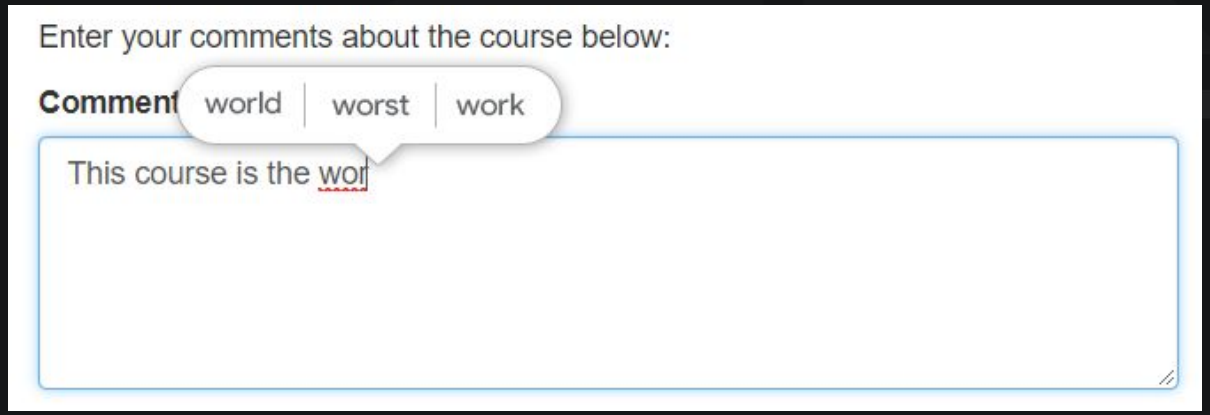
Google Chrome Proposed Feature Presentation

By: Brynnon Picard, Dongho Han, Sam Song, Bradley Kurtz, Alex Galbraith and Roy Griffiths

Proposed Feature

- Provide the user with options of similar words or the correct spelling for the current word they are typing
- Also predicts which words are most likely to be used after the last word they typed
- Similar to the keyboard suggestion feature used in Android or iOS



SKYNET

Functional Requirements

- Interactive popup dialogue box when editing text input fields, giving a max of 3 suggestions
- Spell checker suggestions
- Similarly spelled word suggestions
- Tailored predictive text suggestions
- Cloud sync for predictive text

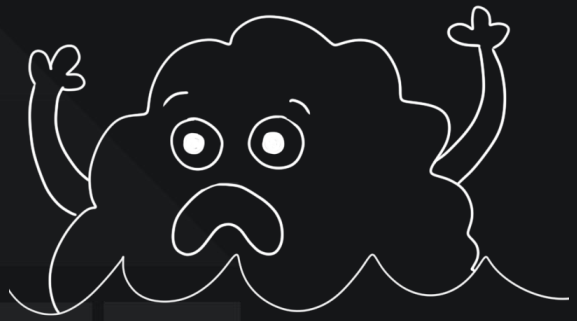


Figure 1. Cloud Syncing

Non Functional Requirements: Performance

- Popup should be populated with suggestions and display < 300ms after user input. This is very lenient because the user doesn't really care if it takes a while as usually they only will use the popup if they have paused to consider a word.
- Should not affect the user's ability to perform normal Chrome functions, i.e., should not cause input lag.

Non Functional Requirements: Security

- User security should not be compromised in any way by the new feature.

Non Functional Requirements: Management

- Functional implementation complete in < 12 days by two people. This is not a major feature and as such should not take too long.

Non Functional Requirements: Portability

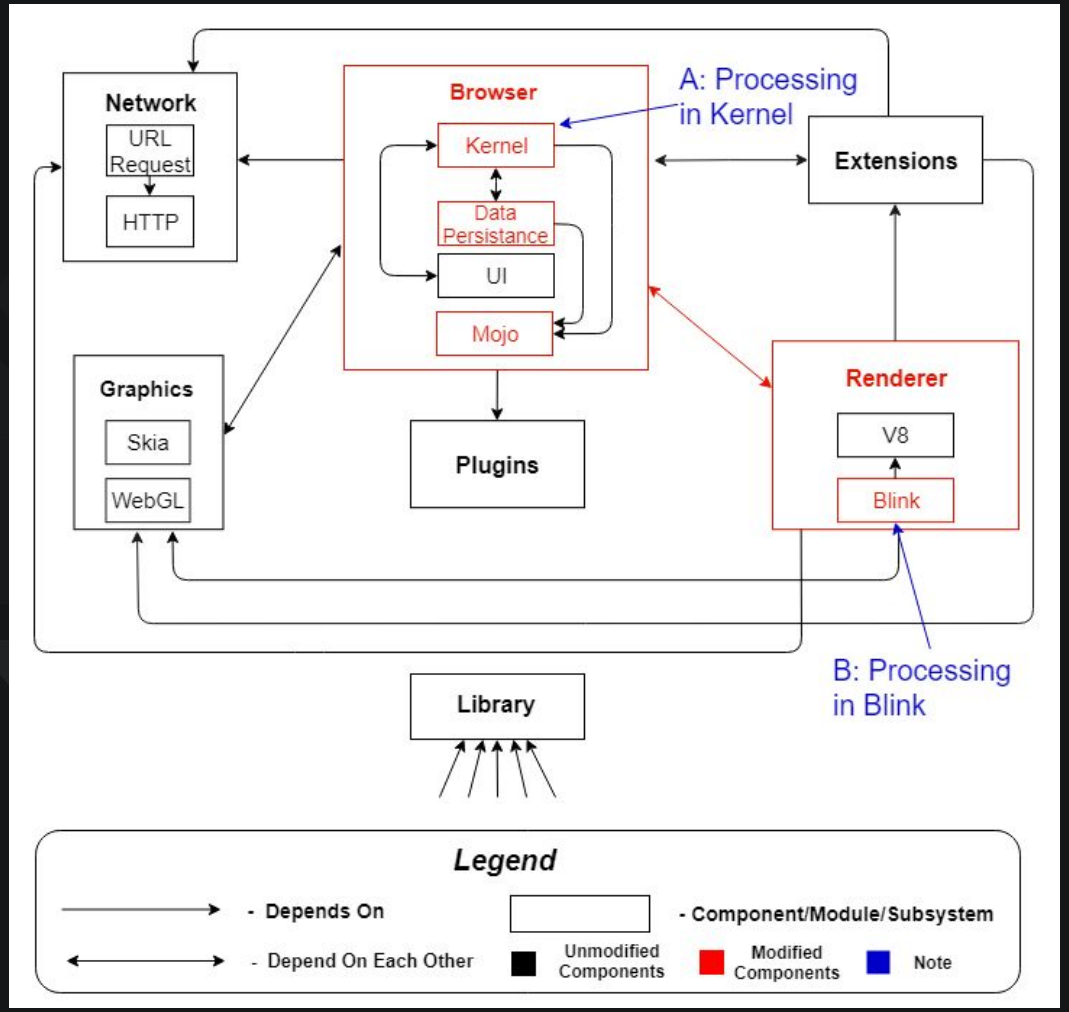
- Works on desktop versions of Chrome (Windows/Mac/Linux/Chrome OS)

Non Functional Requirements: Interoperability

- Customised user data store as XML

Approach Overview

- Two approaches, one performs predictions in the Browser, the other performs predictions in the Renderer.
- Both approaches implement the popup in the Renderer via Javascript injection.
- Both implementations store the user dictionary in the Browser's data persistence.
- Both do prediction processing asynchronously.



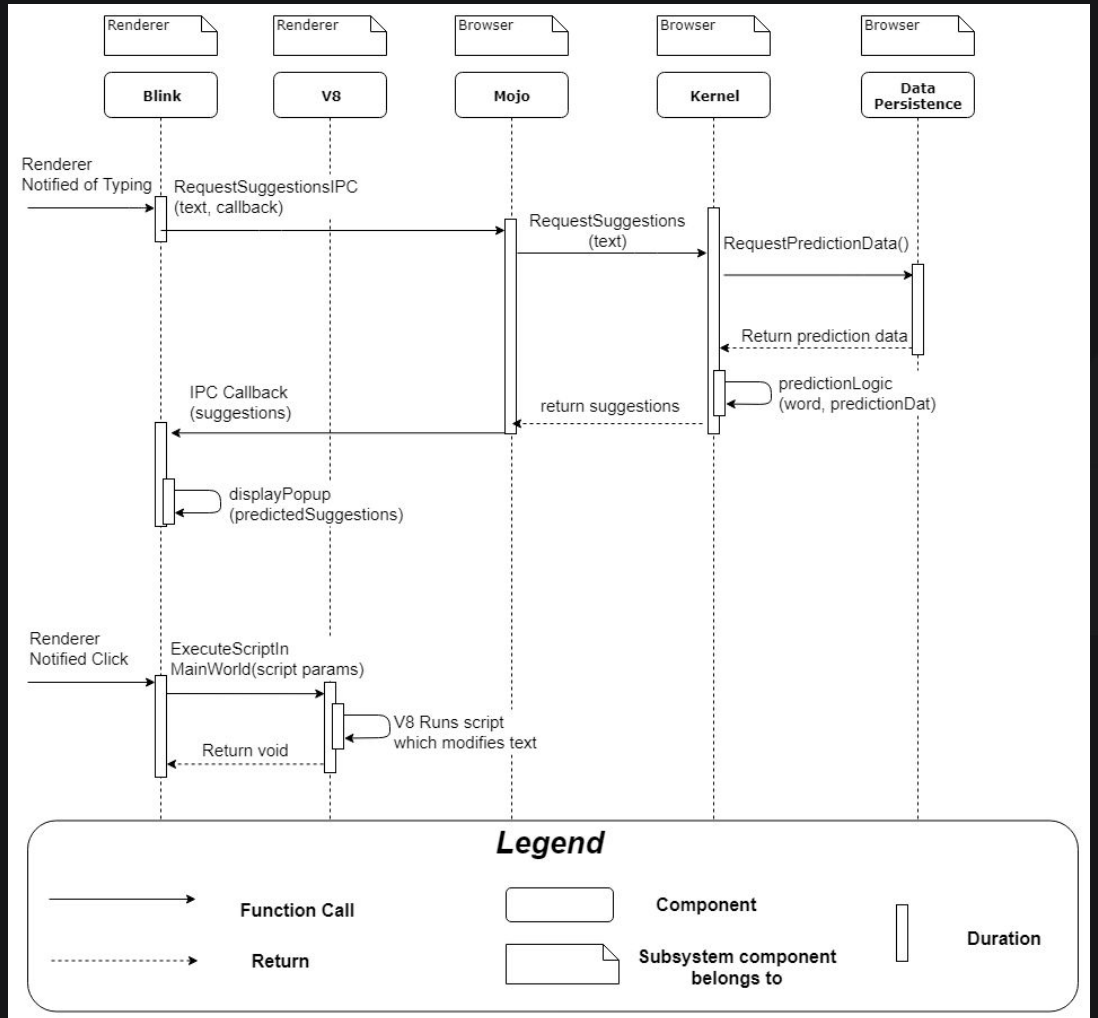
Approach A

- Implement popup dialogue and modification of typed text in the Renderer subsystem under Blink. Implemented as Javascript injection.
- Implement data storage and processing for tailored predictive text in the Browser subsystem. Browser would store data via the Sync services which would also perform cloud sync and maintain security.
- Renderer requests predictive text data from the browser and sends info on typing to the browser for processing.
- Renderer pre-requests and caches commonly used sequences to speed up prediction.
- Renderer batches changes to be sent to Browser
- All processing operations run async so that user experience is not affected.

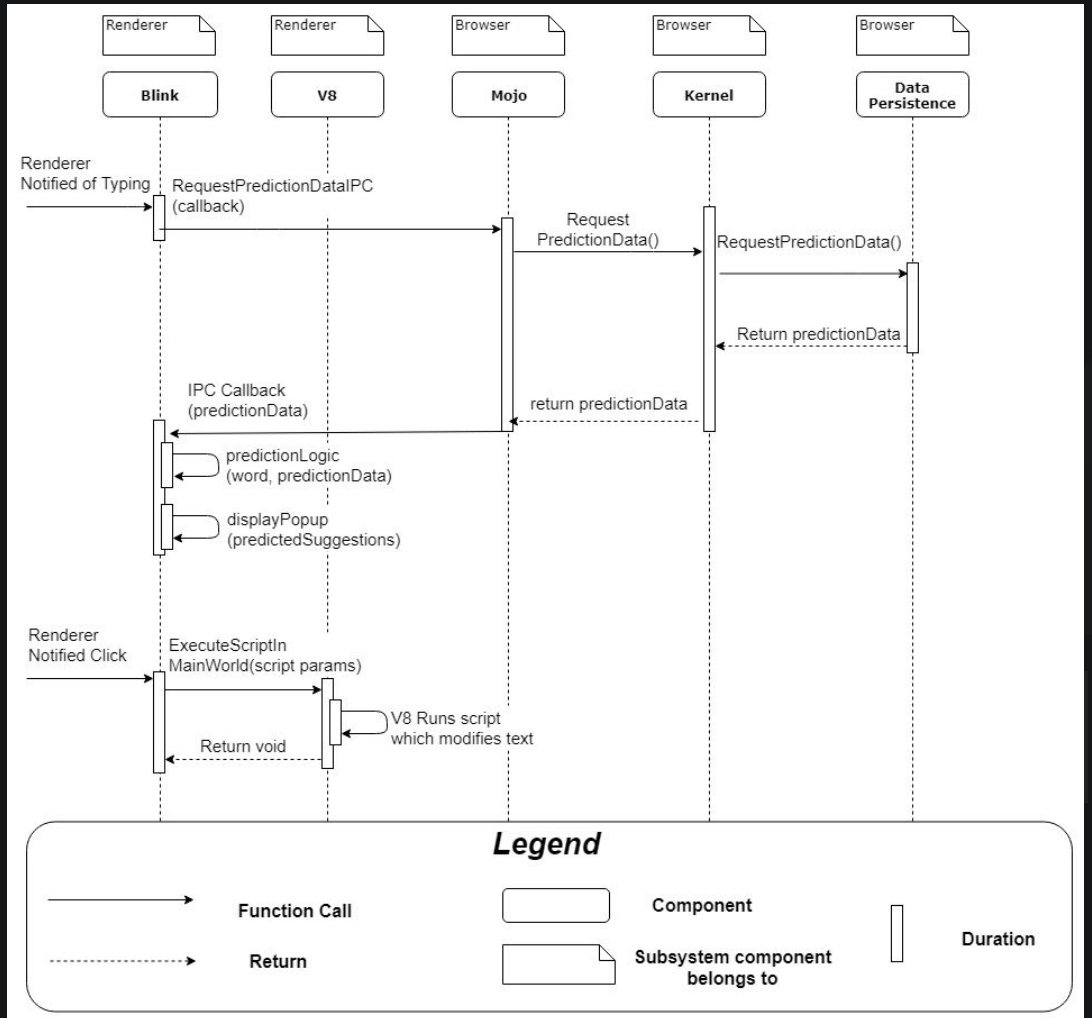
Approach B

- Implement popup dialogue and modification of typed text in the Renderer subsystem in Blink. Implemented as Javascript injection.
- Implement processing for tailored predictive text in the Renderer subsystem in Blink.
- Renderer requests predictive text data from the browser and caches it. This would be done via Chrome’s Sync services which would also perform cloud sync and maintain security.
- Renderer batches data changes to be sent to Browser
- All processing operations run async so that user experience is not affected.

Sequence Diagram - Approach A



Sequence Diagram - Approach B



SAAM Analysis Part 1

Scenario:

- Once user stops typing for specified time (> 0.5 seconds), Chrome suggests a word based on the last typed word
- When user makes typo, Chrome suggests correction as central suggestion.
- Predictive text will suggest most favoured words.

Stakeholders are:

- Chrome users,
- Development team,
- Google, and
- Google shareholders.

SAAM Analysis Part 2

- Approach B will have higher performance as it requires less IPC communication.
  - Not really a major issue as the user doesn't need instant feedback as long as their normal input is not affected
- Approach B will be faster to implement and easier to maintain since all the code will be in one localised place.
- The Blink sub components of Renderer already contains the SpellChecker module thus it makes sense to add the predictive text in the same place as in Approach B.
- Both approaches allow for the processing to be run async, preventing user experience from being affected.

SAAM Analysis Part 3

- A compromised Renderer in Approach B might be able to access predictive text data, e.g., word usage frequencies. This is a large violation of our NFR centered around security.
- Approach B requires each Renderer to maintain a cached copy of the predictive text data which, depending on how the prediction is implemented, could be a fairly large (e.g., dictionary + set of neural network weights).
    - Chrome already uses RAM fairly indiscriminately
- Both implementations will have to be wary of portability, but should rely entirely on other abstracted code in the codebase, so portability shouldn't be an issue.

SAAM Conclusion

- While Approach B will be faster, it directly violates one of our NFRs, specifically our security requirement.
- This violation outweighs the additional performance and development costs. An implementation that exposes security risks would be detrimental to users, and due to its effect of public opinion, would be poorly regarded by shareholders and Google.
- Ultimately the additional performance and labour costs in A are outweighed by increased security, thus we have chosen A as our best approach.

Concurrency/Team Issues

Concurrency

- Our feature runs concurrently with the main rendering thread.
- After a “typing pause” is detected, a dedicated prediction thread is stimulated to generate a prediction.
- This thread is given a callback to the Renderer to inject the popup Javascript into the page.

Team Issues

- We believe that the optimal team to work on implementing this feature would be the Browser team, since the approach we chose uses the Browser heavily.
- The developers working on this feature may want to communicate with the developers who worked on the Android keyboard to discuss how best to implement it.

Lessons Learned and Limitations

Limitations

- No reference architecture for typing suggestion systems
- Only a couple of subsystems would make sense to use for this feature, so there were limited ways to implement it
- Limited information available on how to perform SAAM analysis

Lessons Learned

- In a well-developed architecture, implementing new minor features shouldn’t require major architecture redesign
- There are always trade offs in the possible implementations of any software feature, no one method will be the best in all cases

Conclusion

- We are proposing a feature to provide the user with word suggestions as they type.
- We've created two possible approaches to our proposed feature for our extension.
- We decided to go with approach A due to the security risks exposed by B which outweigh the additional performance and development costs.