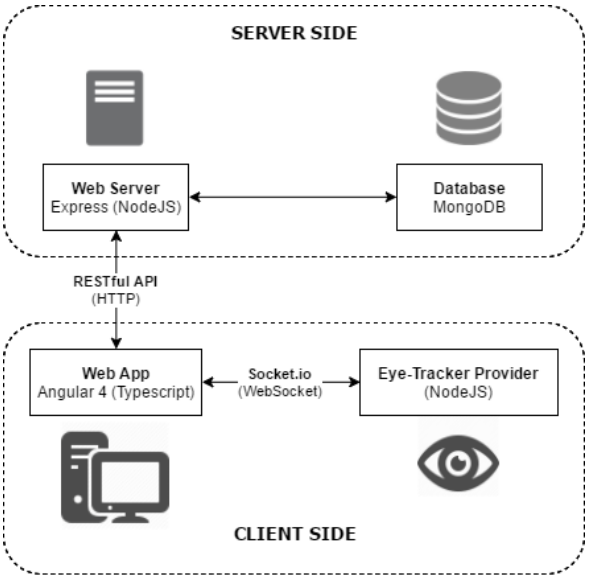
Next Steps Proposals

Eye-tracking augmented website - Clément Jacquet

CURRENT ARCHITECTURE



The app architecture consists of 4 independent units that communicate over the network.

Server side, the Web server, built with the `ExpressJS` framework (`NodeJS`), provide both the Web app (route `'/'`) and a RESTful API that enable interaction with the `MongoDB` database (route `'/api'`).

Client side, once the Web app is retrieved from the Web server, it runs in the user's browser, using the new framework `AngularJS 4`, and receives eye gaze location on the screen through the eye-tracker provider, a `NodeJS` program running locally and that is using eye tracker device API and drivers to retrieve information. In the future, there couled be several eye-tracker providers connected that are not forcibly running on the user's computer.

DONE

1. Loading English texts from the MongoDB database
2. Detecting which words are "read"

   For the moment, once a fixation is nearby a word, this one is tagged as read, without any certainty that it was truly read.

3. Words "read" are saved in the database

   Each word read is sent to the Web server that stores it with a timestamp that indicates when this word was read in the database

TO DO

1. Reading behaviour detection
   — Detect whether the user is reading or not

   If three fixations are located on a same horizontal axis, we can infer the user is reading

— Detect which line is being read

The eye-tracker device is not enough accurate to indicate which exact line is currently reading. But by assuming the user is reading a paragraph from the start to the end without skipping words or without going back to already read words, it is easy to infer which line is read using a simple counter.

— Decide whether a paragraph is read or not

If a user reads 90% of a paragraph's words, can we infer the paragraph is read ? What if he only reads 50% ? 30% ? This threshold can be selected by hand in a first time, and then by using Machine Learning classification algorithms, in a user independent or specific context.

2. Text display settings
   — Changing the font, size, color or line spacing of the text
   — Easily usable settings input

   Using sliders or spinners to change easily the previously described settings

3. Debug pan & tools
   — Toggling word coloring after being read and reticle display

   A word has initially a black font color, but with a debug mode it can turn to red if the word is tagged as read by the app. In debug mode, the reticle, that indicates where the user is looking at, can be displayed as a red circle. Those debug features are already implemented but can't be turned off for now.

   — Showing paragraph reading advancing

   For example, showing a little gauge at the right of the paragraph, showing a percentage of words read over the necessary read word count for deciding that the paragraph is read.

PROPOSALS

1. Reading test with explicit feedback

   A new user will forcibly already have a certain knowledge of English vocabulary. A reading test can be done after the sign-up operation to infer what vocabulary the user already knows. The test could consist in showing a text to the user, and let him rate how easy the reading was to him.

2. Elliptic eye detection (rather than a circle)

   The eye's area of reading is actually more horizontally stretched ellipse-shaped than circle-shaped. Calculations are easier with a circle, but are probably less accurate. Adopting this new shape will eventually improve user experience.

3. Statistics analysis in another page

   Simple statistics can be computed, as how many times a word has been read, which words are the most encountered. Then more complex statistics algorithms could be implemented, such as analyzing which words have not been recently seen and should be reviewed, or as assessing the evolution of the user's English vocabulary.

4. Frequent optimized reviews

   A system of daily or weekly review to train the user.

5. Real-time debug monitoring

   Rather than showing statistics and debug information directly on the user's screen, a simple third-party app could connect to the user's reading instance and retrieve information that would be displayed for monitoring purposes.

6. Real-time definitions for new words

   When a new word is read by the user, it can be displayed in a side pane of the website with a definition, to let him know the meaning of this word. He would have then a list of new words he can check the meaning.

7. Definition shortcut

   By pressing a specific key on the keyboard and hovering a word with the mouse, the app could display this word's definition.

8. Data management tools

   A specific part of the website could offer an interface for interacting with the database in an easy and straightforward way. Such admin tasks as inserting a new text in the database are complicated if no interface is provided.

9. User authentication for multiple user handling

   A classic user authentication to provide user specific reading analysis.

10. Japanese texts support (or other languages)

    Rather than just offering English texts to the user, other languages could be supported, such as Japanese. In the case of Japanese, new problematics would appear such as splitting sentences into words. Such a task is easy with European languages like English or French as words are separated by spaces.

11. Agnostic eye-tracking device API (in order to handle more than only Tobii EyeX)

    For the moment the API is designed for Tobii EyeX devices. The API could become agnostic and provide an interface adapted for any type of eye-tracker, such as Jins Meme.

12. Firefox or Chrome extension that can be applied to websites (complex)

    The user could use this reading analysis not only on the provided website but also on other websites. He would use a shortcut and then select a paragraph that would be launched in a specific pane optimized for reading analysis.