GNU social Frontend for v3 Technical Report
GNU social Summer of Code
August 23, 2020

Eliseu Amaro
Main mentor: Joshua Judson Rosen

2020-03-31 to 2020-08-24

1 Mentors page

Joshua Judson Rosen Eliseu has been a real asset to the project, and I really hope that he'll be able to keep working with us in the future.

He eagerly engaged and communicated both clearly and consistently with me and the other mentors, and both the user and developer communities; handled feedback adeptly, brought himself up to speed quickly on unfamiliar things, was able to work with pretty high levels of autonomy, did solid work in a high-value area, and just generally made people happy.

Summer of 2020 was a hard to time to be a good mentor, so it's been wonderful to have such an excellent pupil.

Phablulo Joel Eliseu has come a long way this summer. I'm proud of the progress he has done, specially considering the initial inexperience with web technologies.

He shared his work at each step of it's development. Thereby, the community could provide valuable feedback at each stage and take a part on the overall design.

I'm pleased with his effort to express the project values and goals in his work.

Diogo Peralta Cordeiro Being the programme organizer, I can confirm the dedicated hours of workload and the accuracy of what's written. In fact, Eliseu has put a lot of investment in this project and learned a lot along the way. We are genuinely proud of all the progress we've made with Eliseu's help. Eliseu had to submit code on the level of typical contributors without spoon-feeding, which was successfully achieved.

Eliseu has soon revealed a full understanding of his tasks and what had to be done. His commits have improved at a fast pace, both in message detail and code quality.

The GNU social community is thankful for his valuable contributions. It was a great summer of code. We hope he keeps contributing in his free time!

Abstract

With the development of GNU social v3 and thus, the desire of improving the whole project,a new interface becomes essential. The broadened scope on this iteration is a perfect excuse for a better and more modern approach while maintaining in touch with the community needs.

The main objective was to find novel solutions to the platform's versatile nature. To achieve this, a simple but thoughtful design was conceived under visual hierarchy, colour and modular design principles. Furthermore, novel approaches to templates, context information and semantic HTML made possible to restructure a block according to it's size and provide a cohesive experience even if the content provided comes from outside instances of the Fediverse.

2 Overview

Repository location `https://notabug.org/rainydaysavings/gnu-social`

Preface This project was carried out from 2020-05-04 to 2020-08-24 under the GNU Summer of Code 2020 program. My sincerest thanks to all mentors involved as well as the community which together helped me achieve something greater than I anticipated, through valuable criticisms and offering help when needed.

The author also wishes to thank the invaluable financial support given by The Freaks Club, without it none of this would be possible.

Contents

1 Mentors page **i**

2 Overview **ii**

3 Introduction **1**
3.1 Requirements and considerations 1

4 Methodology **2**
4.1 Delimiting design ideas 2
4.2 Iterative design 3
4.2.1 First design 3
4.2.2 Second design 4
4.2.3 Third design 5
4.2.4 Final design 6

5 Material and tools **7**
5.1 Modular design 7
5.1.1 Event-driven on a Component-based architecture 7
5.1.2 Meaningful HTML 8

6 Roadblocks **9**
6.1 Firefox's inability to render gradients properly 9
6.2 Checkbox hack and accessibility 10

7 Results and conclusion **11**
7.1 The timeline view 11
7.2 User panel overhaul 12

3 Introduction

The existing GNU social interface was introduced in 2010 and since then received little to no maintenance. With the advent of various interface plugins and additions, the interface became visually cluttered and complex. This created the need for a new interface. However, by working with an already established platform, considerations had to be made.

3.1 Requirements and considerations

Compatibility and accessibility GNU social always had a JavaScript-optional interface, considered an important feature within the community. This means that a comparable user experience should be given with or without JavaScript. Since the project follows the AnyBrowser campaign the same can be said within browsers, each page structure and design were carefully considered attending to each browser capabilities.

Federated networks such as those powered by GNU social imply inter communication between instances. Their representation in a cohesive manner is not always trivial as each instance might render them differently. A consistent and general solution to this problem needed to be researched as well.

A key pillar to the platform is it's accessibility, special attention to keyboard navigation, cultural differences and screen readers was given as well as following W3C Web Content Accessibility Guidelines.

The modular nature of the platform A key principle to a successful user interface is to provide effective visual communication. This can be achieved through a metaphor Marcus (1995), an abstraction which is capable of communicating clarity and scope.

There's difficulty in achieving this however, given the very nature of the platform. Each decentralized network is unique in their own way, with it's own set of plugins and features. The component based architecture provides versatility but also complexity.

Media queries limitations Responsive web design aims to make an interface aware of the viewport size and adjust rendered elements accordingly, currently this is achieved by using CSS media queries. Media queries can only target the viewport however, which means that each part of the interface can only respond to changes of the (global) viewport Wiener, Ekholm, and Haller (2017). This key limitation makes it extremely challenging to make responsive modules context aware.

Lack of container queries The Container Queries proposal enables web developers to style DOM elements based on the size of a containing element rather than the viewport. Unfortunately, it's still just a proposal WICG (2019).

Today, we're still trying to achieve responsive and modular web design with the wrong tool, Media Queries. In order to achieve truly modular blocks one should be able to write-once and use it anywhere Curtis (2010). However, in the present day, if the layout changes a developer needs to update all of their media queries, it's not maintainable. The Element Queries project presents a novel approach that enables this proposal today Wiener et al. (2017), unfortunately, it's reliant on JavaScript, which doesn't align with our aim.

At the most basic level, modular interfaces prove to be an already complex challenge, with these limitations in mind there was a need to research clever ways to tackle it. This paper provides insight on the solutions found and how they came to be.

4 Methodology

4.1 Delimiting design ideas

Familiarity through consistency When working within component scope there is a need to communicate familiar functions and actions. This in turn helps the user understanding of the entire layout. This can be done through perceptual patterns or frames Johnson (2014).

Frames are in essence the repeated exposure to each type of situation, this builds a pattern in our minds of what to expect to see. When browsing the Web we build these patterns which are essential to efficient navigation, by exploiting this consistency in Web design the barrier of entry for new users is lowered.

Each component template should then take advantage of such frames, proving within each block a clearer overall picture.

Creating a visual hierarchy Web design is all about communicating visual information, understanding that people will see our designs in terms of relationships is crucial. One way to achieve this is with the creation of a clear and deliberate style guide that encourages a cohesive visual identity and hierarchy.

Written text is a common example of visual hierarchy, titles convey a physical and mental divide between blocks of text and paragraphs new takes on the ideas at hand Jones (2011). In the same way, components should be merged or divided from others depending on their relationships or lack thereof.

Grouping similar components while providing ways to focus on each one individually supports the notion of active and non-active elements. This divide mitigates potential issues in smaller screens, since the screen real estate is limited.

Typography Optimizing typography is optimizing readability, accessibility, usability, and overall graphic balance. This can be done through micro and macro typography.

Typography plays an important role in presenting a visual hierarchy Keyes (1993), this is at the simplest level done in chunks. As part of macro typography,

chunking divides continuous text into manageable units. With use of variable vertical spacing and contrast type tonal density, a clear structure and relationship is given. A user post for example, provided of it's content and username can be distinctively presented with the latter in bigger font weight and size when compared to the former.

Micro typography on the other hand focuses on the finer details of a typeface, aiming for differentiation or harmony of elements within a larger structure. The x-height is one such consideration, user content should be presented in a typeface with a large x-height since there's more area for each glyph to be rendered upon legibility is greatly improved.

Colour Colour is a very powerful tool, more so in fact in such a visual paradigm such as web design, with careful selection of colours the user can be manipulated to stay in that web page for longer or think of load times to be quicker even.

"Colour-induced relaxation has a direct effect on attitude, in addition to its indirect effect through perceived quickness. This suggests that site-design characteristics such as colour have effects beyond their influence on perceived quickness. (...) The results point to the potential effectiveness of a blue (rather than red or yellow) background screen colour in inducing feelings of relaxation and reducing the perceived download time." Gorn, Chattopadhyay, Sengupta, and Tripathi (2004)

These results point out the need for calming colours for a background, and such a colour is a soothing and relaxing blue. This seems also something unaffected by even culture differences, as seen by how dominating the use of blueish colours is across the globe Cyr and Trevor-Smith (2004). While limited by design such findings are of great interest, so the user perceives each load of a new page to be faster as well.

4.2 Iterative design

Creating a cycle Designs and prototypes were created as a way to set the key visual ideas that work and provide a good foundation for the incoming months work, these were presented to the community and their criticisms gathered. Designs were then refined and presented again, creating a cycle that would eventually lead to a consensus.

4.2.1 First design

An initial prototype focused on establishing the core system model ideas. Similar components were organized in groups and their scope clearly delimited.

Visual ideas Federated networks usually provide a number of timelines which essentially differ in scope. Since their content is visually similar, only differing in content, grouping them together made the most sense. The "timeline compo-

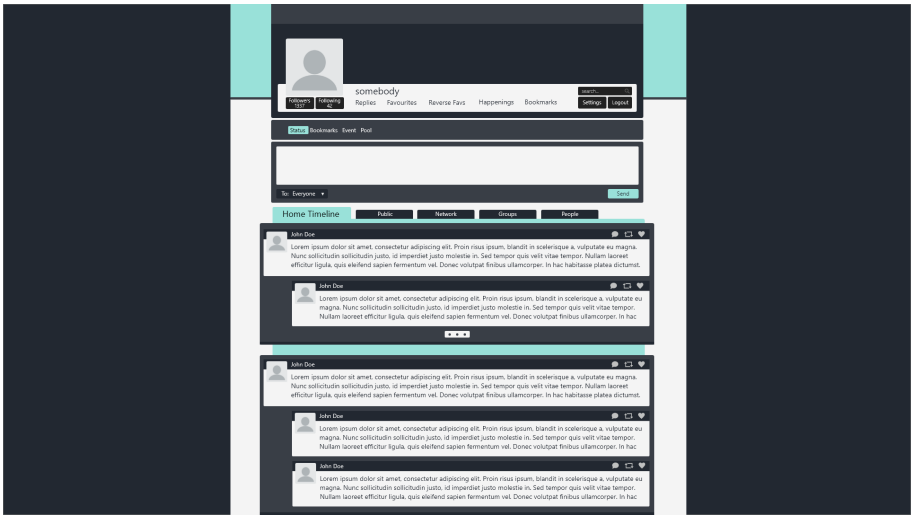

Figure 1: First iteration

nent" was then divided in tabs which visually represented the user location and provided navigation between each timeline.

Community feedback This prototype was generally well accepted as navigating through tabs was an already familiar idea. Most criticisms pointed out the dated design and poor colour choices. Moreover, the layout structure could work on smaller screens but the small buttons and text deteriorated the mobile experience.

4.2.2 Second design

Reacting to community feedback In reaction to the aforementioned criticisms the focus was now in achieving a more appealing visual style. Questions regarding plugins placements also created a need to provide a section that could incorporate them.

Community feedback Criticisms pointed out that the interface was too noisy, cluttered and slow to render. The new panel was a step on the right direction, even if most had noted the lack of flexibility compared to the original interface.

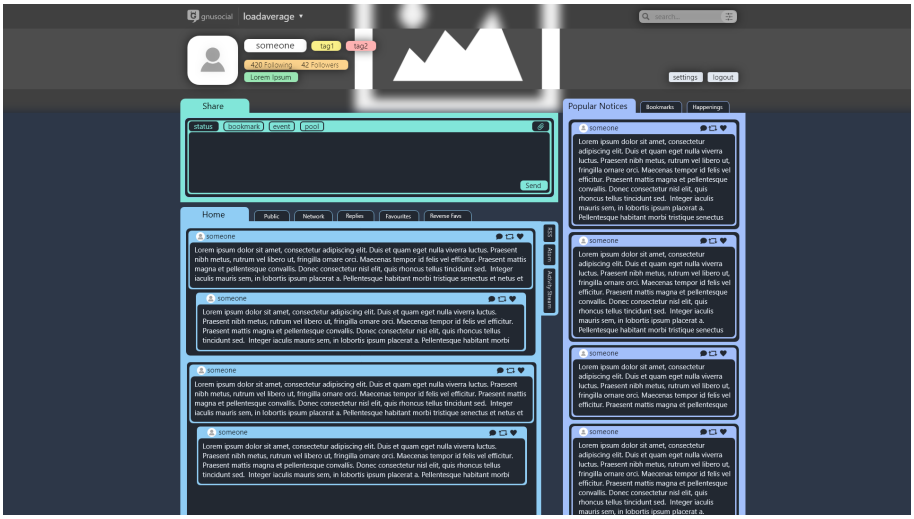

Figure 2: Right panel can be used for plugins

4.2.3 Third design

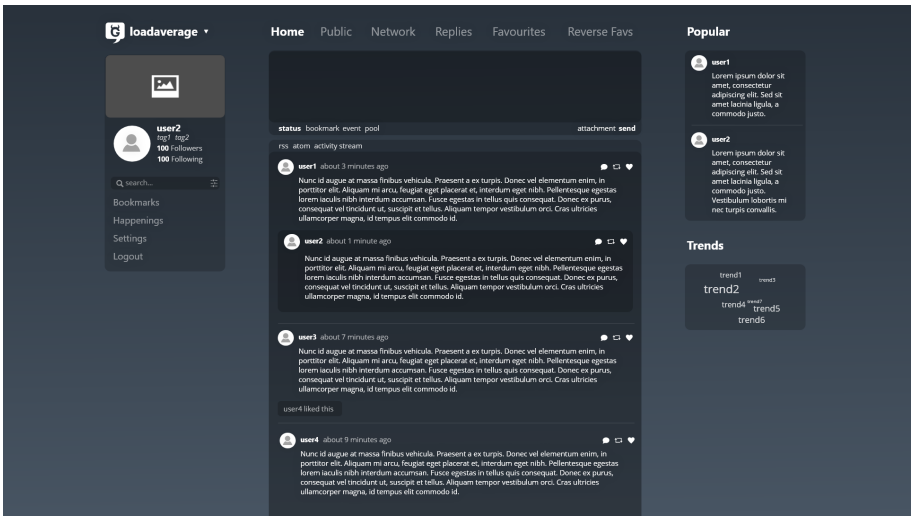

Figure 3: A design shift

Starting over, taking a different approach Symmetry, careful use of typography and minimalism were then followed, a design shift that would mitigate many of the previous issues.

A symmetric interface would ease future endeavours in creating a right to left writing style version, a feature also present in the original interface. The minimalist look would provide faster rendering of visual elements and let the interface work for the user and not against it.

Finally, by dividing the interface in three key areas the design could more effectively direct user flow and focus. The navigation presented on top of the

screen would instantly lay out information to the user of their location. Moreover, an input box placed right below it allows the user to share their status. The addition of a left panel proved that plugins could be placed in more locations as well.

Community feedback The community was pleased with the design shift, further polish was needed and mobile considerations had to be made before beginning implementation however.

4.2.4 Final design

Mobile considerations The concept of active and non-active views should also be applied to the overall design in order to mitigate potential issues with smaller screens and their lack of screen real estate. Both left and right panels are optimal candidates for passive views, which become only active when the user wants to. To achieve this both are accessible through contextual buttons such as an hamburger menu.

In a social network the ability to interact with others as well as their interactions should take center stage, thus becoming the default active view. The user text area would now take a promoted position and navigation limited to it's respective component, clearly delimiting it's scope.

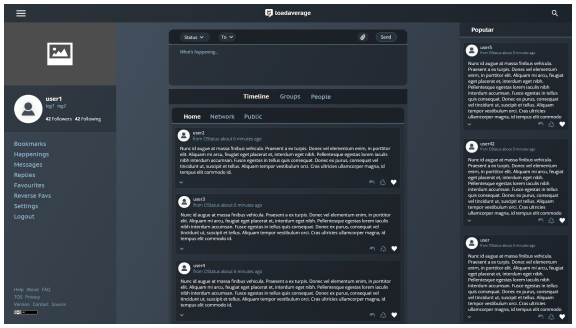

Figure 4: Desktop

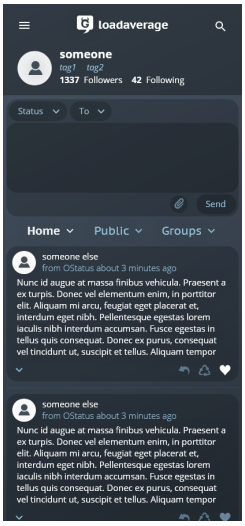

Figure 5: Mobile

5 Material and tools

5.1 Modular design

5.1.1 Event-driven on a Component-based architecture

The internal API to plug front-end components followed the approach adopted in the back-end regarding the event-driven programming, i.e., a declarative paradigm.

The GNU social Component-based architecture overlaps with this concept by providing a clear distinction between what can not be changed (core), what is replaceable but must always be present (component), and what can be removed or added (plugin).

Having that in mind, we wanted the core to make calls with context-sensible events to be handled by different components or plugins. Therefore, we have implemented a Twig function that would enable context-sensible events to be thrown directly by the templates. Hence, components and plugins can handle events and request handling of their own in their templates.

Code Snippet 1: Note template example

```
<div class="note">
  <div class="note-info">
    {% set nickname = note.getActorNickname() %}
    <a href="{{ path('settings_avatar') }}">
      <img src="{{ note.getAvatarUrl() }}"
      alt="{{ nickname }}'s avatar">
    </a>
    <b>{{ nickname }}</b>
    {% set reply_to = note.getReplyToNickname() %}
    {% if reply_to is not null and
        not skip_reply_to is defined %}
      {% trans with {'%name%': reply_to}%}
        in reply to %name% {% endtrans %}
    {% endif %}
  </div>
  <div class="note-content">
    {{ note.getContent() }}
    <div class="note-attachments">
      {% for attachment in note.getAttachments() %}
        {% if attachment.mimetype starts with 'image/' %}
          <div>
            <img src="{{ path('attachment_inline',
            {'id': attachment.getId()}) }}"
            alt="{{ attachment.getTitle() }}">
            <i> {{ attachment.getTitle() }} </i>
            </img>
          </div>
        {% elseif attachment.mimetype starts with 'video/' %}
          <div>
            <video src="{{ path('attachment_inline',
            {'id': attachment.getId()}) }}">
              <i> {{ attachment.getTitle() }} </i>
            </video>
```

```
34                </div>
35              {% else %}
36                <div>
37                  <i> {{ attachment.getTitle() }} </i>
38                </div>
39              {% endif %}
40            {% endfor %}
41          </div>
42        </div>
43        <div class="note-actions">
44          {% if have_user %}
45            {% for act in get_note_actions(note) %}
46              {{ form(act) }}
47            {% endfor %}
48          {% endif %}
49        </div>
50        <div class="replies">
51          {% for reply in note.getReplies() %}
52            {% include '/note/view.html.twig'
53               with {'note': reply, 'skip_reply_to': true,
54               'have_user': have_user} only %}
55          {% endfor %}
56        </div>
57      </div>
```

Using Twig templates This example demonstrates the power of it's template. Note replies use the same template as the parent, only differing in their parent DOM element which provides context. This results in a consistent and replicable way of rendering user posts.

By using templates with their context given, interface blocks can be displayed according to their size and respond accordingly, allowing for them to be written once and be used anywhere. This is what achieves a truly modular interface design that complements it's own platform infrastructure.

5.1.2 Meaningful HTML

Federated networks open themselves to other instances within the Fediverse, allowing for communication and sharing of content throughout the various nodes. As such, there is a need to implement concise and deterministic ways to present this content in a polished and cohesive manner.

Using Microformats to our advantage Since Microformats is one of the the agreed upon standards of operation within the Fediverse it presents itself as a great solution to this end. Microformats is primarily designed for search engines and aggregators, but it also provides a objective and consistent way to determine what content is provided within different instances, allowing it to be displayed correctly to the end user.

6 Roadblocks

6.1 Firefox's inability to render gradients properly

The final design iteration presented a smooth background gradient to achieve more depth and a bit more liveliness to the whole design. Unfortunately, Firefox presents significant banding in gradients, more so in fact when dealing with darker colours (as is the case here). Chromium based browsers on the other hand achieve better results by using some form of dithering.

Currently there is a CSSWG proposal that would mitigate this effect by using easing functions to better control colours mixing, in the future this problem would be easily solvable by just doing something like this:

Code Snippet 2: CSSWG proposal example

```
#future {
  linear-gradient(
    to bottom,
    hsla(330, 100%, 45%, 1),
    ease-in-out,
    hsla(210, 100%, 45%, 1)
  );
};
```

As for now, Andreas Larsen created a very handy script that emulates this feature in the present day Larsen (2017), resulting in this:

Code Snippet 3: Using Andreas Larsen script

```
.forNow {
  linear-gradient(
    to bottom,
    hsl(330, 100%, 45.1%) 0%,
    hsl(331, 89.25%, 47.36%) 8.1%,
    hsl(330.53, 79.69%, 48.96%) 15.5%,
    hsl(328.56, 70.89%, 49.96%) 22.5%,
    hsl(324.94, 63.52%, 50.4%) 29%,
    hsl(319.21, 54.99%, 50.3%) 35.3%,
    hsl(310.39, 46.14%, 49.68%) 41.2%,
    hsl(296.53, 39.12%, 49.7%) 47.1%,
    hsl(280.63, 42.91%, 53.43%) 52.9%,
    hsl(265.14, 47.59%, 56.84%) 58.8%,
    hsl(250.13, 52.52%, 59.88%) 64.7%,
    hsl(235.88, 59.2%, 60.91%) 71%,
    hsl(225.81, 68.23%, 57.85%) 77.5%,
    hsl(218.93, 74.97%, 54.21%) 84.5%,
    hsl(213.89, 79.63%, 49.97%) 91.9%,
    hsl(210, 100%, 45.1%) 100%
  );
};
```

6.2 Checkbox hack and accessibility

Hiding panels without JS GNU social has always had a JavaScript-optional interface and this implies that in some way every core feature, such as hiding panels within view, needs to be implement without it's use. To be able to control panels behavior the so called "checkbox hack" was used, with it an element can be hidden depending on the state of the checkbox in question. However, this trick is quite limited, since the element one tries to control needs to follow the checkbox in the HTML's structure.

Accessibility issues The checkbox hack has a key flaw besides it's limitations, if implemented naively, keyboard navigation of controlled elements becomes impossible. If "display: none;" or "visibility: hidden;" are used screen readers won't be able to read the default checkbox element. This also prevents us from using the :focus pseudo-element on the hidden input Lindsey (2018). Thus, after all considerations the following should be used:

Code Snippet 4: HTML checkbox hack structure

```
<label for="toggle">Do Something</label>
<input type="checkbox" id="toggle">
<div class="control-me">Control me</div>
```

Code Snippet 5: Accessible checkbox hack CSS

```
.control-me {
    /* Default state */
}
#toggle:checked ~ .control-me {
    clip-path: polygon(0, 0);
    /* OR */
    opacity: 0;
}
```

7 Results and conclusion

Solid pillars A lot of effort and research were put in place to create a great base structure to build upon. With these solutions I hope to have offered this community a good stepping stone for the project to reach new heights. Hopefully, through more work and time I hope, to continue developing the idea of reusable assets and templates to it's full potential.

7.1 The timeline view

Responsive design The mobile user experience has been thought after, by separating active and non-active views it's possible to use smaller screens to their full potential. When the viewport is large enough, elements that are now useless such as the hamburger menu on the top left are hidden away and their respective panels shown to make up for a more clean and minimal look.

Navigation and plugins The various timelines can be accessed through "tabs", since they're commonplace in the Web they're are easily recognizable frames. Both "tabs" and left/right panels may allow for plugins, the former in the form of simple links and the latter through links or a new "block" entirely.

Accessibility Navigating is entirely possible with just the keyboard, focused elements are highlighted, even those that are in fact checkbox hacks. The colour palette used aligned with the WCAG contrast requirements, even exceeding them. The chosen typeface offered high legibility and a large x-height, which coupled with the carefully chosen colours allow great readability for the end user.

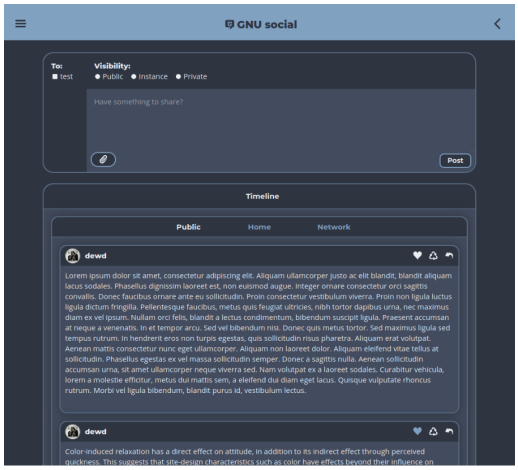

Figure 6: The home timeline

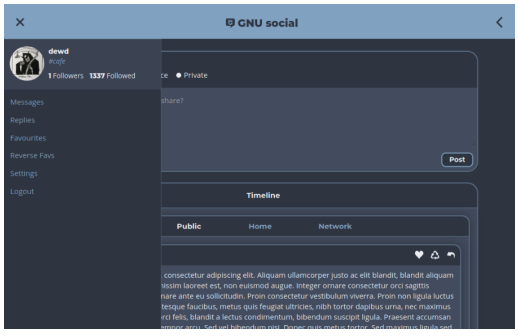

Figure 7: Left panel in view

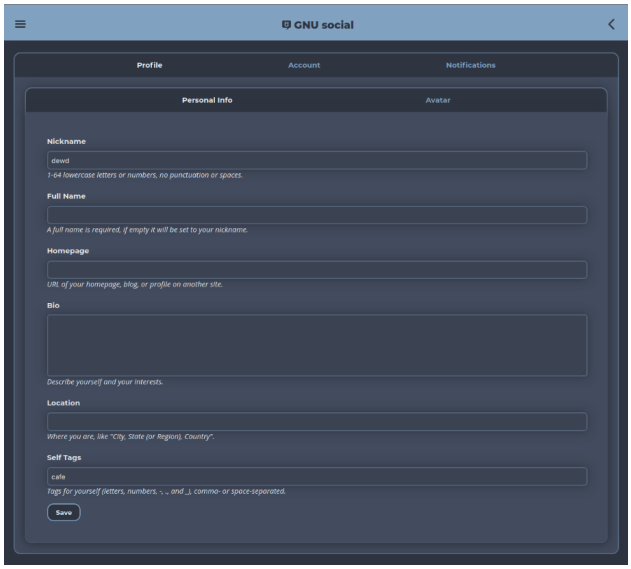

Figure 8: The new user panel

7.2 User panel overhaul

Future proofing One area focused upon was the user panel, the original interface became difficult to organize and cluttered when coupled with various plugins. Moreover, important features provided by plugins such as account backup and deletion were placed, for a lack of better options, in areas that did not align with the page movement flow, breaking the interface experience and polish. The same principle of tabs was applied here.

Better organization The various options were grouped together carefully to accommodate these tabs, they were organized in a way to make it easier to navigate and find whichever options the user wishes to.

References

Curtis, N. (2010). *Modular web design: creating reusable components for user experience design*. New Riders/Pechpit Press.

Cyr, D., & Trevor-Smith, H. (2004, 11). Localization of web design: An empirical comparison of german, japanese, and united states web site characteristics. *JASIST*, *55*, 1199–1208. DOI: 10.1002/asi.20075

Gorn, G., Chattopadhyay, A., Sengupta, J., & Tripathi, S. (2004, 04). Waiting for the web: How screen color affects time perception. *Journal of Marketing Research*, *XLI*, 215–225. DOI: 10.1509/jmkr.41.2.215.28668

Johnson, J. (2014). Chapter 3 - we seek and use visual structure. In J. Johnson (Ed.), *Designing with the mind in mind (second edition)* (Second Edition ed., pp. 29–36). Boston: Morgan Kaufmann. Retrieved from https://www.sciencedirect.com/science/article/pii/B9780124079144000038 DOI: https://doi.org/10.1016/B978-0-12-407914-4.00003-8

Jones, B. (2011, September). *Understanding visual hierarchy in web design.* Envato Tuts. Retrieved from https://webdesign.tutsplus.com/articles/understanding-visual-hierarchy-in-web-design--webdesign-84

Keyes, E. (1993). Typography, color, and information structure. *Technical Communication*, *40*(4), 638–654. Retrieved from http://www.jstor.org/stable/43090213

Larsen, A. (2017, May). *Easing linear gradients: Css-tricks.* Retrieved from https://css-tricks.com/easing-linear-gradients/

Lindsey. (2018, November). *Create custom keyboard accessible checkboxes.* Retrieved from https://www.a11ywithlindsey.com/blog/create-custom-keyboard-accesible-checkboxes

Marcus, A. (1995). Principles of effective visual communication for graphical user interface design. In R. M. BAECKER, J. GRUDIN, W. A. BUXTON, & S. GREENBERG (Eds.), *Readings in humanâcomputer interaction* (pp. 425–441). Morgan Kaufmann. Retrieved from https://www.sciencedirect.com/science/article/pii/B9780080515748500443 DOI: https://doi.org/10.1016/B978-0-08-051574-8.50044-3

WICG. (2019). *2019 proposal/solution for container queries.* Retrieved from https://github.com/WICG/container-queries/issues/12

Wiener, L., Ekholm, T., & Haller, P. (2017). Modular responsive web design: An experience report. In *Companion to the first international conference on the art, science and engineering of programming*. New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3079368.3079404 DOI: 10.1145/3079368.3079404