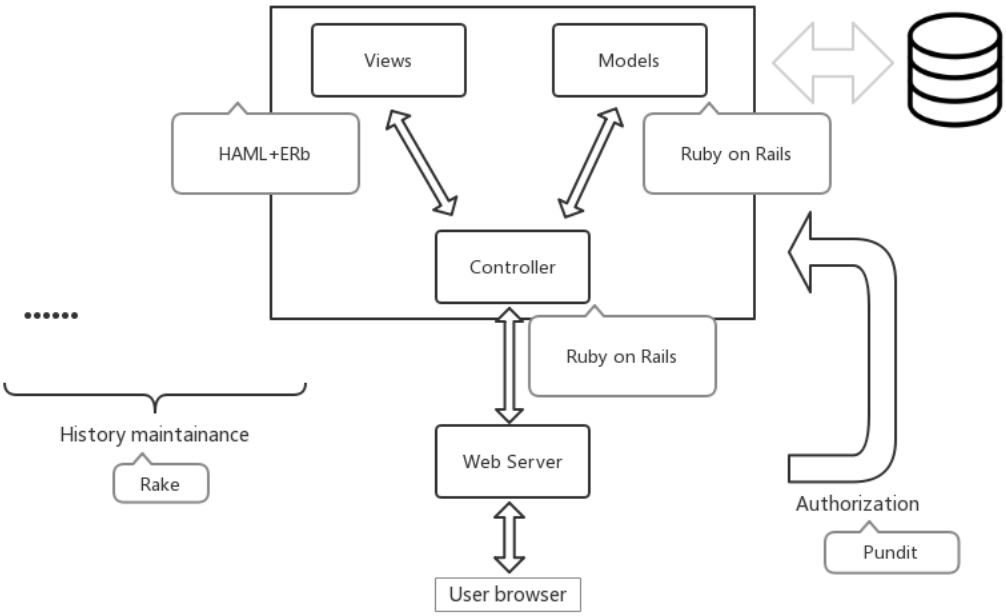
Overview

The overall structure of the system is demonstrated as:



Accordingly, code structure as follows:

`app/controllers`	Code that binds business logic to templates
`app/helpers`	Code that can be used from views, i.e. common operations
`app/lib`	Code that doesn't fit in the other categories
`app/models`	Code that represents data entities
`app/serializers`	Code that generates JSON from models
`app/views`	Templates for generating HTML or other output
`spec`	Automated test suite
`db`	Database migration files
`config`	Configuration files and i18n texts
`app/javascript/mastodon`	Code for the multi-column React.js application
`app/javascript/packs`	Code for non-React.js pages
`config/locales`	Server-side localizations in the YML format

`app/javascript/mastodon/locales`	Client-side localizations in the JSON format

Credit: https://docs.joinmastodon.org/development/overview/

This documents mainly discuss the code inside app directory. The whole Mastodon system is way bigger than current one, but we will only introduce what we modified or appended.

Implementation

Model

We create data structure for global mute list. The model is responsible for managing the data of the application. It receives user input from the controller. Models connect database to the Mastodon system. They directly manage the data, logic and rules of the application. The files are:

```
app/models/global_mute_word.rb
app/models/custom_filter.rb
```

To be specific, each of the files do the validation work and Ruby on Rails will directly retrieve from or save data to PostgreSQL database when running.

Controller

Controller accepts input and converts it to commands for the model or view. In addition to dividing the application into these components, the model–view–controller design defines the interactions between them.

```
app/controllers/api/v1/global_mute_words_controller.rb
app/controllers/global_mute_words_controller.rb
```

For example, in `custom_filter.rb`:

It defines the filters variable for generating views, binds views' action to operation on database.

- To get all filters that is not defined by current user and filters that created by current user:

```
@filters = CustomFilter.where.not(account_id: current_account.id)
@own_filters = current_account.custom_filters
```

- To handle creating new filter operation:

```
  def new
    @filter = current_account.custom_filters.build
  end
```

Views

The view means presentation of the model in a particular format. Any representation of information such as a chart, diagram or table are generated by views. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

There are 2 parts to implement filter in view perspective. One is several pages to do the settings: list, add and manage all the existing filters; the other is to filter the specific status according to all shared filters.

I. In the setting

In Ruby on Rails, views are connected to URL user accessing to. If a user goes to `/filter/new`, Rails will automatically invoke `new.html.haml` file to handle the display of the page. Thus, in our system, several files are needed:

```
app/views/global_mute_words/_form.html.haml
app/views/global_mute_words/edit.html.haml
app/views/global_mute_words/index.html.haml
app/views/global_mute_words/new.html.haml
```

The corresponding pages are `global_mute_words/index`, `global_mute_words/edit` and `global_mute_words/new`.

Besides, shared filter related generation are:

```
app/views/filters/_form.html.haml
app/views/filters/edit.html.haml
app/views/filters/index.html.haml
app/views/filters/new.html.haml
```

Besides, shared filter related generation are:

II. In user feed

To show statuses to the timeline feed, we need to manage all statuses using a manager:

```
app/lib/feed_manager.rb
```

Though user have several contexts needs filter to function: home timeline, notifications, public timelines, conversations, the basic logic is similar: to determine whether a status contains filter phrase. Thus, we design the function to do this:

```
 def phrase_filtered?(status, receiver_id, context)
    active_filters = Rails.cache.fetch("filters:#{receiver_id}")
{ CustomFilter.all.active_irreversible.to_a }.to_a


    active_filters.select! { |filter| filter.context.include?(context.to_s)
&& !filter.expired? }
```

```
    active_filters.map! do |filter|
      if filter.whole_word
        sb = filter.phrase =~ /\A[[:word:]]/ ? '\b' : ''
        eb = filter.phrase =~ /[[:word:]]\z/ ? '\b' : ''

        /(?mix:#{sb}#{Regexp.escape(filter.phrase)}#{eb})/
      else
        /#{Regexp.escape(filter.phrase)}/i
      end
    end

    # This seems to be filtering permanently...
    global_mute_words = GlobalMuteWord.select(:phrase).pluck(:phrase).uniq
    global_mute_words.map! do |word|
      /#{Regexp.escape(word)}/i
    end

    active_filters += global_mute_words


   # active_filters += Rails.cache.fetch("global mute words")
{ GlobalMuteWord.all.to_a }.to_a


    return false if active_filters.empty?


    combined_regex = active_filters.reduce { |memo, obj| Regexp.union(memo,
obj) }
    status         = status.reblog if status.reblog?


    !combined_regex.match(Formatter.instance.plaintext(status)).nil? ||
      (status.spoiler_text.present?
&& !combined_regex.match(status.spoiler_text).nil?)
  end
```

And to design several functions do different context filtering:

```
def filter_from_home?(status, receiver_id)
```

```
def filter_from_mentions?(status, receiver_id)
```

Inside each of the functions, beside of filtering according to shared user custom filters, it also filters status according to whether the status is posted by a blocked user, etc.

API

API to format data and convert them into JSON using serializers. In system, due to complex frontend logic, Mastodon uses Node.js in logic layer. To transfer information between Ruby on Rails and Node.js, and to help other developers, Mastodon uses APIs. Within each API, it will first generate JSON files using sterilizer and then passes json through:

```
app/serializers/rest/global_mute_word_serializer.rb
app/serializers/rest/filter_serializer.rb
```