Components Identified:

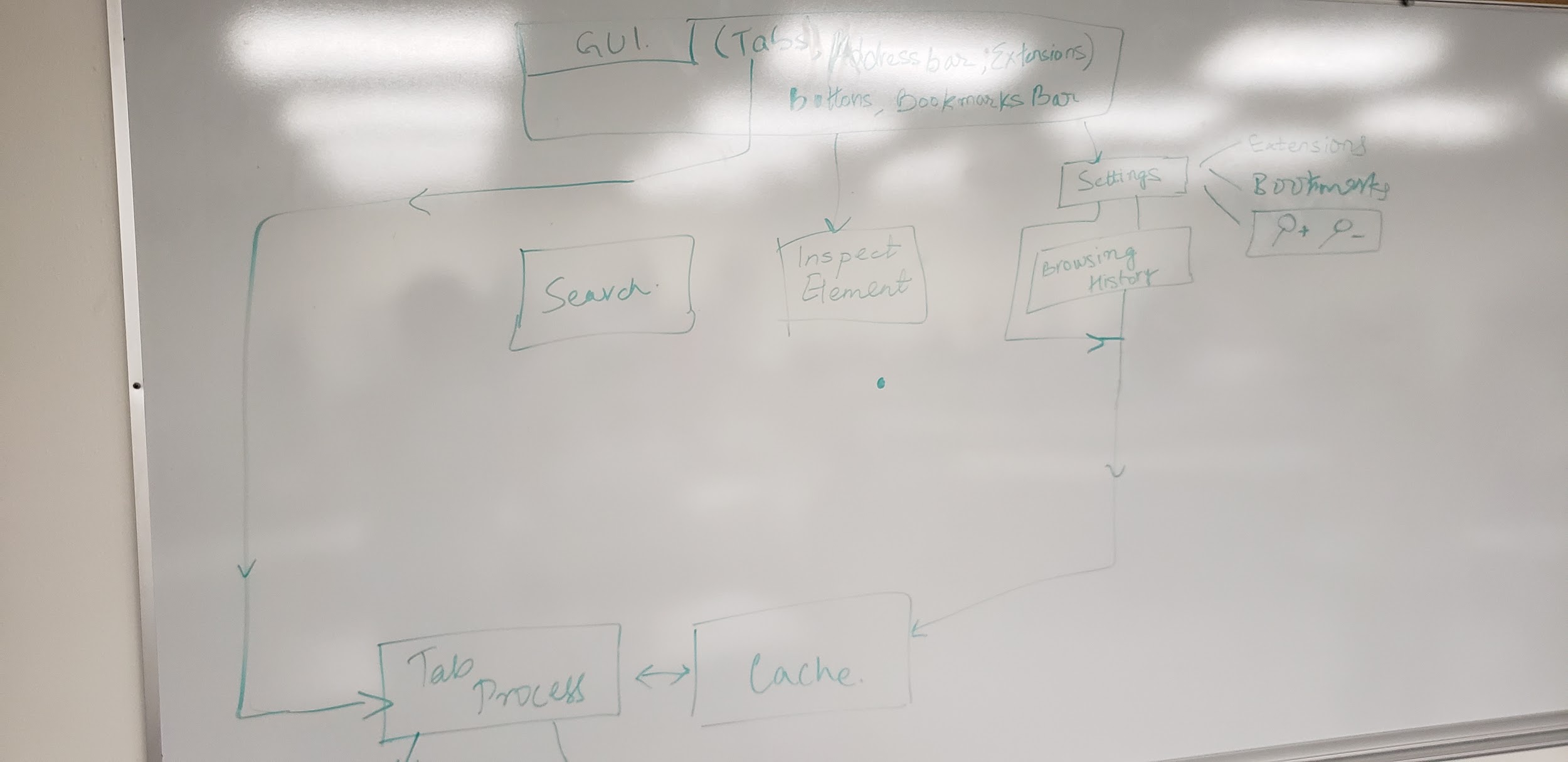
1. Caching (Data storage) (Our team)
2. Tab Process (Browser state)
3. Web Search
4. Inspect Element
5. HTML Parser
6. Bookmark Bar
7. Settings
8. Browser History (Our team)
9. Forward and Back Button
10. Browser Window/GUI(?)
11. Window Buttons/Frame
12. Extension Manager



Component dependencies and interconnection:

Many of the components were in reference to the corresponding GUI elements for their actions rather than the process itself. For that reason, there were many discrepancies between the process requirements and the dependencies shown in the picture below. So, here is a list of dependencies required by these components:

1. Caching and tab processes are independent processes with no dependencies.
2. The HTML parser and inspect element tool requires the tab process for the web page information to parse. Web search simply needs the tab process to provide information to.
3. The browser history, settings, and extension manager components all require the caching process to access saved data (while extensions may require various dependencies, the manager does not).
4. The forward, back, and refresh buttons provide updates through the browser history components and also require the web search component.
5. The windows button and frame can be considered outside of the cope of our program, but our main GUI still utilizes it as a dependency.



Suggested Design: Layered Architecture

Layered Architecture

* Front-end components for our user interface are neatly separated from the back-end components to make them work.
* Clean division of labor split between functionality and data. Back-end components hold the data utilized by the front-end to perform tasks for the user.
* New features can be built on top of the strong foundation with little coupling

Repository

* While many components need access to the caching and tab process components, not all should have access.
* Exposing the back-end so cleanly to the user-interface components or purely functional components will lead to security risks and bugs.
* Adding new features will greatly increase confusion as all of our components dip into the same resource for sharing information.

Client-Server

* This could be reasonable considering the categorizing of tasks between being purely interface/front-end or being purely functional/back-end.
* However, many of our components have very low coupling and do not need any more limitations on dependent data.
* Most importantly, the web browser itself is a fully local process, so the separation of local data based on a client-server model is purely artificial and might lead to a negligible benefit for the hassle.

Pipe and Filter

* A browser process is far from iterative or transformative, the processes one would usually apply this architectural pattern.
* Where the user interacts with the interface is also where he would like to see the changes. Running through so many components in the system for a potential small change the user would like to see is over-engineering the issue.
* If one were to use this route to make a web browser, a custom architecture is better suited as more effort must go into decoupling the system. Many components have few dependencies they must reach without going down their entire pipeline of dependencies first or afterward.