Web-technologies

Stefano Claes
Stefano.Claes@vub.be

Vincent Mostert
Vincent.Koen.Mostert@vub.be

Leander Lismond
Leander.Louis.Lismond@vub.be

December 21, 2020

Contents

Abstract

This report describes our work on a social media application. The core idea involves having a location tag to each of the posts and having some sort of a local news aggregator.

The described application is deployed at `https://nieuws.naamloze.website` as it was submitted.

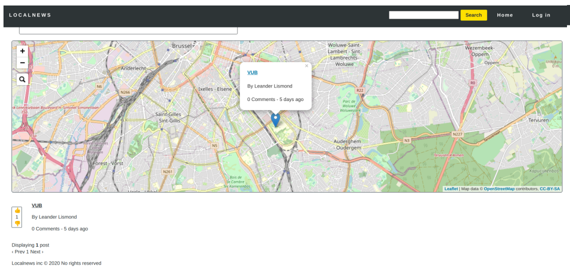
1 Functional targets

1.1 Home Page

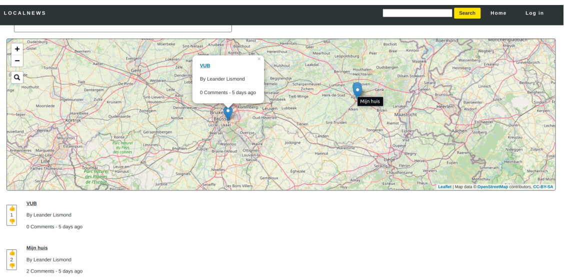
As a user it is possible to submit posts and comments as on a conventional forum. The difference here is that each post has a location. This raises the possibility to view and comment to posts that have a certain location. The way we use this feature is by using the Leaflet map that is visible on certain pages. This gives the user the option to view more posts or less posts concentrated on a smaller location.

There is also a search box to filter posts by content. Both querying mechanisms are combined, meaning that only posts that match the search query *and* are within the visible region on the map will be shown. One exception is on a visit to the posts page without queries. The system will then show all posts, but only until the map is moved or a query entered.

The posts listed are sorted using a combination of the vote count and the creation date.



(a) The home page when using a small area



(b) The home page when using a large area

Figure 1: A comparison between using a small and large area on the leaflet map.

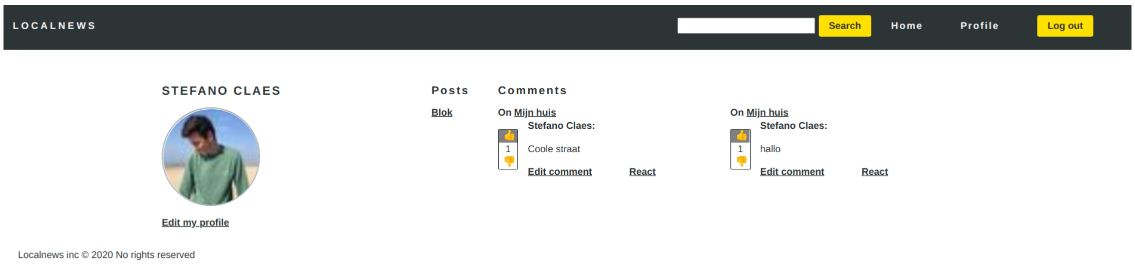
As seen on figure 1 it is clear that more post will be visual according to the range of the map.

1.2 Post import

By entering a URL into the source field it is possible to import pages from other websites. The website will try to extract a title and body from the webpage. On certain sites the system is also able to import the location.

1.3 Profile Page

Every user has the ability to view his account, The profile page contains all past comments and posts. Besides this it is also possible to edit your profile by clicking on Edit my profile.



On Mijn huis
Stefano Claes:
1
Coole straat

On Mijn huis
Stefano Claes:
1
hallo

Localnews inc © 2020 No rights reserved

Figure 2: A profile page generated by using Facebook sign up

As seen on figure 2 the Facebook sign up will automatically use my name, profile-picture and e-mail to make up an account.[1]

1.4 Profile Edit Page

The edit page gives the user the ability to change certain information about the user. The user can alter folowing elements:

1. The username.
2. The used avatar on the site.
3. The e-mail adres.
4. The used password given the old one is known.

2 Non-functional targets

2.1 Media Queries

The website uses media queries. These media queries are needed if the developer wants the site to look nice on desktop and on a phone. The resolution is different and so will be the layout. Hence media queries to filter upon the screen resolution.

2.2 Javascript

This website, like many others uses JavaScript. Not all clients however, (want to) support Javascript. Because of the way the Rails framework works it was possible for us to build one application for both server-side rendering and client-side reactive updating.

2.3 Accessibility

The website is designed to be user friendly and easy to use. The website is understandable because of the row column usage. Every big component is divided into a row layout or a column layout. This way we prevent the content of the website to be clustered and difficult to read.

[1]It is important to note that the deployed version of this project is not validated by Facebook, and thus it is impossible to sign up using an unvalidated account.

Navigating through the website is made easy because the links are underlined and change color. The navigation bar hosts links to, with enough margins to be able to read the titles. The interaction with the website is fun and easy. When successfully logging in, the user gets a message on the screen he/she did it well. The buttons change color when hovering above them. Confusion for color blind users has been avoided to the maximum by avoiding the color blindness pairs (when using yellow not using blue) and adding enough contrast.

2.4 Web 2.0

The website is web 2.0 meaning we emphasize on user generated content. The page will look better when users use the website and write their own content. The site has little content on its own except from accessibility features. This invites the user to contribute to the sites content.

3 Frontend

3.1 AJAX/WebSockets

The frontend is both available as a reactive PWA, and as a stateless, Javascriptless webapp. Without Javascript the maps are not shown, and may features require a page visit, but the core functionality still remains. We encourage the reader to try using the site with Javascript disabled. Especially on low-resource or legacy clients this can be very useful.

When Javascript is enabled, the overall responsiveness of the website is severely improved. Using Turbolinks all visited pages are cached, and new visits won't have to re-render the page. To further increase the reactivity we used several libraries:

3.1.1 CableReady

CableReady is a Ruby/Rails utility to send DOM[2] updates over websockets using ActionCable. We can essentialy 'push' several kinds of DOM manipulations to the client. This allows us to respond multiple times to a single request.

3.1.2 Stimulus Reflex

Stimulus Reflex uses the same websocket connection as CableReady to send real-time updates from the client to the server, not unlike an RPC. This allows the server to then react to these stimuli. By default SR[3] will run the reflex in the context of the controller of the current page. After the reflex is done, SR will use the new state and the controller to re-render the page, and send it back over to the client. The client will then use `morphdom` to walk the DOM-tree and patch the changed items.

This system effectively allows use to use the same systems for both full page renders and partial updates, achieving the progressive enhancement we desire.

3.1.3 StimulusJS

StimulusJS, or just Stimulus, is a client-side Javascript framework that uses MutationObserver to attach controller instances to page elements. This allows us to still modify the DOM using other means, and react to those modifications. The attributes used by Stimulus all live in the `data-` namespace, keeping our HTML as valid and usable HTML when Javascript is disabled.

3.2 Map

The map API we used is called Leaflet. We chose Leaflet because it is very light and has everything we would need from a map, and it is free. We have a stimulus controller to update the map with the statically rendered/AJAX injected data. This happens in `/app/javascript/controllers/map_controller.js`.

[2]Document Object Model
[3]Stimulus Reflex

On the posts search page we additionally maintain a binding between the URL and map position. This way, users can reliably share, bookmark, and reload the page while still viewing the same content. This also means that, with Turbolinks, the map state is stored in the browser history. Another advantage of using Turbolinks is that the same map element can be reused for the posts search and detail page. Switching between the two will animate the map from one state to the other (within reason[4]).

When the map is moved, an empty reflex is triggered on the server. This will rerender and morph the page. By marking the map element as permanent but not the surrounding controller, the controller will be re-run when the page changes, but the map will stay loaded. The controller then (lazily) initializes the map and markers.

3.3 Voting

The voting components on posts and comments are both forms and Javascript-enabled elements. When javascript is enabled they will immediately reflect the new state. When using the forms they will try to redirect to the same page.

3.4 Comments

The operations on comments will display their UI elements inline. This is accomplished using ViewComponentReflex. Using this library we can write components with their own reflexes. By storing the component state in the HTML we still have a single source of truth for the view, instead of maintaining a separate state using Javascript in the client. All of the comment operations are implemented in about 100 lines of code.

3.5 HTML5

3.5.1 Header

The HTML `<header>` element represents introductory content, typically a group of introductory or navigational aids. We used the header element to house our information and introduction of the posts.

3.5.2 Main

The HTML `<main>` element represents the dominant content of the `<body>` of a document. We used the main element to house our body information of the posts.

3.5.3 Footer

The HTML `<footer>` element represents a footer for its nearest sectioning content. We used this footer to mention our licence. For our licence we used like many sites including Wikipedia *Creative Commons.*

3.5.4 Nav

The HTML `nav` element represents a section of a page whose purpose is to provide navigation links, either within the current document or to other documents. The navigation bar uses a checkbox to slide the navigation bar out when in phone display. The navigation bar contains an unordered list `<ul>` with list items `<li>`. The list items are: Home, to navigate to the homepage and Profile or Login, to navigate to your profile page if you are logged in, if you are not logged in this list item will display Login and will navigate you to the login page.

[4]With large zooms or pans the map will simply ‘jump’, as it doesn’t make sense to animate these kinds of transitions.

3.5.5 Summary, Detail

The HTML Details Element `<details>` creates a disclosure widget in which information is visible only when the widget is toggled into an "open" state. The HTML Summary Element `<summary>` creates a summary or a label in the disclosure widget. We use `<summary>` and `<detail>` tags to add collapsible comments.

3.5.6 Section

The HTML `<section>` element represents a standalone section. Usually these sections are used to house tags like `<h1>`,`<h2>` and `<p>`. We used sections for the user page. The user page is divided into three sections: The profile section, the profile section and the comments section. It gave us a clearer view of the layout of the page and made it easier to access the content with CSS.

3.6 CSS

We chose not to use bootstrap because it is known to be quite heavy and we wanted to minimize the data usage. We integrated a lot of CSS, mainly to make the website look nice and responsive. We also used SCSS, to make some variables we can use in the whole project and use nested rules. We used flex box and grid to distribute our content on the website.

4 Backend

This is a Ruby on Rails application, using a PostgreSQL database and Redis for caching and session storage. In production, we use an S3-compatible object store to keep the images, and variants of those images.

4.1 API

The api is documented at `https://nieuws.naamloze.website/apipie` or under `/apipie` on a self-hosted instance. Be sure to add a `.json` extension or set the correct `accept` header to receive an API response, and not a webpage.

To authenticate against the API one can use an OAuth client. The OAuth endpoints reside under `/oauth/`. Registering an application can be done using any account. The application management interface is provided by the `doorkeeper` gem and is available at `/oauth/applications`. This link is not available in the application itself to prevent spam and confusion. Submit your OAuth token using a `Token: Bearer YOUR_TOKEN` header.

All api endpoints will also accept a normal user session for authentication, this can be used for experimenting with the API from within the browser.

4.2 Metadata

The backend renders pages with embedded opengraph data. Because of the server-side rendering, all pages are easy to crawl.