ColorFriendly uses several different components to accomplish its goal: the popup, the background script, and the changer. Popup, the first component and the component that the user directly interacts with, is comprised of an HTML file and a Javascript file. This component is where the user can configure the extension and that forwards the user's settings to the backend logic that actually does the color adjustments. There are event listeners in the Javascript file to ensure that anytime the user changes a setting, it is sent to the backend logic to make any necessary changes immediately.

The popup's Javascript file is also home to the storage implementation. All user settings are stored using the Chrome.storage.sync API, allowing for persistence of all settings between runs of Chrome. Specifically, the type of adjustment the user has selected, as well as if adjustments should be enabled at all, are stored. Changes to settings are stored immediately, while settings are read every time the popup is loaded (when the user clicks on the tool bar icon). This is to ensure that the most recent and accurate user settings are displayed to the user (so as not to induce user confusion or frustration) and are applied to all and any pages (so what the user sees in on his or her websites is what the user expects to be there).

The background file, background.js, is loaded when the extension is first run. This background script is the link between the frontend that the user directly interacts with, the popup, and the part that does the color changing, the content script. When the user changes a setting, it sends the new setting to the background script, which sends it to each content script that is running in each tab. The content script cannot talk with the popup directly, so the background script is a medium that allows for this communication to happen. The background script was designed to be fast and light, as its only functionality is to forward communication between the frontend and the backend.

Each content script iterates through all of the elements in the page and determines if the color of that element should be changed, and if it should, changes it. The content script uses getElementByTagName, using "*" as a wild card, to get an array of all of the elements on the webpage. At this point, the content script can easily iterate through the array, changing any colors as necessary. To change the colors, the content script leverages Tinycolor. TinyColor is an open source library that allows for color values to be saved as RGB, hexadecimal, or HSL values, precisely alter those values in a variety of ways, and switch between the different color formats.

Each tab has its own content script injection. This feature means that ColorFriendly can intelligently change the colors based on user events, such as page refreshes, new tab creation, and ColorFriendly settings changes (for instance, if the user switches the desired adjustment, the content script will know to switch on the page as well). All of these changes happen automatically, without the user having to manage the changes.

Originally, the design was to provide a patch for Chromium that would provide an API for receiving all of the elements on a given page of a given type. For example, if ColorFriendly wanted to change the colors of all of the links on the page (all <a> tags), the extension could ask Chromium, and Chromium would return all of the IDs of all of the <a> tags. We spent about two and a half weeks working on the API, but we encountered several issues in getting it to run. We wanted to avoid falling into the escalation of commitment trap, and decided that the best solution would be to move on and just get the elements from the DOM in the extension itself. This contingency plan has been thought about since when we first proposed this project, and it works nicely as it is.