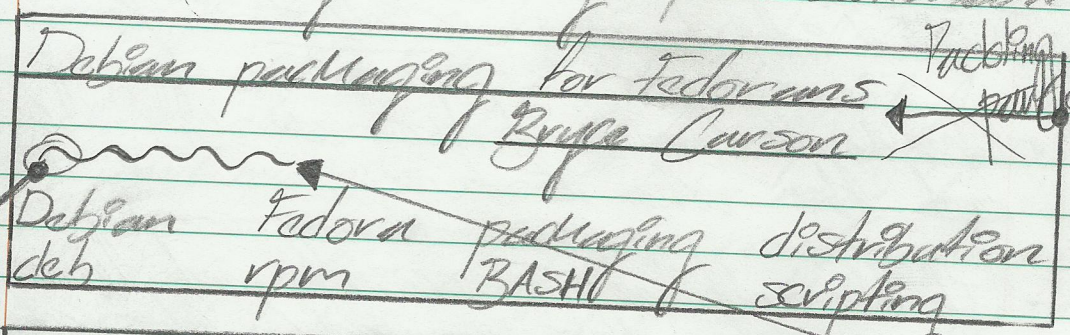
# JavaScript Idea One

Article card tag and synopsis animation:




Interaction causes article description to appear instantly.

Debian packaging for Fedorans
Bryce Carson

This posting describes my struggle with Debian packaging as a Fedoran, a Fedora user. It is not a guide to packaging for Debian for Fedora-using developers, despite the title.

The post deals with `pbuilder`, a Python tool for deb packaging; as the tool uses virtualization to create reliable Debian software packages it should also work on Fedora. However, it does not without struggle.

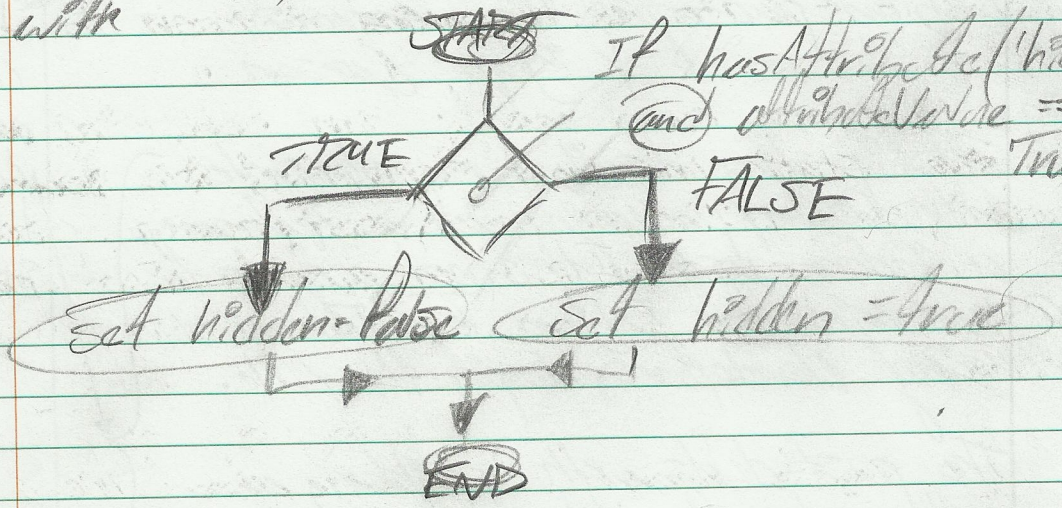
Debian deb Fedora rpm packaging BASH distribution scripting

Word count - (word count × reading speed / time) = READING (~3 minutes)

Pseudocode

for all cardSides (the class) attach an event handler to do the following when the "reveal info" box is clicked:

- Check if the card has an attribute (hidden) and the value is true;
- If the attribute is present and true, remove it; otherwise add the attribute with




Progressive revelation, or however GitHub calls it.

See design on pages 44-

# JavaScript Idea Two: site-search

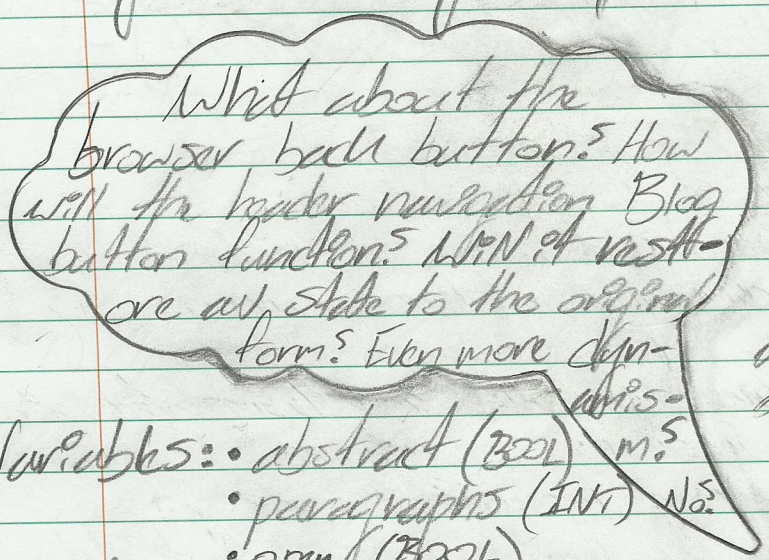
Visually impaired users, like Vanjo Smithson, may have difficulty navigating a website if the visual design is too minimalistic for too crowded. A minimalist theme places elements relatively far apart, with padding and margins more than may be absolutely necessary, making the nominal amount of content seem rather than a navigable sea. Density can be useful, but even good food like cake can be ruined if it is too dense, making it tough to chew.

Site search functionality with highlighting and an attempting static link (not highlighting the found content, only linking to the nearest heading or section titling) is a useful tool to quickly find specific content, or to see if the content is appropriately written.

For example, many blogs write about the endless features of Emacs, but few delve into it like marine welders; most reiterate the same basic information. Searching for that basic information and checking against the background knowledge the user has, can be a good way to assure the user they aren't wasting their time.

- Provide a search utility with radio buttons for "Text", "Code", "All".
- Query the document for all nodes matching the selected result type.
  Ex: document.querySelectorAll('.code');
- Grep the resulting array for the search term(s).
- Use the logical array resulting to subset the first array. Present the results to the user by following a presentation algorithm described separately.
- If JavaScript is not available the search interface is disabled (the default); on the standard platform, JS will enable the search tool.

# Dynamic Blog Entry Navigation Design




Events:
- onEllipsesClick
- onOpenClick

These two events are defined for each article card.

Variables:
- abstract (BOOL)
- paragraphs (INT) No.
- openf (BOOL)

Functions:

1: onEllipsisClickHandler(articleCardNode)

2: onOpenClickHandler(articleCardNode)

Additional idea: if you're reading a card and are enjoying it, you can continue reading from the last paragraph you previewed.

Options
1) Continue from last position [Scroll to node]
2) Read from beginning

Perf: colour change to a highlight, then fade.

Functions

onEllipsisClickHandler(articleCardNode) {

// Only the HREF is required

get URL from property of argument node if object or use URL directly.

get query component of URL and translate into an object

assign the increment of the "paragraphs" field to the object and update the query component reconstruct URL;

GET that resource from the server

assign the response to a variable

If that's 'ok' use the response.body.HTML to make a DOM node

Append the node as the last child of the argument node (article card)

If the next value is 'nil' then disable the button/hide it.

}

```
onOpenClickHandler(articleCardNode){
    // Procedure on the left can be abstracted.
}
```

Compose some functions before redefining those void functions. Consider more thoughtfully if these really are void functions. Perhaps they should return an object for network dispatch and not be so monolithic.

Associating variables with blogs: JavaScript or HTML data-? Attributes?

Ex: <button data-?paragraphs=13&article=false">

OR

```
let articleDataScope = (articleNode) => {
    let articleOpenButton = articleNode.
        getElementByClass('articleOpenButton');
    // Do the same for the dispose button
```

... I'm not designing the rest of this function now. The time investment is not appropriate.

The example (Ex:) ends here. See the next page for an abstraction overview and pseudocode.

# Dynamic Blog Pseudocode (not algorithmic)

- When a user clicks on an article background or open button,
  - Send an HTTP POST request for the article node (respectively).
  - Hide the other articles.
  - Reveal/include a back button.

- When a user clicks on an article ellipsis button the next paragraph is fetched, if available. A paragraph (or other sub-article (child node) content) is available if the previous article ... button click response returned a "Next" flag alongside the other (if any) response content.

- Upon the successful construction of the DOM by the window object, JavaScript (if and only if present) will insert the ellipsis and open buttons for every article. Additionally, JavaScript will hide the last two paragraphs of every article (leaving only the Abstract visible).

  - Essentially, JavaScript will convert the static design into the dynamic design, and attach event handlers to buttons and such.

Site S

Quest

1)

2)

Pseudo

2:

# Site Search (Client-side Only) Tool

Question: • how are query parameters used?
• how do query parameters persist across page navigation?

1) The user inputs a query into a text field and presses a "Search" button.

2) All text content of all pages is grepped, and the user is redirected to a search result page (static) which is updated with the client-generated search results.

Note: only if the ◉ All radio button is the selected search type are all pages content searched in entirety.

Pseudocode:

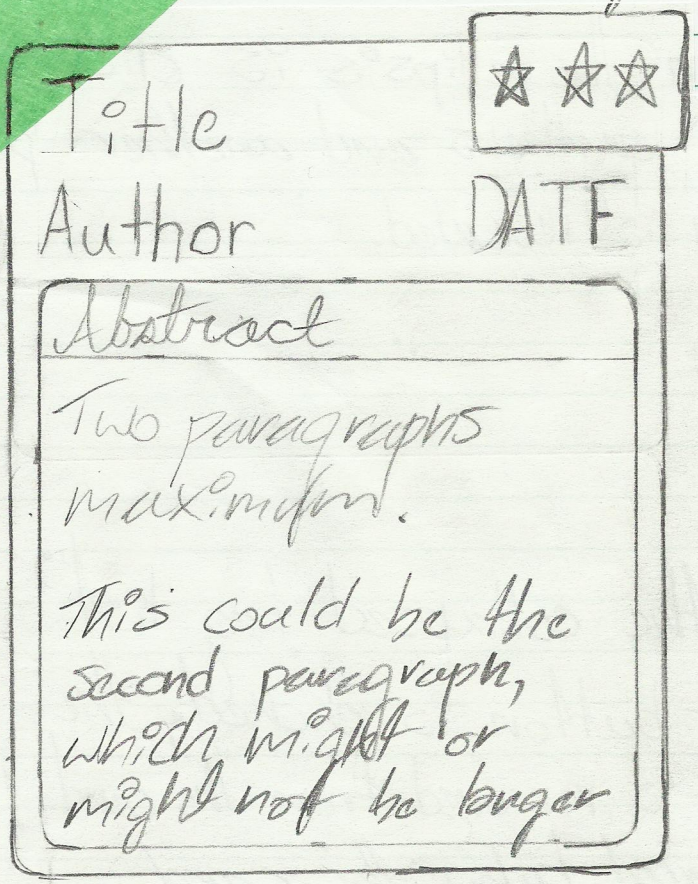
2:
- using fetch() retrieve the body content of all pages. According to the selected radio button, filter the body content.
- grep the filtered content.
- Insert generated elements onto page with the match and a link for all results.
- On the selected result page after navigation use the query parameters to scroll to the selected match on that page.

# Learner.js (Idea Three) Design

NOTE: The implementation of this feature could be varied. Z-ordering, "display: none", or transparency and new divs could be used.

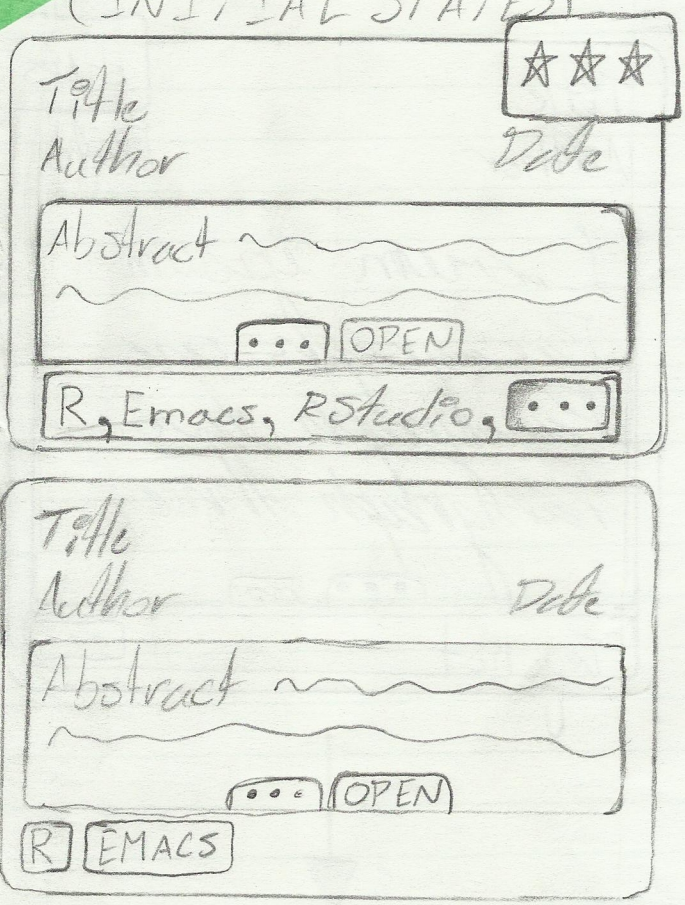
I recall reveal.js' name, but what does that library do again? I need to be reminded.

# Static Design Reference




Lorem ipsum: this is marginalia for the static design on the left side.

DYNAMIC
(INITIAL STATES)




Opening an entry will navigate to the full-page of that entry. Other entries are hidden, and the entry is fully expanded.