plain' } }",
  "context": { "msg": "Olá do contexto" }
}
```

  O n8n irá enviar esse JSON no corpo com content-type application/json. Uma dica: construir a string do código dentro do JSON no n8n pode ser complexo se for muito longa ou precisar de escapes. Uma alternativa é montar o código antes (em um Function Node do n8n talvez) e passar como variável. Mas se for muito complicado, considere usar o método de JavaScript puro para evitar lidar com aspas escapadas.

- **Cabeçalhos no n8n:** Em *Headers*, adicione manualmente `Content-Type` com o valor adequado (`application/javascript` ou `application/json`). Embora o n8n possa inferir em alguns casos, é mais seguro definir explicitamente. Também insira o header de Authorization se você preferir passar o token no header (ex: `Authorization: Bearer SEU_TOKEN`). Se o token está na URL, não é preciso adicionar esse header de auth.

- **Formato da Resposta (Response Options):** Por padrão, o nó HTTP Request do n8n tenta interpretar a resposta automaticamente. Se o Browserless retornar `application/json` e um JSON válido, o n8n irá parsear e você verá os dados normalmente. **Porém, se a resposta não for JSON (por ex., um PDF, imagem, HTML ou mesmo uma mensagem de erro em texto), o n8n pode gerar um erro do tipo "Unexpected end of JSON input" ao tentar interpretar.** Para evitar isso, é recomendável ajustar a opção de resposta:

- Expanda **Options** (Opções) no nó HTTP Request.

- Em **Response** -> **Response Format**, selecione **"String"** (Texto) ou **"File"** conforme o caso.
    - Se sua função retorna deliberadamente um conteúdo não-JSON (HTML, texto simples, etc.), usar **"String"** fará com que o n8n trate qualquer retorno como texto bruto, evitando erro de parsing [18].
    - Se sua função retorna um binário (imagem, PDF), você pode escolher **"File"** ou **"Binary"** para que o n8n trate adequadamente. Assim, você poderá pegar o binário nos outputs (marcando para *Binary Data*).

- **Dica:** Deixar em "Automatic" também pode funcionar se o content-type retornado for bem conhecido, mas para garantir, defina explicitamente. Por exemplo, usuários relataram que definir para *Texto* resolveu o erro de *JSON input* quando a API não retornava JSON [18].

- **Tratamento de Erros HTTP:** O Browserless retornará códigos de erro HTTP (400, 500 etc.) em situações de falha. Por padrão, o n8n vai considerar status >=400 como erro do workflow. Se você quiser capturar a resposta de erro e seguir o fluxo (por exemplo, para ler a mensagem de erro retornada pelo Browserless), você pode habilitar **"Ignore HTTP errors"** ou **"Continue On Fail"** (dependendo da versão do n8n, pode ser uma caixa "Never error"). Um usuário reportou que habilitar a opção *"Never error"* permitiu que a execução continuasse e ele pôde inspecionar a resposta de erro do Browserless no próprio n8n [19] [20]. Use isso com cautela – é útil para depurar respostas do Browserless (por exemplo, ver o conteúdo de um 500 Internal Server Error), mas em produção você talvez queira tratar essas condições explicitamente.

Resumindo a configuração no n8n: 1. POST na URL correta (com token se necessário).
2. Ativar JSON/RAW e definir Body Content Type = raw.
3. Colocar o código (ou JSON com código) no corpo exatamente no formato esperado.
4. Definir `Content-Type` nos headers de acordo.
5. Ajustar Response Format para lidar com o tipo de retorno esperado (JSON, texto ou binário).
6. (Opcional) Ativar ignore errors para capturar respostas de erro do Browserless.

Seguindo esses passos, sua chamada ao Browserless via n8n deve funcionar sem encontrar os erros de input inesperado, pois estaremos enviando o conteúdo no formato correto.

7. Erros Comuns do `/function` e Motivos (segundo a documentação)

Mesmo com tudo configurado, é útil conhecer os erros mais comuns retornados pelo endpoint `/function` e suas possíveis causas:

- **Erro 400 – Bad Request:** Isso acontece quando a requisição em si possui algum problema. De acordo com a documentação, *"The request contains errors or didn't properly encode content."* [10]. As causas típicas: código JavaScript inválido (sintaxe incorreta), JSON malformado, ou cabeçalho Content-Type incorreto (fazendo o server ler o corpo de forma errada). Basicamente, o Browserless não conseguiu entender seu pedido. A mensagem de erro retornada pode incluir detalhes, como um SyntaxError JavaScript (por ex., *Unexpected end of input* se o código estiver incompleto). Solução: revisar o corpo enviado, conferir se o `Content-Type` bate com o conteúdo (JSON vs JS) e corrigir eventuais erros de formatação.

- **Erro 401/403 – Unauthorized/Forbidden:** Significa falha de autenticação. Normalmente, **token ausente ou inválido** na chamada. O Browserless exige token quando configurado, e a doc indica que chamadas sem token válido resultam em 401/403 [15] [10]. Solução: verificar se você

passou `?token=` correto ou o header de Authorization. Se estiver self-hosted sem token, cheque se por acaso a instância foi iniciada com exigência de credencial.

- **Erro 408 – Request Timeout:** Indica que seu código demorou demais para terminar. O Browserless, por padrão, tem um tempo limite global (geralmente ~30 segundos, a não ser que configurado diferente). Se sua função não conclui dentro do tempo, o servidor aborta. A doc menciona: *"The request took too long to process."* [17]. Isso pode ocorrer se a página demorou a carregar ou se seu código entrou em loop. Solução: tentar reduzir o tempo (p.ex., usar `page.waitForSelector` com timeout menor, ou verificar se não há espera infinita). Você também pode aumentar o timeout passando um parâmetro `&timeout=` na URL (em milissegundos) se realmente precisar de mais tempo, mas isso deve ser usado com cuidado.

- **Erro 429 – Too Many Requests:** O Browserless está rejeitando porque há requisições demais simultâneas ou você excedeu sua cota. A doc: *"Too many requests are currently being processed."* [21]. Em fluxos n8n, isso pode acontecer se você disparar muitos nós Browserless ao mesmo tempo ou se sua conta na nuvem estiver acima do limite. Solução: implementar filas/limites no n8n (por ex., usar nó de espera ou limitar a concorrência) ou contratar mais capacidade.

- **Erro 500 – Internal Server Error:** Indica que algo deu errado durante a execução da sua função no lado do Browserless. A documentação descreve: *"An internal error occurred when handling the request."* [22]. Isso acontece normalmente por **exceções no código** – por exemplo, acessar algo undefined, lançar um erro não capturado, ou retornar um tipo de dado que não pode ser serializado como mencionado antes. Quando o código quebra, o Browserless retorna 500. A resposta pode vir em formato JSON com uma mensagem de erro, ou em texto/HTML. Por exemplo, se sua função lança uma exceção no Puppeteer, você pode receber uma mensagem de erro de stack. Outra situação: no passado, usuários que retornaram diretamente objetos do DOM obtiveram respostas HTML estranhas (um bug chamado "browserless function runner" mostrando HTML em vez do resultado esperado). Esse comportamento foi esclarecido: era necessário usar `page.evaluate` para obter o conteúdo da página, em vez de retornar `document` diretamente, pois o código roda fora do contexto da página [23]. Em resumo, 500 significa que a função não completou com sucesso. Solução: debugar o código – rode localmente se possível, ou capture o erro retornado. Use a opção do n8n de não parar em erro para logar a resposta de erro. Corrija exceções e garanta retornar apenas dados válidos.

Além desses, erros menos comuns: - **Erro 404 – Not Found:** Se você vê um 404, provavelmente a URL está incorreta (endpoint errado) ou, possivelmente, está usando um caminho de API inexistente [16]. Verifique se está usando `/function` corretamente na URL e não, por exemplo, confundiu com outro caminho. - **Erro 502/503:** Podem indicar problemas temporários no serviço (gateway ou instância do Browserless indisponível). Nesses casos, tente novamente ou verifique a instância.

Para finalizar, lembre-se que seguir as especificações oficiais é o melhor caminho: monte a requisição exatamente como mostrado nos exemplos da documentação do Browserless (que cobrimos aqui). Assim você evita os erros de *Unexpected end of input* (relacionados a formatação) e *500 Internal Server Error* (relacionados ao código em si). Com a configuração do n8n ajustada para envio bruto e tratamento correto da resposta, você poderá integrar o Browserless nas suas workflows de forma robusta e eficiente.

Referências: As recomendações acima foram elaboradas com base na documentação oficial do Browserless [2] [24] e em experiências práticas com n8n. Siga este guia e bons automations!

1 2 3 4 5 6 7 8 9 11 12 13 /function API | Browserless.io
https://docs.browserless.io/rest-apis/function

10 16 17 21 22 24 Browserless | Browserless.io
https://docs.browserless.io/open-api

14 15 Connection URLs and Endpoints | Browserless.io
https://docs.browserless.io/rest-apis/connection-urls

18 19 20 ERROR: Unexpected end of JSON input - Questions - n8n Community
https://community.n8n.io/t/error-unexpected-end-of-json-input/23034

23 browserless.io function buged · Issue #4590 · browserless/browserless · GitHub
https://github.com/browserless/browserless/issues/4590