INFO 498: Architecture Report

Anya Hsu

Natalia Abrosimova

## System Description

Google Chrome is free, open-source browser was released by Google on December 11, 2008. It is available for Mac OS X, Linux, Android, Windows and IOS. Its basic features include but are not limited to tab browsing, organization of bookmarks, address bar search, synchronization with Google accounts, incognito search. To increase Chrome’s performance it has each open page runs alone, independently from other pages and the browser, which also helps prevent one page with malicious software cause problems for the rest of the pages. Its interface is simple yet effective, Chrome combined the address bar and search box, let bookmarks tie up to the Google account, making it easy to sync with other devices and also added apps that users can download and add to their browser making it easy to personalize.

## Architecture Description

### Chrome

At the highest level, Google Chrome’s overall conceptual architecture is a semi-strict layering style. Figure 1 demonstrates the different layers, and how the components of the hierarchy do not directly interact with the layers that are not adjacent to it. At lower levels however, the coupling between some of the interacting components can be seen as object-oriented behaviour as well, such as how the browser and the V8 Javascript Engine are coupled (see diagram).

There are many components to this architecture:

* The User Interface is the layer that lets the user access the functionality of the browser (i.e. what the user sees)
* The Browser is the central component, and is responsible for tasks like handling user input, window management, communicating with the network, history database, spawning new tabs, etc.
* The Network Stack takes care of all the Universal Resource Locator (URL) requests from the browser and fetches resources from the network.
* V8 is Google’s lightweight and fast JavaScript interpreter .
* WebKit is a rendering engine that lays out web pages.
* The Display Backend is responsible for rendering graphics, creating widgets and rendering fonts. The following four components make this up:
  + Views is the framework for custom UI.
  + Windows Template Library is for GUI creation.
  + GDI is a graphics render, though Chrome uses it solely for its Windows-native text rendering capabilities.
  + Skia provides graphics rendering, excluding most text.
* libXML is used to parse XML
* Plugins are tools used to provide extra tools that are not included within the browser

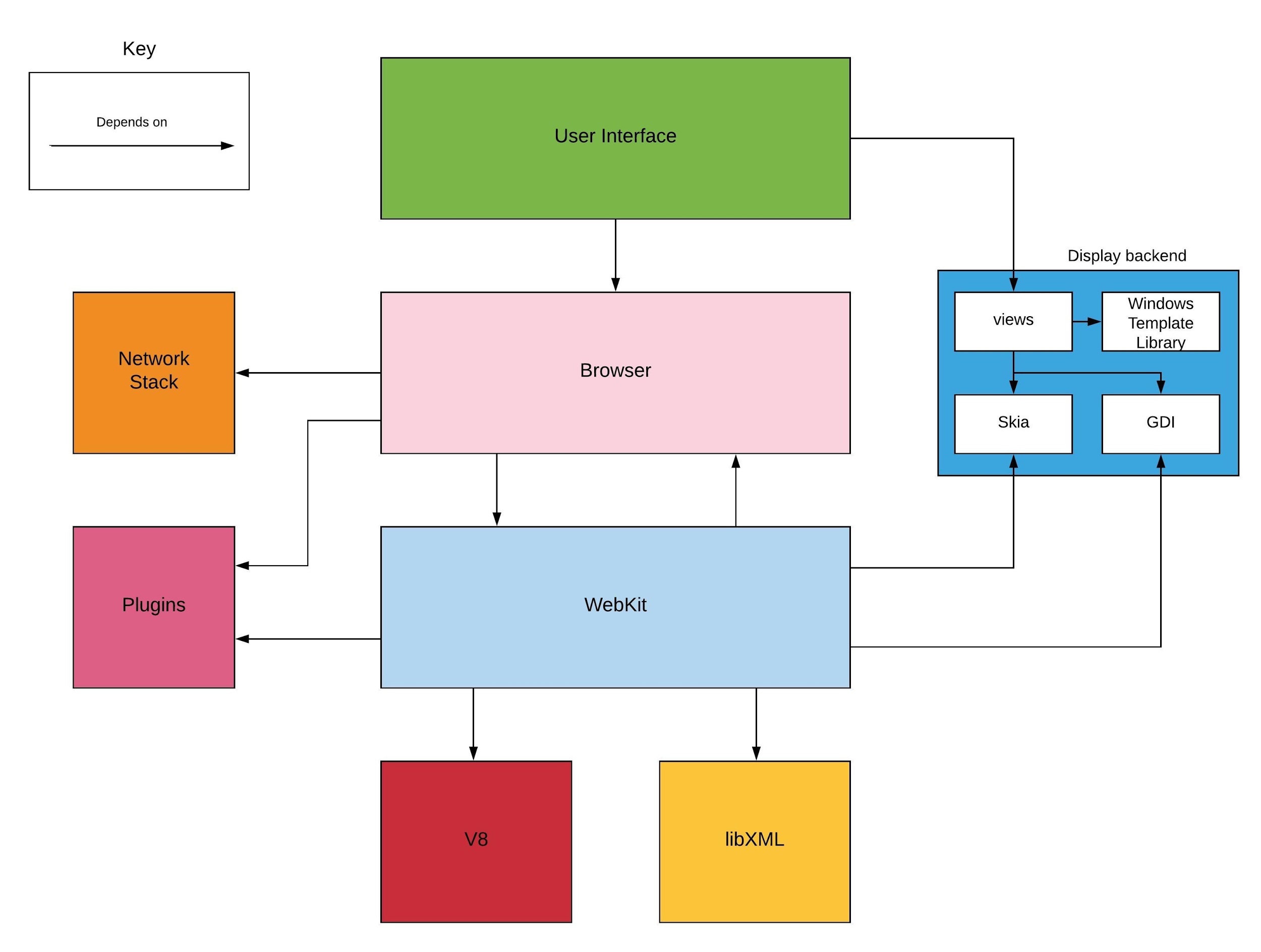


Figure 1: Google Chrome conceptual architecture

(from Reference 1)

### Extensions

Chrome extensions are small plugins with single purpose that changes functionality of chrome to fit users needs, usually they are build with html, javascript and CSS. Its key components are: manifest file, some HTML pages, optional JavaScript, optional images and other files extension needs.

* The manifest file, ‘manifest.json’, with name, version, description, icons, browser actions and other important information.
* The Background page, ‘background.html’, an invisible page contains logic of the extension, like JavaScript code for the behavior of the extension. P[ersistent background pages](https://developer.chrome.com/extensions/background_pages) are always open and running and [event pages](https://developer.chrome.com/extensions/event_pages) are opened and closed as needed.
* It can have simple HTML page like a popup. Any extension can have options pages, which helps users personalize extensions behavior.
* A content script is some JavaScript that executes in the context of a page that's been loaded into the browser.

Also, extensions can also use Chrome-only APIs (often called *chrome.\* APIs*) that allow tight integration of extension into browser.

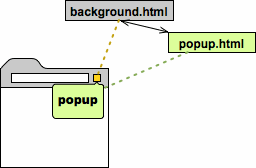
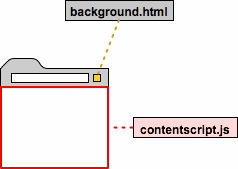
** **

Figure 2: Popup and content script extensions

Those are examples of popup and content script extensions, as we can see they both have background page. Popup is not a copy of the background page but it can reference to it as needed.

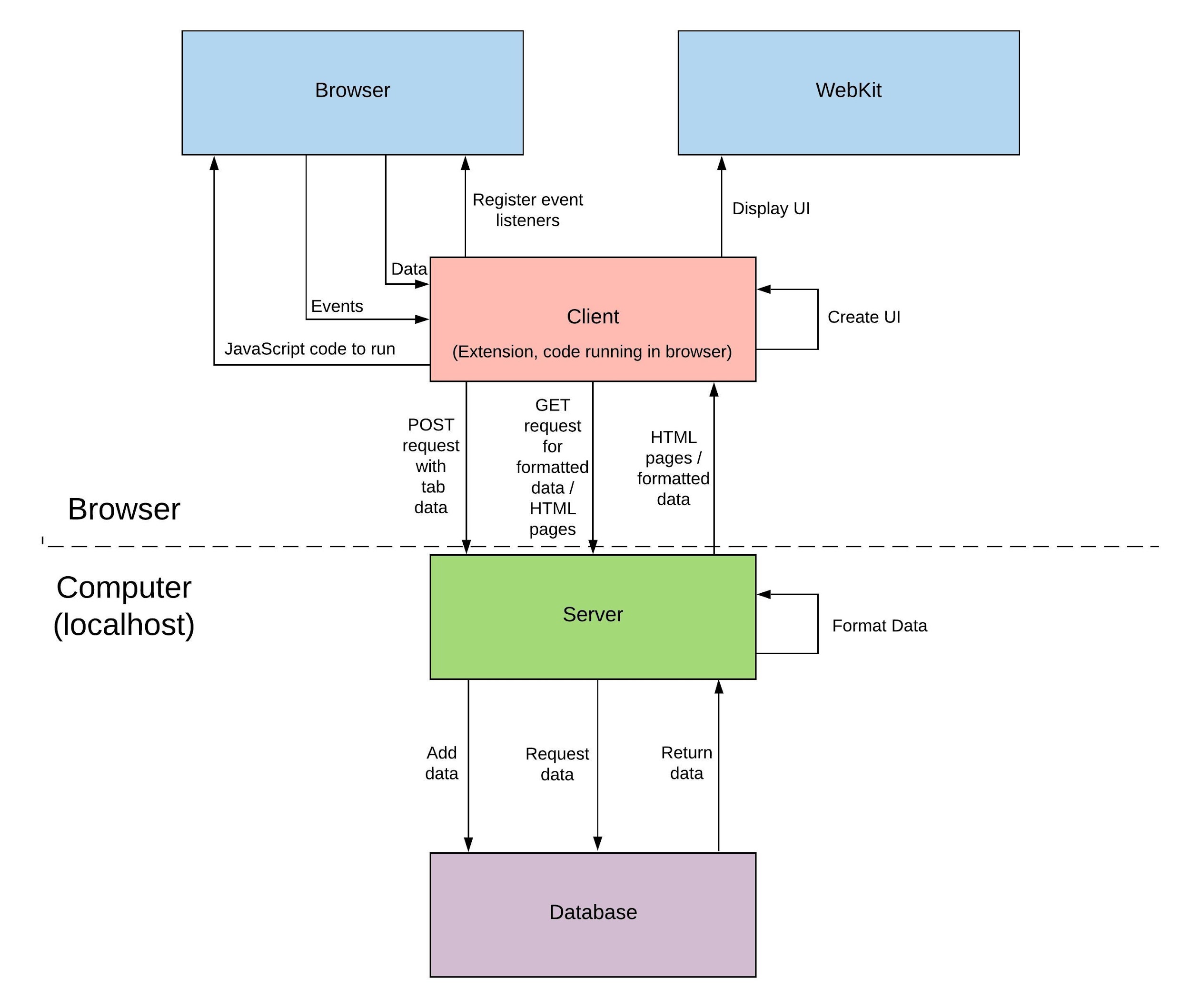


Figure 3: Conceptual architecture of our plugin

## Architecture Analysis

The dominant architectural style of this system is semi-strict layering, and also uses the component architecture. Strict layering means that layers are vertically dependent and components can only interact with layers directly underneath it. In Chrome’s high level architecture we see top-down flow, high level of abstraction and low and reusable layers. For example, the network and rendering engine do not interact with each other directly, but use browser component communicate. Component architecture, on the other hand, emphasizes the separation of concerns between the components by grouping individual pieces of larger components into separate, self-sustaining micro systems build from the smaller components. On the lower levels of the architecture we could notice multi pattern design.

This architecture contributes to many quality attributes. For one thing, it contributes to the maintainability of the system by having separate and strict communication rules. This helps make the code easier to make changes to, and maintain its functionality. Another quality this architecture improves is extensibility. Each specific component can be replaced by another comparable component without changing the entire system. This can help end users or third party developers extend the system to make more diverse and nuanced tools. Security is also improved with this system. By compartmentalizing the system into smaller components, if one component is compromised, only that single component is compromised, instead of the entire system. The other attribute affected the most by this architecture is portability. Here, the browser is the only component interacting with the operating system, which allows Chrome to have different versions of the browser for different operating systems, only one component (i.e. the browser, or a few components) needs to change.

Furthermore, the layered structure abides by the Law of Demeter, which says that units should have a limited amount of information in the units next to it, and also to only talk to the units near it. This helps follow the single responsibility principle, which also contributes to many of the qualities already listed. In addition, Chrome has a reasonable amount of cohesion, since the components do not directly depend on other levels, and the communication between components that there is are limited.

Chrome is multi-process architecture system, meaning it keeps each web site separate from one another and from the computer. Since rendering engine became such a complex system that sometimes faces malicious code from the web, if all the websites are rendered in the same place, the system would collapse if even one of the pages crashed. Also, it would mean that all the elements are competing for the CPU time on single thread which could crash the entire browser, making it unresponsive to the user. However, Chrome avoided this problem by making sure that each website, browser, and plugin gets its own rendering process which makes program more secure and stable.

There is a very clear and clean way for plugins to interact with the browser, thereby extending the software easily. Plugins are not part of the browser, nor are they part of rendering engine which makes it more stable and lets plugins have their own process that is destroyed after closing the tabs using that plugin. Since each plugin runs on its own and interacts with the browser and rendering engine through the interface, it does not affect the work of the browser if it crashes.

## References & Citations

1. <https://archrometects.files.wordpress.com/2009/10/assignment-01-conceptual-architecture-of-google-chrome-archrometects.pdf>
2. <https://www.computerhope.com/jargon/c/chrome.htm>
3. <https://developer.chrome.com/extensions/overview>
4. <https://blog.chromium.org/2008/09/multi-process-architecture.html>
5. <https://en.wikipedia.org/wiki/Law_of_Demeter>