Pleroma API based GNU social frontend and Pleroma API

Name:	Susanna Di Vita
E-mail:	susanna.divita.2@gmail.com
Location:	Genova, Italy (GMT+2 Rome Timezone)
Website:	github.com/SusannaDiV
IRC & XMPP:	susannadiv

Project Name:	New Frontend Modern + API
Designs:	notabug.org/susdiv/designs-gnusocialv3
Proof of Competence:	notabug.org/susdiv/gnusocial.network

Summary

This project aims at developing full-stack Pleroma API plugins that will replace Qvitter in GNU social v3.

GNU social is a communication software used in federated social networks. GNU social's current UI has been implemented on top of the Qvitter API and is currently undergoing a revamp from v2 to v3. Due to Qvitter API's lack of documentation and hard-coded nature, support will be discontinued in v3. This project aims at completely replacing Qvitter API code with the highly customizable and well supported Pleroma API by taking advantage of Pleroma's compatibility with both GNU social's formerly employed open protocols - OAuth2 and ActivityPub.

This will be achieved by implementing 3 plugins:

- a Pleroma API based GNU social frontend
- the Pleroma API
- a Chat system for direct messaging

The first part of the project aims at developing a GNU social UI with an improved UX by reducing user friction. Clean lines, lots of space, and well-defined elements will visually indicate to users how to move through your UI, eliminating the need for cluttering annotations. The resulting UI will be more user friendly – both aesthetically and functionally – and will fix some of the minor and major bugs present in the v2 frontend, such as markup issues and non-strictly necessary UI features that might mislead newcomers and interrupt user flow.

The second part of the project aims both at implementing admin-specific actions through the development of relevant existing Pleroma API endpoints and at developing user-side authorization request handling. This will be achieved in accordance with both industries' best practices for REST API development and Pleroma's specifications, meticulously studied though endpoint analysis. The resulting Pleroma plugin, completed with the user-facing frontend, will be fully compatible with the pre-existent software.

As for the last part, this project aims at developing a Chat system plugin. This will allow users to communicate through direct messages within GNU social, offering an alternative to IM protocols.

Creation of all UI's components will be completed following Object Oriented Analysis and Design principles (Analysis and Design, Implementation, Testing and Deployment). Testing stage can be carried out using Exploration testing as it leads to a frontend-optimized final unit. To promote accessibility, each design change will encompass thoughtful CUD accessibility testing through the Coblis filter.

Completion of every further improvement will be followed by its documentation. The latter contains explanations of the implementation approach and testing process in addition to the responsive design mockups and wireframes – for the frontend – or the automated unit test sections – for the backend. The goal is to keep an optimized balance between documentation/reports – carried out with regards to the GNU Contribution Guidelines - and actual feature implementation and testing.

Benefits

Such improvements will be greatly beneficial to GNU Social, the fediverse and, obviously, the whole community involved. Indeed:

- Improved user experience will gratify former fediverse users
- A catchier design will make a great first impression and attract newcomers
- Increased portability thanks to Pleroma API's low system requirements will permit the installation of new instances on dated hardware
- Improved accessibility will help both spreading the word of Free Software and making it more attractive to those familiar to traditional mainstream social network platforms
- The addition of a chat plugin will allow users to communicate through direct messages – thus enabling stronger interactions among users
- Mobile-friendly frontend will allow users to access their accounts' direct messages on the go, thus increasing contact

Implementation plan and Relevant Research

Frontend UI Design and Implementation for improved UX and User Interaction

The original Qvitter FE has an indistinctive, non-content-centered design that focuses more on customizable tweaks than on post readability and user interaction.

The offered UI results in a bubble-tiled, old-timey two-column structure; the left side is cluttered with less than relevant, non-user specific sections that could be easily omitted in most of the timelines (namely the Home, Public, Popular and Network), while the timeline's main content is relegated to the admittedly less attention grabbing column. This layout goes against all UI visual hierarchy best practices, and therefore will be completely replaced by the content-centric three-column design.

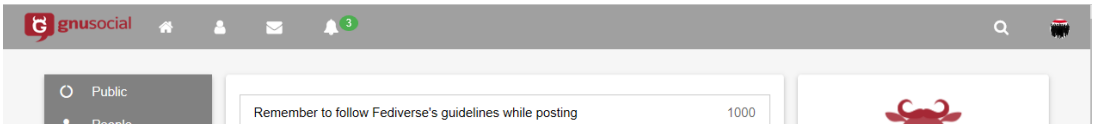
Moreover, in order to comply with the F-pattern reading techniques unconscionably followed by most left-to-right user readers and therefore ameliorating the overall UX, all call-to-action elements like the Profile and Timeline Navigation tiles will be placed at the top left and right margins of the screen, instead than being regrouped in a single clump as the previous implementation did.

Additionally, in order to bring a sense of prioritized order within the elements themselves, the background will be neutrally-colored and separated from the user's sight within the framework. This will provide a blank canvas on which to arrange the sections, now perceived as easily-recognizable and eye-catching instead of too-heavily loaded.

Finally, Qvitter UI's suboptimal placement of call-to-action elements like buttons and drop-down menus will be corrected as follows in accordance with the Z-pattern design standard - in order to redirect the user's focus on the timeline's main content after the first-contact horizontal-search scanning phase.

- Adding a completely new Navigation Bar – containing the Home, Profile, Messages fields and the unified Favorites/Replies Notification drop-down menu. The presence of a navbar will help streamline the interface by creating a starter point for user's browsing, thanks to newly introduced universally recognizable icons.

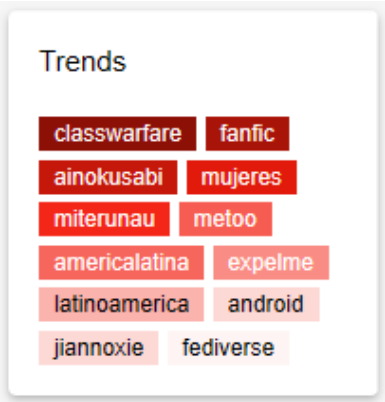
  Moreover, space management will be improved as such addition would eliminate the need for the user-specific section in the left column, making the following Popular Tags and Feed sectors more visible.



- Tag cloud redesign – obtained through CUD-friendly redundant color/darkness/saturation coding. Such distinct and colorful gradient separation of Popular Tags will result in a more space efficient relevance-ranking system.

  The currently employed dimension-importance ratio technique clogs a good portion of the right column making the Popular Notices section less visible in the Home, Public and Network timelines.

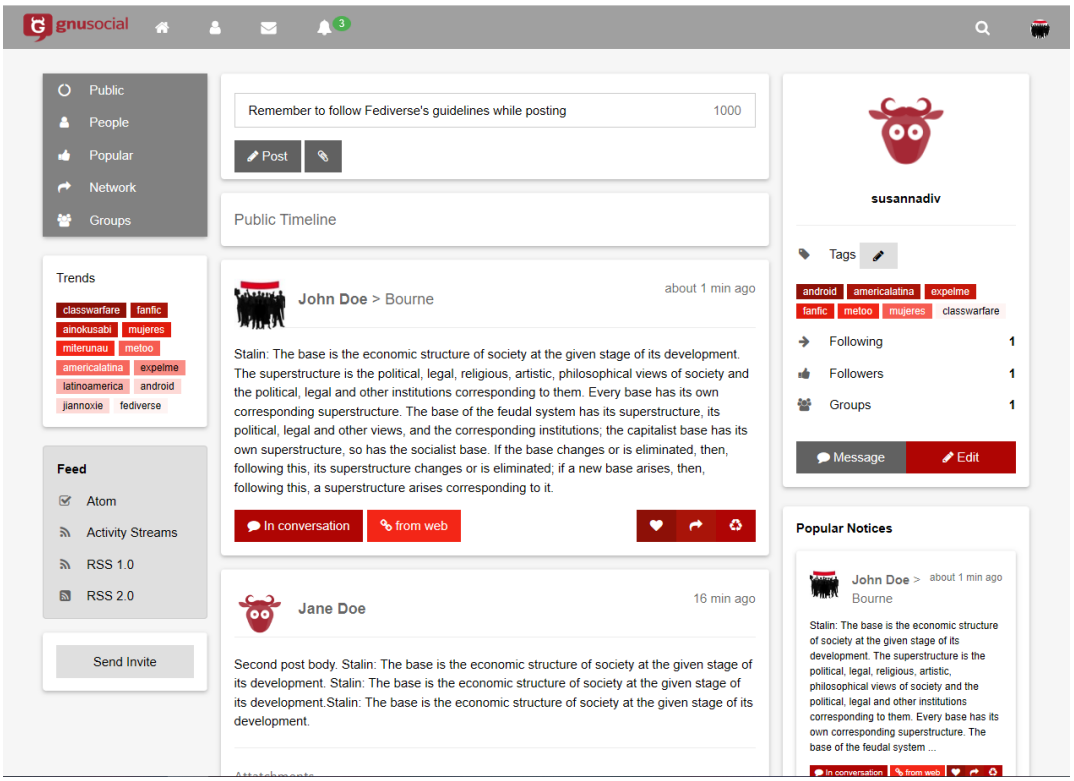
Not to mention, in the current v2 implementation the purpose of a Recent/Popular Tags cloud gets defied by making users scroll to see the ending tags in the list.



- UI Timeline redesign

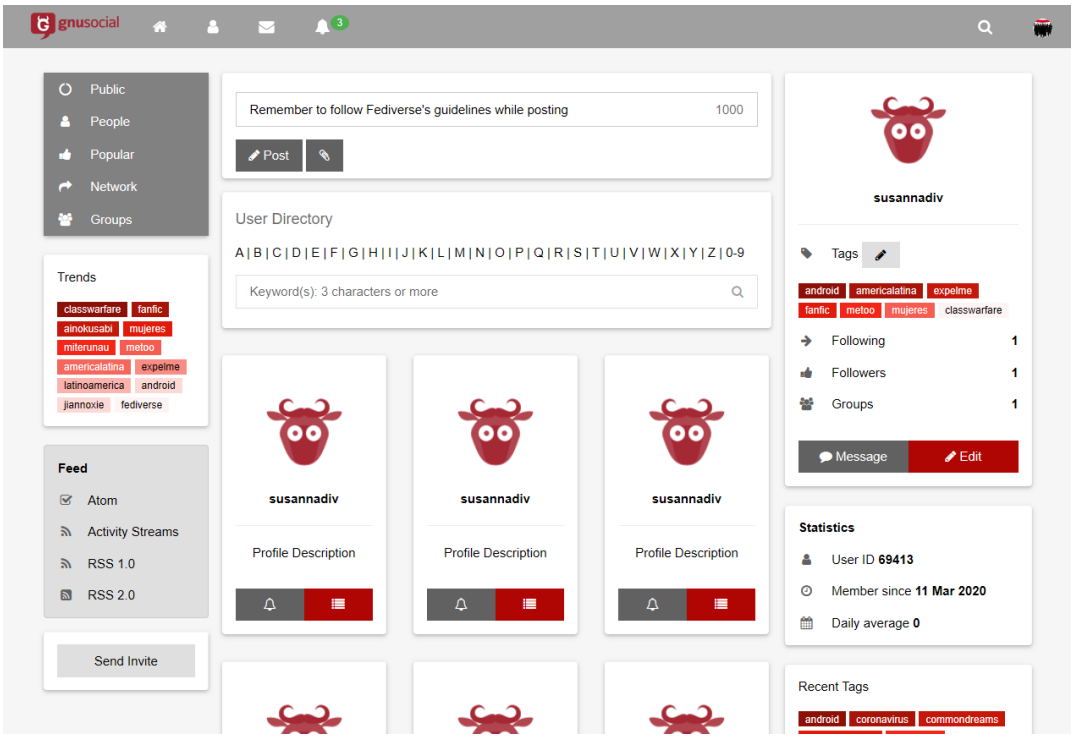
  In order to lead users down the content page organically, this project will implement an efficient strategy for elemental hierarchy; essential navigation page elements will be stored in the left column, with different grayscale shades to highlight their level of importance; the former Recent Tags section, now renamed Trends for improved readability, will follow and so will the Feed timeline menu.

  The right column will store the Personal Profile section instead, followed by different sectors., according to the Timeline's content (Active/Popular Groups list, User Statistics or Popular Notices). This new separation technique will also bring out the heavily text-based content of the Popular Notices section.



- User Directory and Group Directory Redesign – in order both to maintain consistency throughout the design and to reduce first-approach fatigue, the standard unordered list common to both the Users and Group Directory will be replaced with an easily readable and catchier tile design, which users/groups information will be listed in right after their avatar instead of being half-hidden under the User's Profile section (as frequently occurs due to lack of proper CSS nesting).

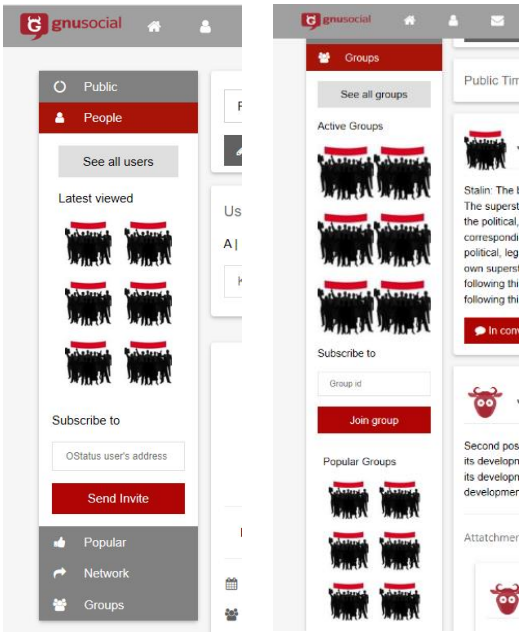
  Additionally, the new User Directory alphabetical index & search bar will be compressed together as to avoid wasting user-focusing space (currently they occupy more than half of the page), that shall be assigned instead to the main content.



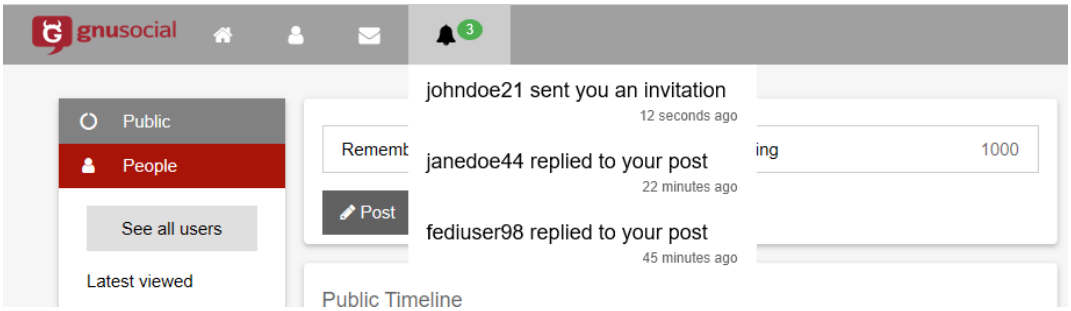
- Button Redesign with more recognizable icons and text descriptions to raise the UI's recognition potential and to minimize recall fatigue on the user's end. The added color-coded contrast helps action buttons to stand out and have their function immediately recognized by users – while breaking the content's grey scale monotones and allowing a catchier design implementation.

- Added hover background focus and improved general responsiveness of the UI - for a modernized, accessible look. This will provide users with immediate feedback on their actions, while also allowing for device-optimized layout changes thanks to the extensively documented CSS Grid usage. Mobile-friendly features - such as horizontal-scrolling avoidance, consistent zoom-free text readability and adequate spacing for tap targets - will be implicitly implemented.

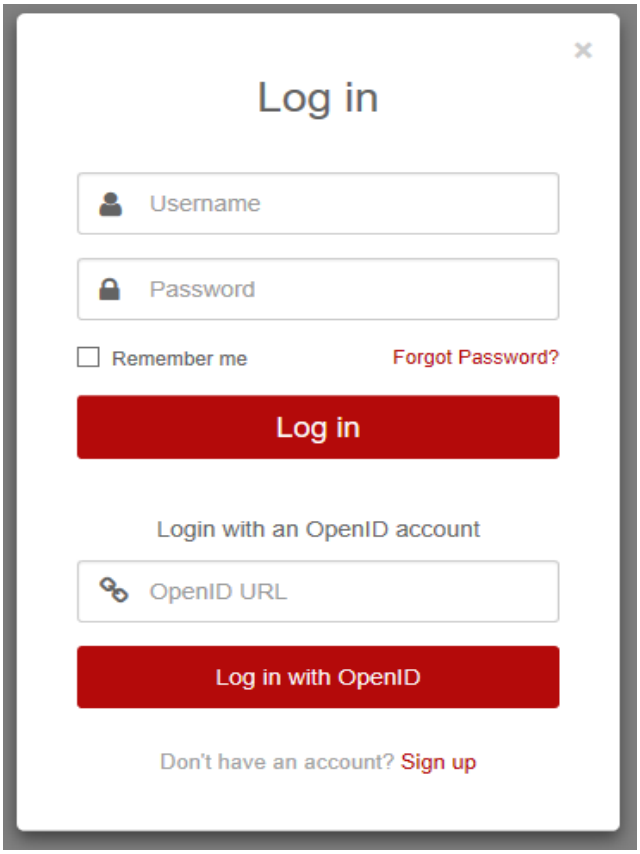
- Added profile and group shortcut in the Timeline navigation section – to optimize interaction flow and to endorse user-side exploration. The aim is to improve design immediacy and to make both content and features available for discovery; (in turn) this will help avoiding disengagement once a user has begun working with maximum efficiency. Integrating the Subscribe To form within the People and Groups fields helps achieving both a UI free from friction-inducing clutter and a better free space management in the right column.



- Drop-down Notices Menu implementation – it will contain notifications coming both from the Favorites and from the Replies Timelines. Having all notifications stored in one place will improve time responses to other user's action and therefore increase user interaction. users will be directed to the corresponding timeline by clicking on the above notifications. A second drop down menu will be present in the ride side of the navbar for Account-related options – namely, the Settings and Logout buttons.



- Login page implementation – the current login and registration can only be accessed through a barely visible, non-intuitively placed all-in-one button. This not only makes it difficult for new users to locate the starting point of their registration process, but it also takes them to an unmodified, old-fashioned timeline layout. I intend to develop a separate Login and Registration page, complete with a more directly approachable form containing all login options.



- Introduce a simplified settings page by regrouping related sectors. The Avatar and URL fields will be integrated as subsections of the Profile main settings page, while access-related Email, OpenID and Password fields will be stored under the Security and Login section. Finally, both the Themes and Connections will be clustered in a Personal section.

Pleroma REST API Plugin backend Design and Implementation

This part of the project aims at implementing the Pleroma API to be offered bundled with its newly created frontend as a full-stack plugin in GNU social's v3. In order to completely eradicate the need of employing Qvitter API.

Request's Authorization Handling

The authorization framework currently employed in both GNU social and in Pleroma's server-side backend is OAuth2. Therefore, all authentication requests will be handled with OAuth2 tokens.

Requests that require authentication are:

- Account-related - like the ones regarding the deletion/disabling of a user's account or changing their email
- Notification-related - like marking of notifications as read, subscribing and unsubscribing to all statutes coming from a given user or updating of notification settings
- Conversation-related - like obtaining the timeline for a given conversation, updating its recipients, marking it as read or acquiring a conversation by the given status.

In addition to the above, all Admin-privileged actions regarding GNU social's users, groups and singular accounts will require authentication tokens to prove their Admin status. I will implement an easily maintainable authentication token structure by adding an OAuth2 admin scope requirement toggle. If set to true, admin actions will explicitly demand admin OAuth2 scope presence within the authentication token. If set to false, access to admin-specific actions will be granted only in presence of the is_admin flag.

Following Pleroma's specifications, request parameters will be passed via query strings and files will instead be uploaded as multipart/form-data. All attached files will have an additional string field mime_type under the pleroma object, defining the file's MIME type.

Pleroma's endpoints are divided in two categories, Pleroma API and Admin API. The prior contains the endpoints related to user-permitted requests, while the latter collects all admin-specific actions.

Pleroma API

Relevant available Pleroma API user-side endpoints - to be implemented inside the GNU social Plugin - are:

Timeline endpoints

Adding the parameter **with_muted=true** to all timeline queries will also return activities by muted (not by blocked) users. This will be employed in the Public, Popular and Network Timelines, as they contain conversations from a wide range of instances – both local and remote - not directly connected to the user that might still be of interest.

Adding the parameter **exclude_visibilities** to the timeline queries will instead exclude the statuses with the given visibility types specified in the input array. This will be employed in all but the Home and Profile Timelines, as to exclude private and direct conversations the user's not part of (e.g. **exclude_visibilities[]=direct&exclude_visibilities[]=private).**

- **/api/v1/pleroma/accounts/:id/favorites** – returns the content of the Favorites timeline of any user. The optional parameter "limit" will not be specified as a loading animation will be encompassed at the end of the first results page.

Conversations

- **GET /api/v1/pleroma/conversations/:id** – returns the conversation with the given ID. As Pleroma Conversations statuses can be requested by their Conversation id, this endpoint will be employed for the "See conversation" button functionality.
- **GET /api/v1/pleroma/conversations/:id/statuses** – returns the retrieved timeline for a given conversation. This request accepts the additional "in_reply_to_conversation_id" parameter which, when set, will change the visibility to direct and address only the users who are the recipients of that Conversation.
- **PATCH /api/v1/pleroma/conversations/:id** – updates a conversation by changing the list of user ids able to receive messages in the conversation, specified in the parameter "recipients" under the "pleroma" key. Such list will be updated by replacing the current parameter's content the full list in the newly updated set of recipients. As the owner of the owner of the conversation will always be part of the resulting set of recipients, it will not be necessary to implement safeguards in the development process to avoid accidentally exclude the original poster from any possible replies. Moreover, adding or removing elements will be allowed only if explicitly changed by the user though this request.
- **GET /api/v1/pleroma/conversations/read** – marks all user's conversations as read. The returning JSON file contains the list of conversations entities that were marked as read (200 healthily, 503 unhealthily).

User Actions

Notifications

In addition to the user's id parameter, I am also going to specify two additional parameters: **exclude_visibilities** and **include_types.** The prior excludes the notifications for activities with the visibilities present in the input array (which may contain one or more of the following options: public, private and direct). This parameter will be set accordingly, whether it will be used in the chat's direct message implementation or in the Profile Timeline (which contains both public and private mentions).

Usage example: **GET /api/v1/notifications?exclude_visibilities[]=direct&exclude_visibilities[]=private**.

The latter, instead, includes the notifications for activities with the types reckoned by the input array (which may contain one or more from the following: mention, follow, reply and favorite). I will set this parameter accordingly to the Timeline I will be implementing (Favorite or Replies) and to the type of notices that will be displayed in the navbar drop-down.

Usage example: **GET /api/v1/notifications?include_types[]=mention&include_types[]=reblog**.

- **GET /api/v1/notifications –** used to retrieve a user's notifications which will be displayed both in the navbar Notices drop-down and in the respective Favorite and Replies Timelines.
- **/api/v1/pleroma/accounts/:id/subscribe** – employed to subscribe to receive notification for all statues posted by a user. It requires the id of the account to subscribe to as the only parameter; the JSON response contains a relationship object on success, **{"error": "error_msg"}** otherwise.
- **/api/v1/pleroma/accounts/:id/unsubscribe** – employed to unsubscribe to stop receiving notifications from user statuses. Parameters and response types are in common with the above.
- **/api/v1/pleroma/notifications/read** – marks notification as read. The response will contain the notification entity – or the array of notification entities - that were read.
- **/api/pleroma/notification_settings** - used to update user notification settings. This endpoint will be employed in the "Settings" page, with the following boolean fields as parameters:
    - **followers** - user receives notifications from followers
    - **follows** - user receives notifications from people the user follows

  This request returns JSON **{"status": "success"}** if the update was successful, otherwise returns **{"error": "error_msg"}**.

Profile/Account

- **PATCH /api/v1/pleroma/accounts/update_avatar -** used to Set/clear user avatar image
- **POST /api/pleroma/delete_account** & **POST /api/pleroma/disable_account** – used for deleting and disabling an account, respectively. They require the user's password as the only parameter. They both return JSON responses; **{"status": "success"}** if the account was successfully deleted/disabled, **{"error": "[error message]"}** otherwise.
- **/api/pleroma/captcha** – returns a new captcha. It requires no parameters and the response is a provider specific JSON, in which the only guaranteed parameter is type.

Admin API

User

- **GET /api/pleroma/admin/users –** returns the list of all users in a JSON file. The optional parameters will be added in this project are:
    - Query – string search term containing the nickname, domain, or nickname@domain of the users we want to list.
    - Filters – comma-separated strings of filters to list only local, external, active, deactivated users. Moreover, the addition of the is_admin and is_moderator filter makes lists users with and admin or moderator roles.
    - Tags – string containing the tags list

- **POST, DELETE /api/pleroma/admin/users – used to** create and delete a specific user, respectively. User creation requires as **users**: [{ **nickname**, **email**, **password** }] only parameter and returns the user's nickname. User deletion only requires the target's nickname and returns an array containing the user nicknames left.
- **PATCH /api/pleroma/admin/users/activate** & **PATCH /api/pleroma/admin/users/deactivate** – utilized to activate and deactivate given users passed though a nicknames array, respectively. The JSON response contains a list of the newly added/deleted users as objects.
- **POST /api/pleroma/admin/users/follow** & **POST /api/pleroma/admin/users/unfollow –** employed to make a user follow/unfollow a specific user, respectively. Requested parameters are the nicknames of both the follower and the followed, contained in the homonymous parameters.
- **PUT /api/pleroma/admin/users/tag** & **DELETE /api/pleroma/admin/users/tag** – used to respectively tag and un-tag a list of users from a "nicknames" array.

Profile

- **GET /api/pleroma/admin/users/:nickname_or_id** – retrieves the details of a user. Such user can be identified by either the id or nickname parameter. The response is the JSON file of the user if the request was successful, "Not found" otherwise.
- **GET /api/pleroma/admin/users/:nickname_or_id/statuses** – returns the user's last statuses. Other than the id/nickname fields for unequivocal user identification, the additional boolean "godmode" parameter will be set as true in order to allow private-message timeline showing after login. This endpoint is necessary for the user's Home and Profile Timeline implementation, where the latest statutes (max 100, following Pleroma's specifications) contained in the returned JSON array will be displayed.
- **GET /api/v1/statuses** – similarly to the above, this endpoint will be implemented to return multiple statutes in a status array; this however will be achieved by passing an array of activity ids instead of a user's id. This endpoint is necessary for retrieving statuses from permalinks. Usage example: **GET /api/v1/statuses/?ids[]=1&ids[]=2** – retrieves as array with the activities 1 and 2
- **GET /api/pleroma/admin/statuses** – retrieves the instance's latest statuses. This endpoint will be employed for the Network Timeline implementation (with the godmode value set to false). On success, this request returns the JSON array of instance's latest statuses; on failure, "Not found".

Group

- **GET /api/pleroma/admin/users/:nickname/permission_group** – retrieves user permission group membership, does not require parameters. This endpoint will be employed to check if a given user has moderator or admin permission before accessing privileged operations.
- **POST /api/pleroma/admin/users/permission_group/:permission_group –** adds users to the permission group; requires an array of nicknames as sole parameter. Returns the JSON of the user if the request was successful, **{"error": "..."}** otherwise.

- **DELETE /api/pleroma/admin/users/:nickname/permission_group/:permission_group** – removes an user from the permission group. Both the parameter and the response follow the same format as above; however, admins cannot revoke their own admin status.
- **GET /api/pleroma/admin/moderation_log –** employed to get the moderation log in a JSON response. This endpoint's optional parameters I will implement hold date, user and search filtering properties regarding log display. In particular:
    - **start_date, end_date** – filter logs by creation date, starting from **start_date** and ending by **end_date**. All datetime values will be in ISO 8601 format (YYYY-MM-DDThh:mm:ss) in order to follow Pleroma's specifications.
    - User_id - filter logs by user's id (integer format).
    - Search – post-filtered search logs by their log message (string format).

Registration

- **POST /api/pleroma/admin/users/invite_token** & **POST /api/pleroma/admin/users/revoke_invite** – used to create and revoke an account registration invite token, respectively. The former requires the date string "expires_at" parameter, the latter just the name of token. Both requests return a JSON response.
- **POST /api/pleroma/admin/users/email_invite** – employed to send a registration invite via e-mail. This plugin's implementation requires just the email parameter.
- **PATCH /api/pleroma/admin/users/confirm_email** – utilized to send the account confirmation email to the users specified in the nickname's parameter; return an array containing the nicknames of said users.
- **PATCH /api/pleroma/admin/users/resend_confirmation_email** - employed to resend the confirmation email to the users specified in the nicknames parameter. The response will contain the array of users nicknames whom the confirmation email has been re-sent.
- **GET /api/pleroma/admin/users/:nickname/password_reset** – utilized to get a base-64 password reset token for a given name.

Settings

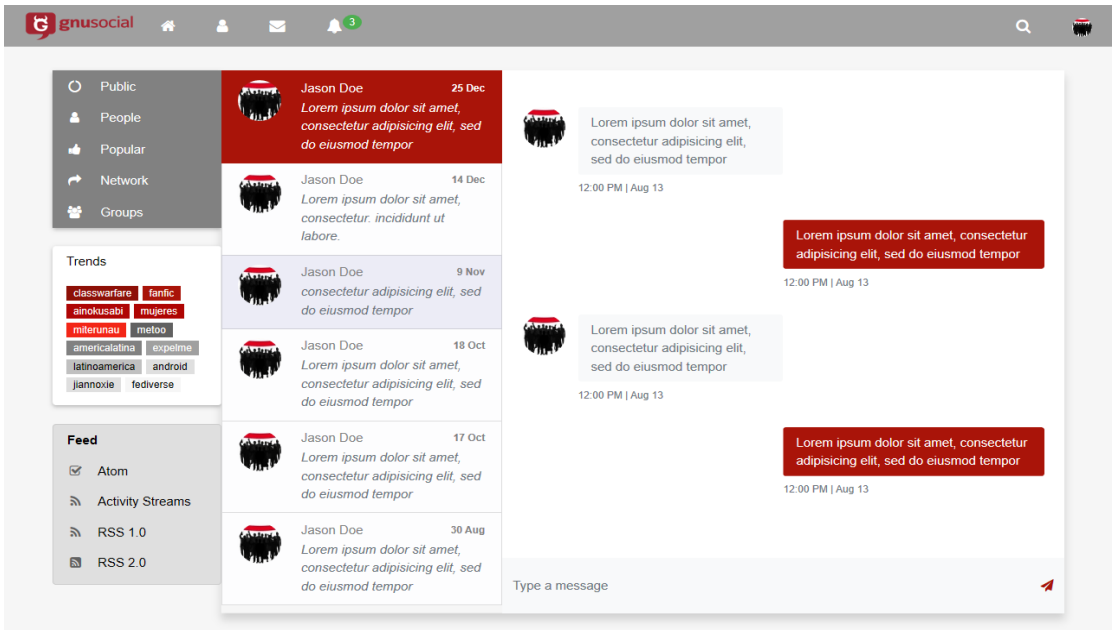
- **GET /api/pleroma/admin/users/:nickname/credentials** - Gets the user's email, password, display and settings-related fields as a JSON response; requires only the nickname as parameter.
- **PATCH /api/pleroma/admin/users/:nickname/credentials** – required for changing the user's email, password, display and setting-related fields. The number of parameters to be added to the JSON body depends on the amount of changes requested by the admin; this plugin's available parameters will be email, password, bio and avatar.

Implementation of Chat System Plugin

I propose the introduction of a new plugin – the Chat plugin – to allow direct messages between users. The current Message layout employed in GNU social has an inbox/outbox non-user-regrouped structure, which makes it difficult for users to communicate as messages are displayed in chronological order while neither keeping track of responses nor categorizing user-specificities. Both Qvitter and Pleroma implement private messages by integrating them in the timelines and setting them to be recipient-exclusive. However, this implementation is wildly considered as over-complicated and riddled with compatibility issues.

In order to provide users with a seamless IM-like direct chatting experience, while simultaneously keeping things simple on the implementation end to ease compatibility, this project will employ ActivityPub's plugin in order to implement a chat plugin compatible with the Pleroma API. Implementation of the private message notification system straightforwardly invokes WebSockets.

- Creation of a minimizable Chat widget – The aim is to improve user contact via private conversations and to highlight notifications which decrease user response time. Furthermore, the UX browsing won't be affected: users will not have to quit their content viewing/writing activities as the chat functionality would be accessible from any page.
- Message page design – with users' conversation indexed on the left and chat bubbles appearing on the right. This chat will be completely responsive and will feature a site-wide accessible emoji pack.



- Backend chat implementation – Pleroma API's Emoji endpoints and Pleroma Conversation's key parameter will allow me to develop the chat's backend:
    1. Conversation IDs can be found (and therefore implemented) in direct messages thanks to the *pleroma.direct_conversation_id* key.
    2. *Mix.Tasks.Pleroma.Emoji* will be used for importing pre-existent emoji packs; site-wide customized ones will be adapted thanks to

the **GET /api/pleroma/emoji, PUT-DELETE /api/pleroma/emoji/packs/:name** and **POST /api/pleroma/emoji/packs/:name/update_file** endpoints - which list, create, update and delete custom site-wide emoji packs, respectively.

3. Notifications will be displayed as read when clicked on thanks to the additional field is_seen under the “pleroma” object.

Tentative Timeline

May 4 – June 1

- Familiarize with the community
- Familiarize with the code, with further insights in:
    1. Pre-existent JavaScript for better migration technique planning
    2. Timeline Notices Handling
- Familiarize with both documentation, development and test system used
- Familiarize with the Pleroma API
    1. Pleroma FE options
    2. Chat employable backend API Endpoints
- Create additional flow state diagrams for UI path indexing
- Keep on conducting user tests for quality-of-life issues fixing in the original UI

June 1 – June 29

June 1-7

Frontend UI Design and Implementation for improved UX and User Interaction

- Work on 1.1 and 1.2
    1. Adding a completely new Navigation Bar
    2. Tag cloud redesign
- Work on 1.3 and 1.4
    1. UI Timeline redesign
    2. Button Redesign with more recognizable icons and text descriptions
- Work on 1.5 and 1.6
    1. User Directory and Group Directory Redesign
    2. Added hover background focus and improved general responsiveness of the UI

June 8-14

- Work on 1.7 and 1.8
    1. Drop-down Notices Menu implementation
    2. Added profile and group shortcut in the Timeline navigation section
- Work on 1.9 and 1.10
    1. Introduce a simplified settings page
    2. Footer redesign
- Work on 1.11 and 1.12
    1. Login page implementation
    2. Bug Fixing and JavaScript code factorization
- Exploration Testing + Interactive Documentation

June 15 - 21

Pleroma REST API Plugin backend Design and Implementation

- Implementation of user-side Pleroma API backend – Profile/Account endpoints related to the Settings page implementation
    1. Setting/clearing user avatar image
    2. Deleting/disabling an account
    3. Obtaining a new captcha
- Implementation of admin-side Pleroma API backend – admin-specific Settings endpoints employed in the Settings page implementation
    1. Acquiring user's email, password, display and privacy-related settings
    2. Changing user's email, password, display and privacy-related settings

June 22 – 28

- Implementation of user-side Pleroma API backend – Timeline (specifically Conversation-related) endpoints utilized for both the Favorite, Public, Home, Popular and Network timelines and the status' conversation structure development
    1. Favorite user-specific statuses retrieval
    2. Conversation recovery based on a status id
    3. Timeline recovery based on a given conversation
- Write automated unit tests – in accordance with industry's best practices - to verify functionality of the API implementation up to this point

- Review and test all API actions that require authentication to ensure their functionality
- Buffer time + documentation - in the newly implemented GNU social's landing page Doc section

July 3 - July 27

July 3 – 10

- Implementation of user-side Pleroma API backend – continuation of Timeline (specifically Conversation-related) endpoints development related to:
    1. Updating a conversation by changing its recipients
    2. Marking user-specific conversations as read
- Implementation of user-side Pleroma API backend – User Actions (specifically Notifications-related endpoints) utilized for both the drop-down list of recent Notices and the Favorite & Replies timeline development
    1. Acquire the user's notification list
    2. Subscribe & unsubscribe to receive notifications for all statutes posted by a given user
    3. Mark clicked-on notifications as read
    4. Update the user's notification setting (this endpoint will be employed in the Settings page implementation as well as in the Notices section)

July 11 – 18

- Implementation of admin-side Pleroma API backend – User Settings endpoints employed both in the pre-existent admin dashboard page and in the User Directory page implementations
    1. Acquire the list of all known users and instances in GNU social – with applicable query and tag filters
    2. Create/delete a specific user given their id/nickname
    3. Activate/deactivate a specific user given their id/nickname
    4. Make the requesting user follow/unfollow another specific user
    5. Add/delete user-chosen tags
- Unit testing + buffer time for documentation writing and eventual bug-fixing

July 19 – 27

- Implementation of admin-side Pleroma API backend – Profile tweaking endpoints employed in both the Home & Profile and in the Popular & Network timeline implementations
    1. Acquire all profile and setting related information from the user's id/nickname
    2. Return last 100 statuses from any given user on GNU social – (this endpoint will be heavily employed in this plugin's implementation; it will therefore require occupy much of the testing-allocated timeframe before the Second Evaluation)
    3. Retrieve multiple activity-identified statuses from a specified user
    4. Obtain the current instance's latest statuses (this endpoint will be also extensively tested as it is vital for displaying conversations with a fediverse-wide scope of visibility)
- Implementation of admin-side Pleroma API backend – Group handling endpoints employed in both the Groups Timeline and the Group Directory page development
    1. Acquire user permission status for group membership
    2. Add/Remove a given user to the permission group
    3. Obtain group-specific moderation log – with filtered results according to date, user id or search-related parameters
- Review and test thoroughly all timeline, user, profile and group related API actions
- Buffer time + documentation writing

July 31 - August 24

July 31 – August 7

- Implementation of admin-side Pleroma API backend – Registration enabling endpoints employed for internal configuration. In the current version, account registration can only be finalized through invitation tokens; I am going to develop the following while following code reusability best practices in order to ease any possible future readjustments.
    1. Create/revoke an account registration invite token
    2. Send a registration invite via e-mail
    3. Send the account confirmation e-mail to the newly created user
    4. Resend the account confirmation e-mail
    5. Acquire a base64 password reset token

August 8 – 15

Chat System Plugin Implementation

- Implementation of Direct Message system – independent from the timeline backend implementation - through WebSockets
- Message page UI design
    1. Main Message page front end implementation
    2. Minimizable Chat widget development
- Direct conversation-related endpoint backend implementation – with already-seen message marking
- Site-wide emoji packages
    1. Imported from Pleroma's server default list
    2. GNU social specific (facultative – only if the time permits it)

August 16 – 23

At this stage I should have completed all the proposed work. This part of the month will therefore be assigned as a buffer time in which I'll be completing additional testing and documentation writing - or previously assigned tasks in case I'll miss something from the previous months.

August 24 – 31

- General review, documentation and testing
- Tech-report write up
- Final merge and evaluation

Deliverables

- Fully functioning GNU social full-stack Pleroma plugin compatible with pre-existent software
    - GNU social UI for Pleroma API compatible with LibreJS
    - Pleroma API backend implementation with OAuth2 protocol
- Code refactoring and API endpoints addition
- Fixing of areas with unexpected behavior in GNU social's codebase
- Addition of a Chat plugin for direct messages (+ documentation)
- JavaScript codebase refactoring
- Mobile optimized UI with Chat widget
- Documentation and testing

Communication

The GNU social's IRC will be used to communicate with the community and the mentors (I can use whichever channel is more convenient). I will regularly submit my progress to both mentors and community members for UI-related feedback.

I'll be dynamically adapting my academic workload in order to maintain the momentum through the entire summer, while communicating any possible timeline changes or communication hours to Daniel and Diogo. Both my project's additions code and all relevant changes to the existing code will be hosted on a fork of Diogo's fork. This is to prevent accidental commits, as a significant part of the community has switched to Diogo's repository.

Qualification and Relevant experience

I'm currently a second-year BSc Computer Science 4.0 GPA student at the University of Genova (UNIGE), Italy. I am also a member of the IANUA-ISSUGE Excellence Academic Program (admission requires a perfect GPA score). In addition to the above, I also have 4 autonomous JavaScript, Bootstrap, HTML5, CSS3, Node.js, Vue.Js, React, GraphQL, Express.Js and Angular frontend projects on my GitHub account – one of which secured my team first place in the Innovation 4.0 SMAU Award for a responsive React Native & JS mobile app for flood alert display.

The inspiration to code for the Federated Social Network came from the words of Richard Stallman himself, whom I had the honor of meeting during a conference on the importance of Free Software at my University – the idea of offering a privacy-focused alternative to mainstream social media platforms immediately resonated with me. I started to get involved in the movement first through contact with the community and then through taking interest in the software that powered such federation.

For working in GNU social, I will have to keep learning more about the project itself in order to get comfortable with the Pleroma API and freshen up my JavaScript skills.