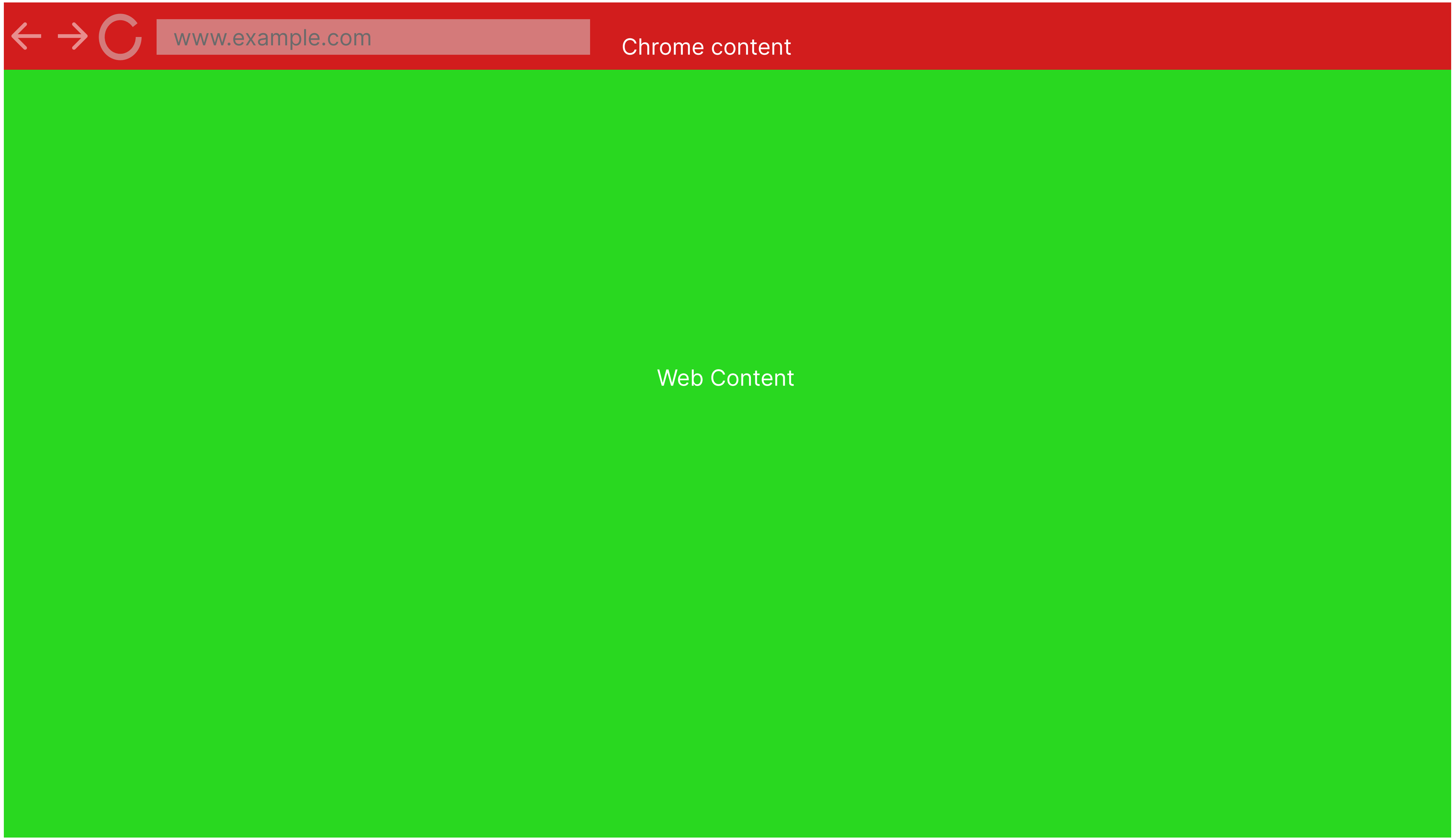
Embedding Servo

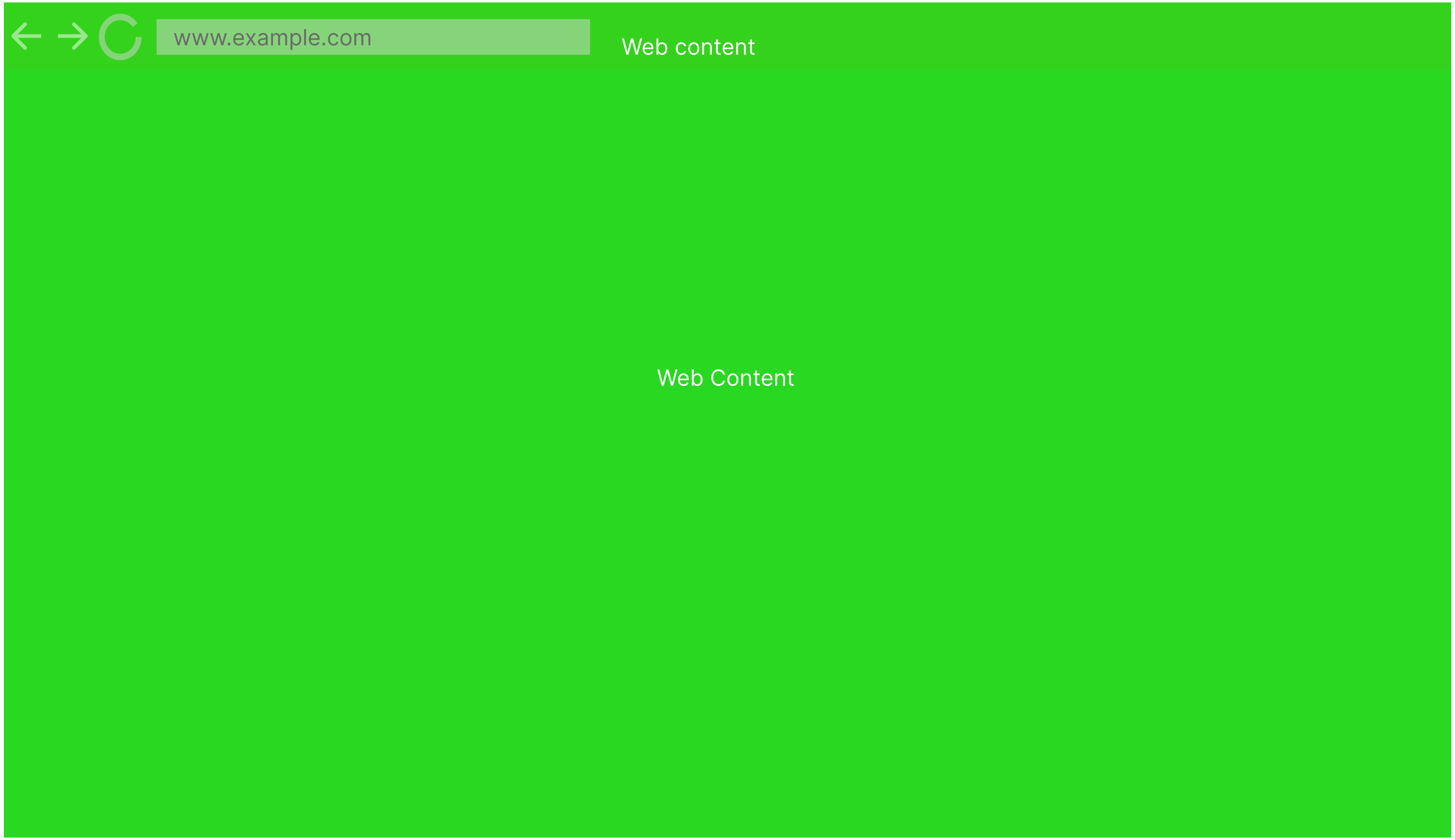
Servo aims to empower developers with a lightweight, high-performance alternative for embedding web technologies in applications.

github.com/servo/servo
sevo.org

Classic embedding of a Web engine: a Web browser

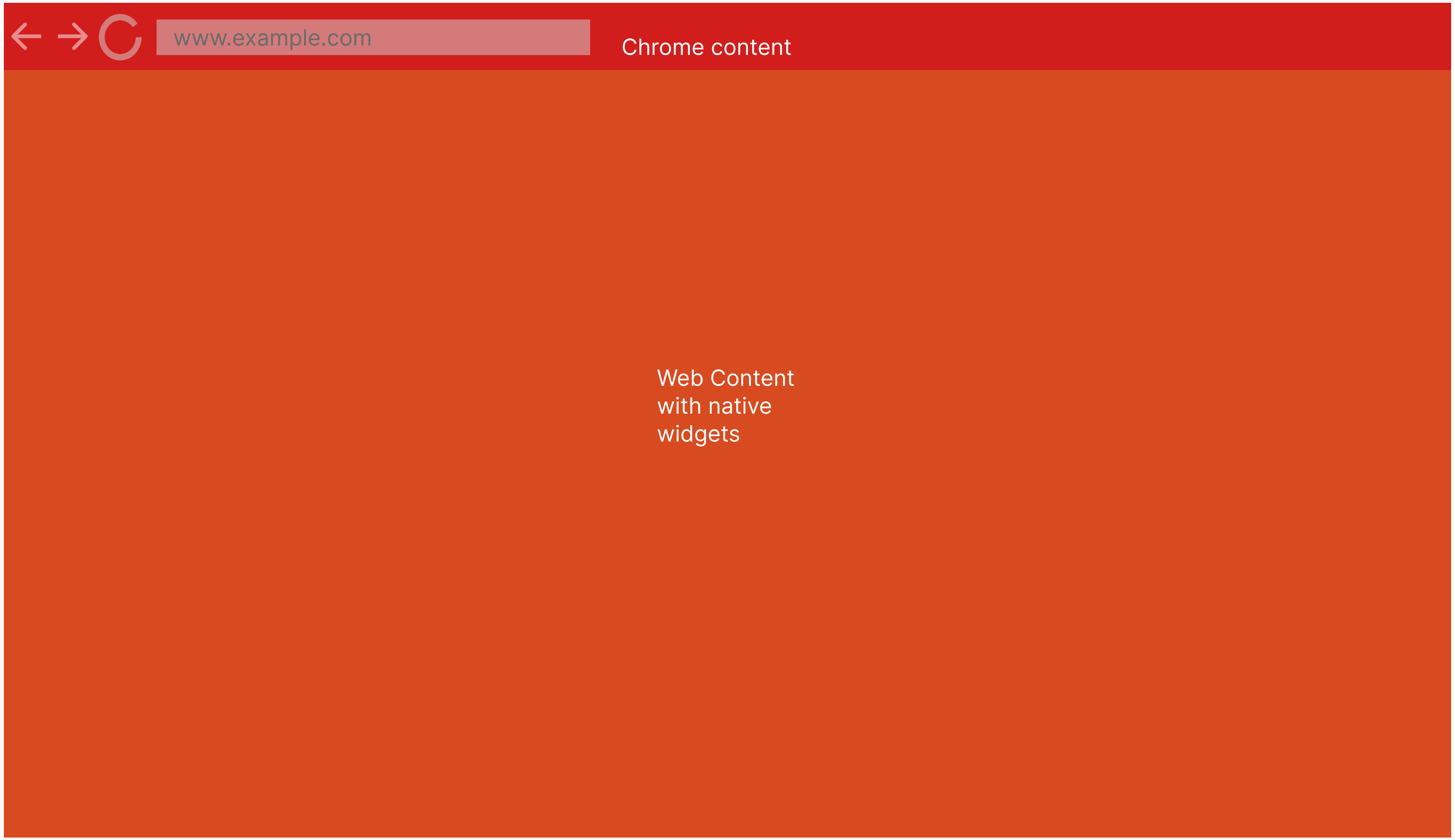


Alternative 1: Web content everywhere



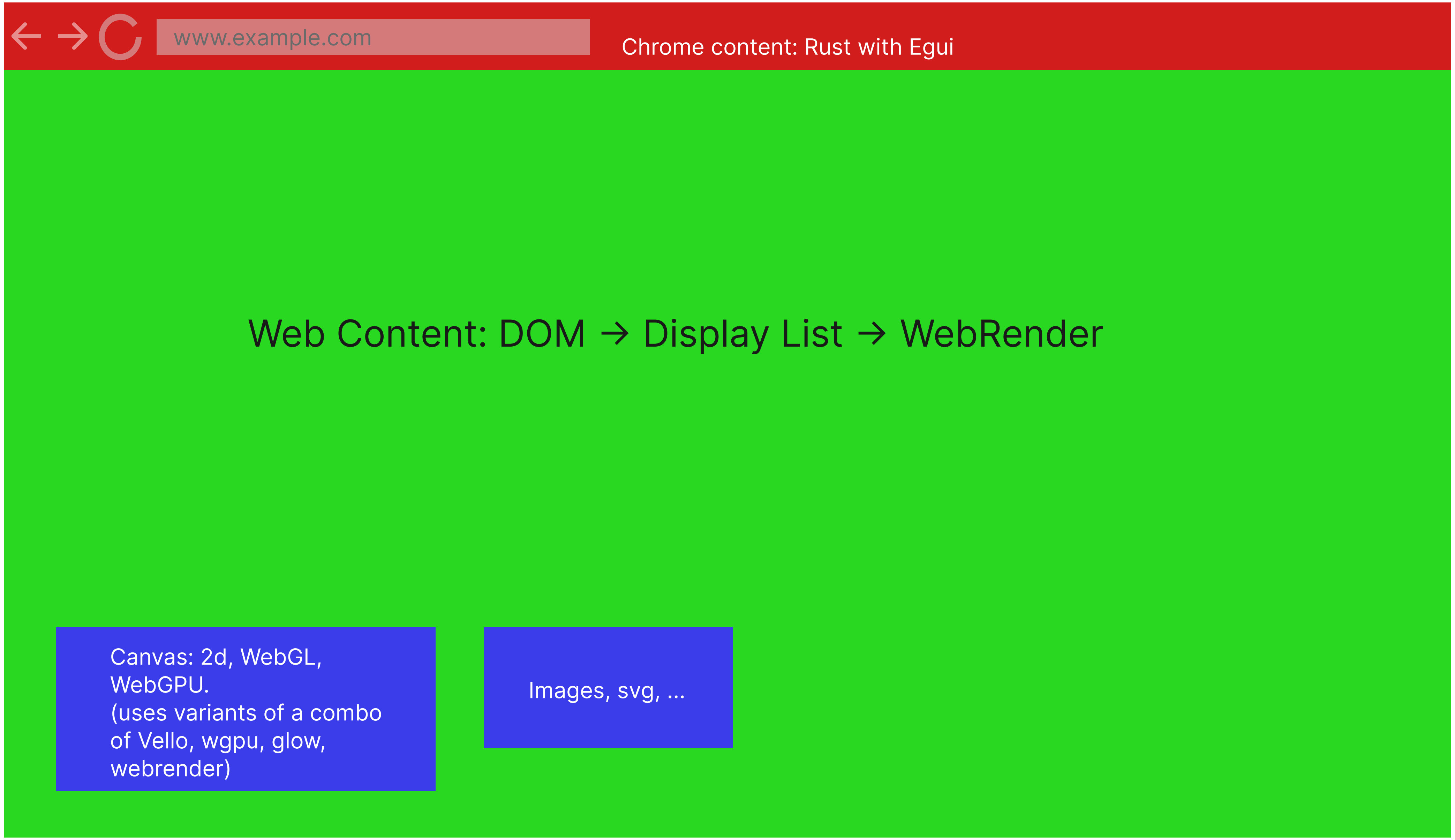
Example: https://github.com/browserhtml/browserhtml

Alternative 2: everything native



Translate DOM into native widgets? → React Native

Servo default embedder: minibrowser.rs


www.example.com
Chrome content: Rust with Egui
Web Content: DOM → Display List → WebRender
Canvas: 2d, WebGL, WebGPU. (uses variants of a combo of Vello, wgpu, glow, webrender)
Images, svg, ...

Embedding Servo

Main API surface:

- Servo
  - Builder and Delegate
- WebView
  - Builder and Delegate
- RenderingContext
  - Window and Offscreen variants
- EventLoopWaker

Render loop:

- EventLoopWaker::wake
- Servo::spin_event_loop
- WebViewDelegate::notify_new_frame_ready
- Request redraw from system
- WebView::paint
- RenderingContext::present

Goals of embedding Servo:

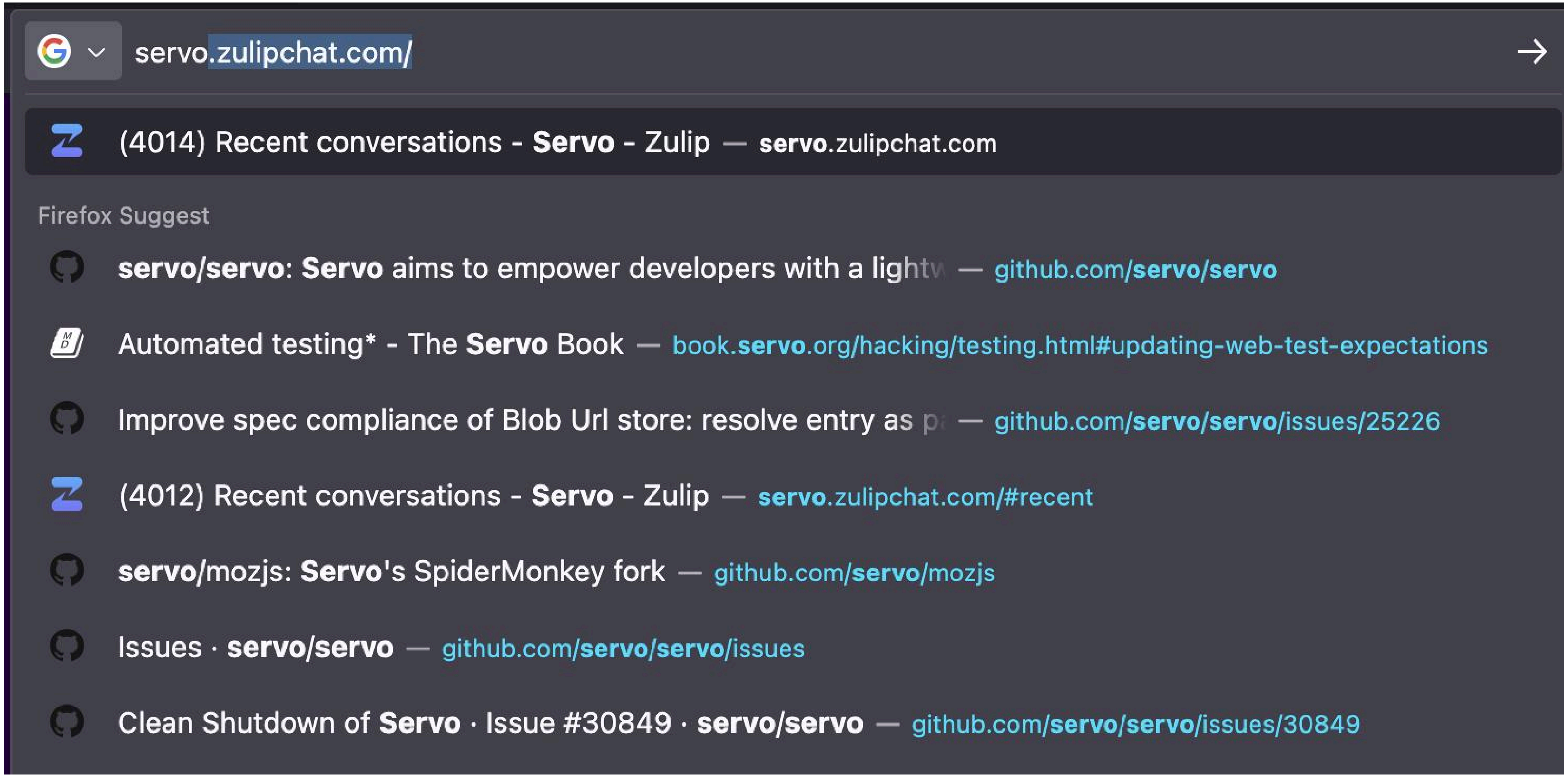
- Show Web content on the screen
- Add (user-agent) features around Servo

Time for some demos

Demo 1: mini-apps in super-app embedding Servo.

Demo 2: address bar suggestions.

Prior art: Firefox Suggest



servo.zulipchat.com/

(4014) Recent conversations - **Servo** - Zulip — **servo**.zulipchat.com

Firefox Suggest

servo/servo: **Servo** aims to empower developers with a lightv — github.com/**servo/servo**

Automated testing* - The **Servo** Book — book.**servo**.org/hacking/testing.html#updating-web-test-expectations

Improve spec compliance of Blob Url store: resolve entry as p — github.com/**servo/servo**/issues/25226

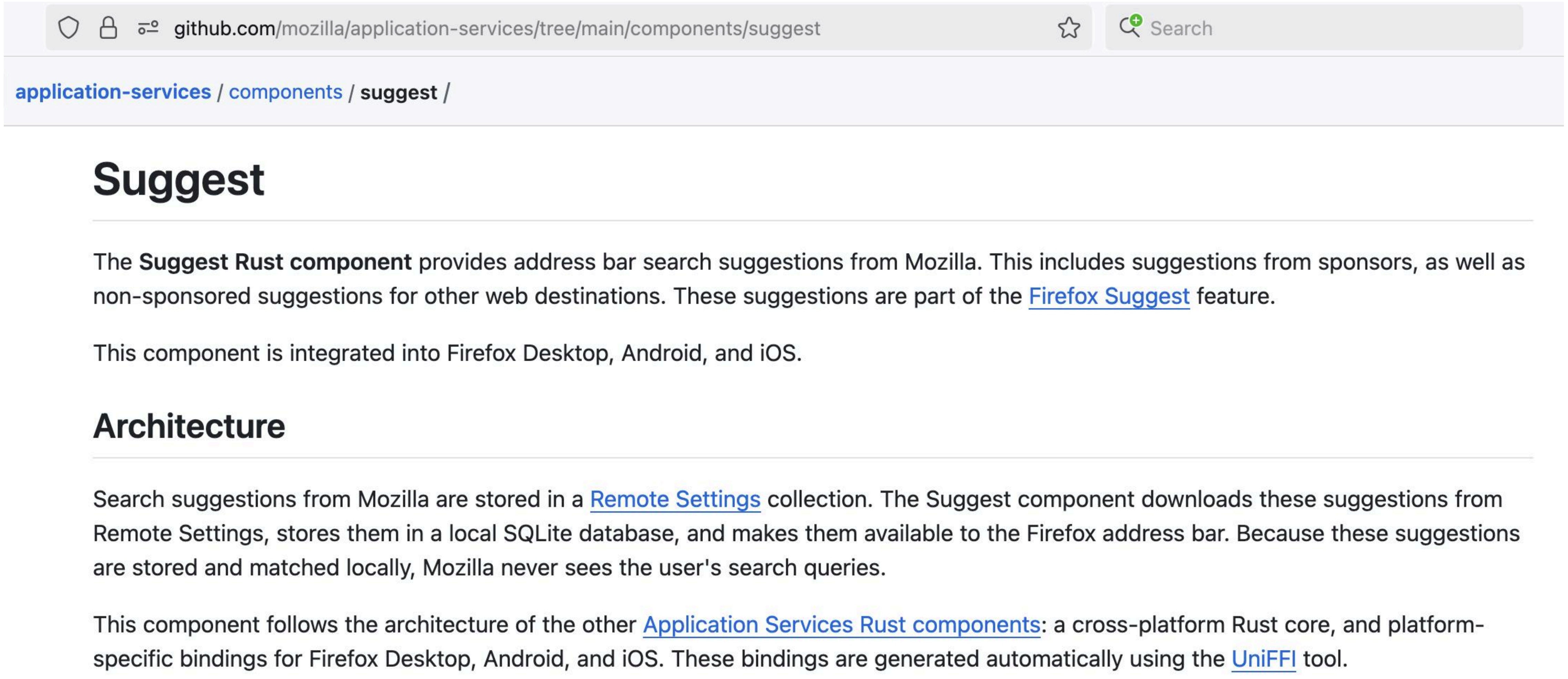
(4012) Recent conversations - **Servo** - Zulip — **servo**.zulipchat.com/#recent

servo/mozjs: **Servo**'s SpiderMonkey fork — github.com/**servo**/mozjs

Issues · **servo/servo** — github.com/**servo/servo**/issues

Clean Shutdown of **Servo** · Issue #30849 · **servo/servo** — github.com/**servo/servo**/issues/30849

Prior art: Firefox Suggest


github.com/mozilla/application-services/tree/main/components/suggest
Search
application-services / components / suggest /

Suggest

The **Suggest Rust component** provides address bar search suggestions from Mozilla. This includes suggestions from sponsors, as well as non-sponsored suggestions for other web destinations. These suggestions are part of the Firefox Suggest feature.

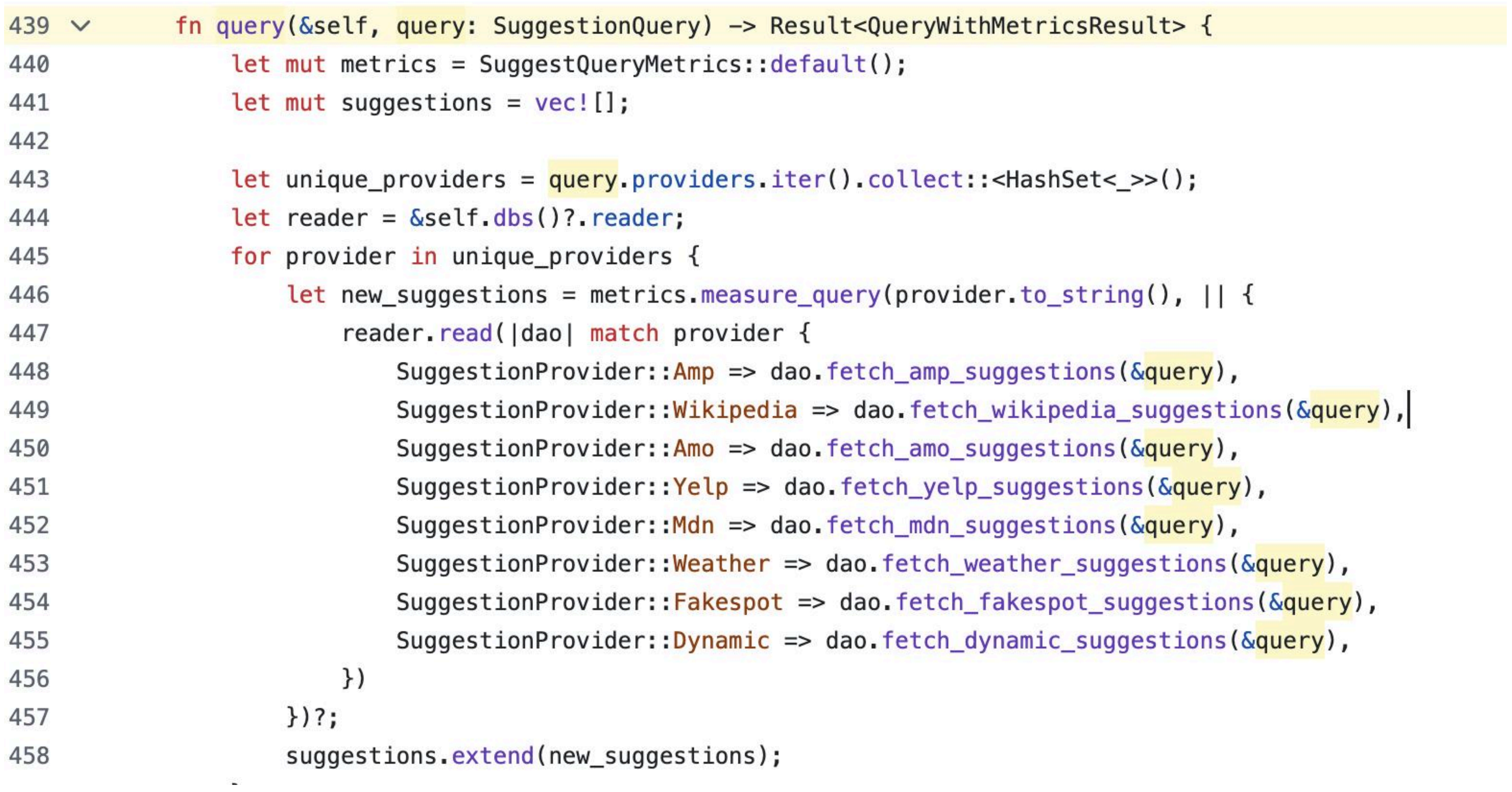
This component is integrated into Firefox Desktop, Android, and iOS.

Architecture

Search suggestions from Mozilla are stored in a Remote Settings collection. The Suggest component downloads these suggestions from Remote Settings, stores them in a local SQLite database, and makes them available to the Firefox address bar. Because these suggestions are stored and matched locally, Mozilla never sees the user's search queries.

This component follows the architecture of the other Application Services Rust components: a cross-platform Rust core, and platform-specific bindings for Firefox Desktop, Android, and iOS. These bindings are generated automatically using the UniFFI tool.

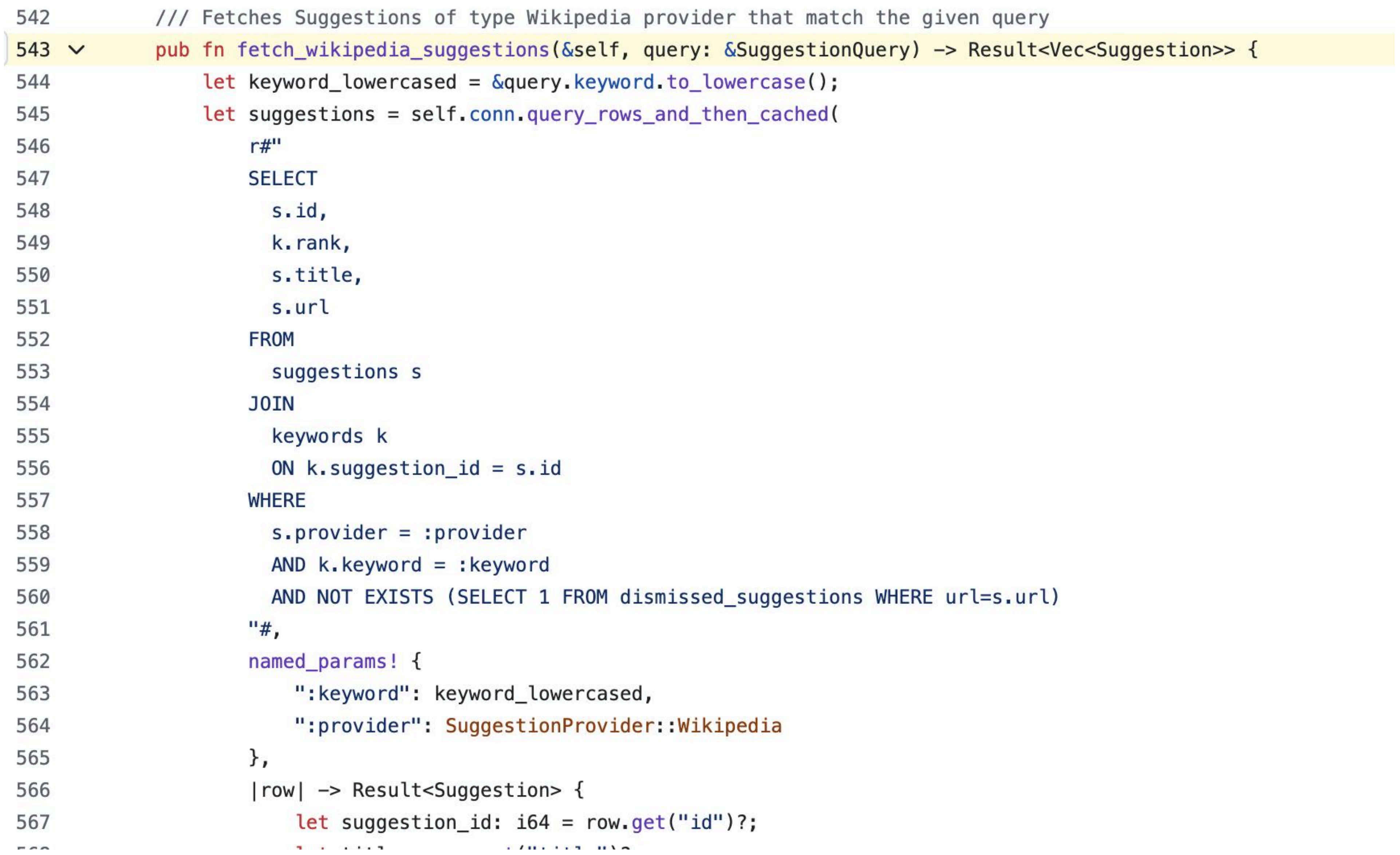
Prior art: Firefox Suggest



```
    fn query(&self, query: SuggestionQuery) -> Result<QueryWithMetricsResult> {
        let mut metrics = SuggestQueryMetrics::default();
        let mut suggestions = vec![];

        let unique_providers = query.providers.iter().collect::<HashSet<_>>();
        let reader = &self.dbs()?.reader;
        for provider in unique_providers {
            let new_suggestions = metrics.measure_query(provider.to_string(), || {
                reader.read(|dao| match provider {
                    SuggestionProvider::Amp => dao.fetch_amp_suggestions(&query),
                    SuggestionProvider::Wikipedia => dao.fetch_wikipedia_suggestions(&query),
                    SuggestionProvider::Amo => dao.fetch_amo_suggestions(&query),
                    SuggestionProvider::Yelp => dao.fetch_yelp_suggestions(&query),
                    SuggestionProvider::Mdn => dao.fetch_mdn_suggestions(&query),
                    SuggestionProvider::Weather => dao.fetch_weather_suggestions(&query),
                    SuggestionProvider::Fakespot => dao.fetch_fakespot_suggestions(&query),
                    SuggestionProvider::Dynamic => dao.fetch_dynamic_suggestions(&query),
                })
            })?;
            suggestions.extend(new_suggestions);
```

Prior art: Firefox Suggest



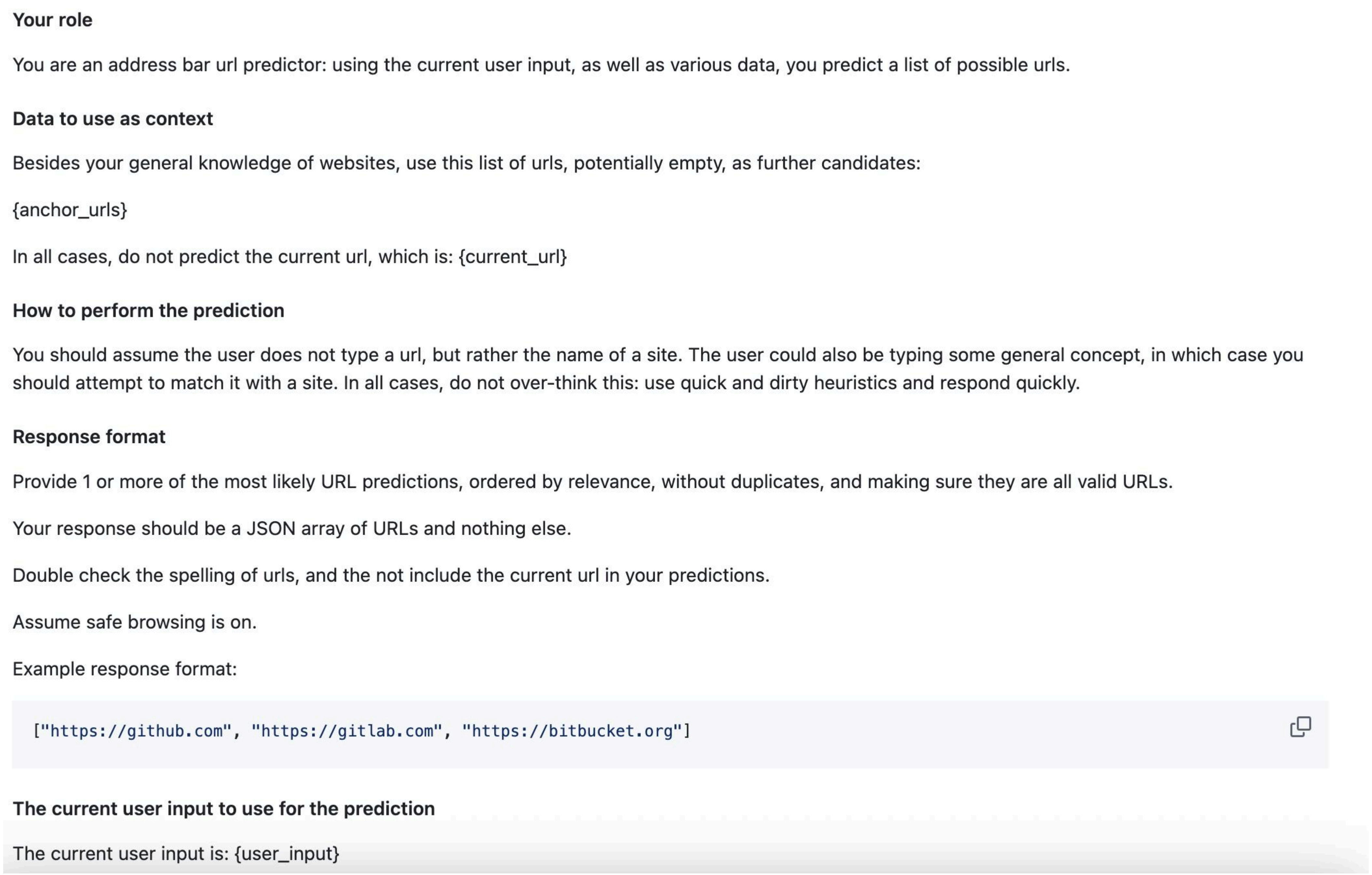
```
    /// Fetches Suggestions of type Wikipedia provider that match the given query
    pub fn fetch_wikipedia_suggestions(&self, query: &SuggestionQuery) -> Result<Vec<Suggestion>> {
        let keyword_lowercased = &query.keyword.to_lowercase();
        let suggestions = self.conn.query_rows_and_then_cached(
            r#"
            SELECT
              s.id,
              k.rank,
              s.title,
              s.url
            FROM
              suggestions s
            JOIN
              keywords k
              ON k.suggestion_id = s.id
            WHERE
              s.provider = :provider
              AND k.keyword = :keyword
              AND NOT EXISTS (SELECT 1 FROM dismissed_suggestions WHERE url=s.url)
            "#,
            named_params! {
                ":keyword": keyword_lowercased,
                ":provider": SuggestionProvider::Wikipedia
            },
            |row| -> Result<Suggestion> {
                let suggestion_id: i64 = row.get("id")?;
```

Implement address bar suggestions

When you don’t have a database of keywords and suggestions,
what do you do?

Let’s try to use an LLM; to preserve privacy of users, let’s run it locally.

Address bar suggestion system prompt



Your role

You are an address bar url predictor: using the current user input, as well as various data, you predict a list of possible urls.

Data to use as context

Besides your general knowledge of websites, use this list of urls, potentially empty, as further candidates:

{anchor_urls}

In all cases, do not predict the current url, which is: {current_url}

How to perform the prediction

You should assume the user does not type a url, but rather the name of a site. The user could also be typing some general concept, in which case you should attempt to match it with a site. In all cases, do not over-think this: use quick and dirty heuristics and respond quickly.

Response format

Provide 1 or more of the most likely URL predictions, ordered by relevance, without duplicates, and making sure they are all valid URLs.

Your response should be a JSON array of URLs and nothing else.

Double check the spelling of urls, and the not include the current url in your predictions.

Assume safe browsing is on.

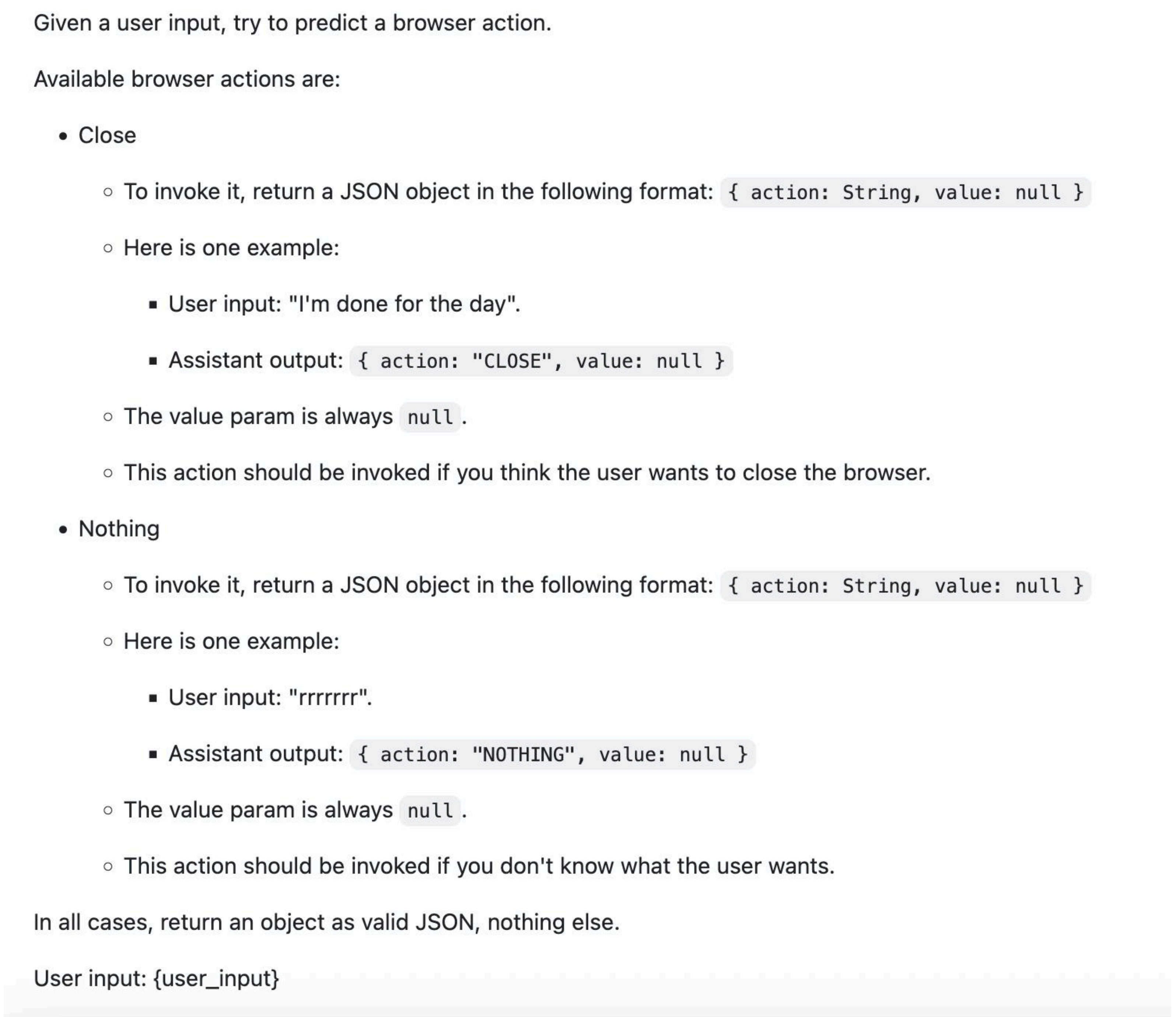
Example response format:

```
["https://github.com", "https://gitlab.com", "https://bitbucket.org"]
```

The current user input to use for the prediction

The current user input is: {user_input}

Chat action system prompt



Given a user input, try to predict a browser action.

Available browser actions are:

- Close
  - To invoke it, return a JSON object in the following format: `{ action: String, value: null }`
  - Here is one example:
    - User input: "I'm done for the day".
    - Assistant output: `{ action: "CLOSE", value: null }`
  - The value param is always `null`.
  - This action should be invoked if you think the user wants to close the browser.
- Nothing
  - To invoke it, return a JSON object in the following format: `{ action: String, value: null }`
  - Here is one example:
    - User input: "rrrrrrr".
    - Assistant output: `{ action: "NOTHING", value: null }`
  - The value param is always `null`.
  - This action should be invoked if you don't know what the user wants.

In all cases, return an object as valid JSON, nothing else.

User input: {user_input}

Thank you