newsR: A Responsive Newsreader Web App

Paul Ganster, Peter Grassberger, Magdalena Mayerhofer, Ana Lopez Camarero

706.041 Information Architecture and Web Usability WS 2019/2020
Graz University of Technology

3 Feb 2019

Abstract

newsR is a fully responsive web-app to read newsgroups with all the basic features. With newsR one can subscribe to newsgroups, read threads and write new articles. The website was designed mobile first and with the integrated PWA features like add to home screen it nearly feels like a native app on a smartphone. However, it was not designed mobile only and therefore it is also perfect for checking the latest news on the operating system of choice. State of the art web technologies are used to provide an application like feel. Communication with news servers is facilitated over WebSocket and a thin web server.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

newsR is a responsible newsreader Web App implemented as a Progressive Web App for a richer native experience. The project is implemented as a brand new application written in Typescript using React, SCSS and Newsie for NNTP communication. A newsR logo was also created, as can be seen in Figure 1.1.

1.1 Short History

Usenet's Newsgroups appeared in the 80s when two students of Duke University were looking for an efficient way of sending and sharing information taking into account that it should be organized by topic. Therefore, Usenet, the users' network, is the internet's prime discussion area where all the newsgroups are organized like a hierarchy by subject. The content of this network is based on the posted information by the newsreader client which are the people subscribed to the channel. In the application that has been created, several factors were taken into account such as user experience, responsive design and a good structure in order to create an efficient and functional navigation.

1.2 Step before implementation

Before the creation of newsR, an analysis of already existing newsreaders was done. This was necessary in order to get some points that should be avoided or must-haves the app could focus on. The ideas that were taken consist of a well-structured app where all the processes (subscribing to a newsgroup, reading and writing, looking into the own clients list of newsgroups) are divided so the use of the app is very efficient and accessible for the user.

Figure 1.1: Logo of newsR

Chapter 2

Theory

Newsgroups work as a forum accessible through Usenet. Each newsgroup is specialized in a topic which can be determined by its name and description. This is possible due to its top-level hierarchical organization which consists of a prefix that indicates to which topic is related. Usenet has 8 principle prefix: comp.* (Discussion of computer-related topics); news.* (Discussion of Usenet itself); sci.*(Discussion of scientific subjects); rec.* (Discussion of recreational activities); soc.*(Socializing and discussion of social issues); talk.* (Discussion of contentious issues such as religion and politics); misc.*(Miscellaneous discussion—anything which does not fit in the other hierarchies); hum.*(Humanities discussions). Additionally, another different prefix with smaller distribution can be found such as alt.* (alternative discussions) or, for example, the ones that classify the newsgroups by regions or countries. Another feature of newsgroups is that they can be moderated by a human moderator that needs to approve the post before it appears in the newsgroup or regularized by some rules or protocol that set what can or cannot posted. Additionally, they can be exclusively for a region or widely broadcast/open and some software offer the option of responding privately (directly via email) or publicly in the newsgroup. That said, it is important to know how they work:

1. Configuring the News Reader settings: The usual settings are news server, name that will appear as the sender, email address, and, optionally, the recipient email.

2. Finding and subscribing to newsgroups: Once you are connected to a server it will show all the newsgroups it has. This part of the process is fast because it is only required to find the group the user wants to subscribe to and a click of a button to join the group. Usually the group can be found by its key words or by name, but it is not possible to know if they are active or not.

3. Reading Article: Once subscribed to one newsgroup generally the software downloads all the headers of the group and if a message is selected it will download the content of it. Some newsreaders preserve it for offline access but that depends on the newsreaders.

4. Posting Article: It is the equivalent of sending an email, so you have to options:
   - Start a new thread
   - Follow up one existing

   Some newsreaders have the possibility to quote some parts of the article content to respond only to the part you would like to refer to.

5. Cancel Article: Some servers will have the option of taking down what a user has posted but it can cause problems if someone has already responded because it will leave these messages as orphans.

Before the implementation of the newsreader it is necessary to make an analysis of some other newsreaders in order to get an idea on what to focus, what to avoid and what to take into account. Different kinds of newsreader can be found, and a classification can be made according its interface or

the client type. The latter will create six different kind: Text newsreader (mainly for reading/posting text post), traditional (capable to read text and binary attachments), binary grabber/plucker), NZB downloader (binary grabber client without header support – cannot browse groups or read/post text messages; can only load 3rd-party NZBs to download binary post attachments Wikipedia 2019), binary posting client (made exclusively for posting multi-part binary files) and combination client (supporting text reading/posting, as well as multi-segment binary downloading and automatic Par2 processing). On the other hand, according to the interface classification, a shorter categorization is found graphical, text-based or web-based. Graphical is the most common one. In this case, this newsgroup reader is a traditional one implemented with a graphical user interface; therefore, it has the possibility of attaching any kind of binary document. Moreover, it will be necessary to consider some information architecture and web usability aspects. Regarding to the information seeking behavior, the kind of search that may be found is the explanatory search due to the user's knowledge what he or she is looking for but another it is open to find more information. In newsreaders groups the clients can ask or search for certain question but also find other responses that could be interesting. The same happens with the process of looking for a newsgroup; the user could look for a specific theme but finds lots of other interesting newsgroups that she could join. Due to this last idea it is important to have a clear scheme of the newsreader. The more structured and clear it is the easier will be for the reader to find what she is looking for. Overall, the organization of the page is intended to have a heterogeneity ambiguous organization task based scheme, because the main idea is to divide the spaces according to the main different process that can be found: finding and subscribing to newsgroups, getting the groups that the user is subscribed to and the reading and posting process. This last point leads to the navigation and visual design which main objective is to make the user experience functional, comfortable and easiest as possible.

The following step consists of comparing some existing newsreaders in order to make the best profit by analyzing the strength and weakness of them. Regarding some technical aspects of the newsgroup reader, it is necessary to know that one of the most important points to take into account is the User Interface (UI) for reading/writing. In this case one of the most important Usenet providers is EasyNews due to its easy access for users and its high-speed download of the header and information. Therefore, other points that should be considered are the fast and easy accessibility and the efficient message download. Another newsreader that it could be considered is Grabit which is known for being able to download articles without the need of downloading the headers and for the possibility of downloading multiple articles at the same time. According to the "look and feel of User Interfaces" analysis of this newsreader, it was necessary to make an analysis of other readers in order to see what should and what should not be done while creating a newsreader app. In this case, we compare the different styles with three free newsreaders: Knode, Thunderbird and Pan. As it can be seen in the images the structure of all the readers are very basic (see in Figure 2.2): A left sidebar where all the newsgroups the user is subscribed to can be seen, another one to the right with all the headers of a newsgroup (they are shown when one newsgroup is selected) and then a bigger "box" at the bottom where you can see an entire article if one is selected. Regarding the article that is being read, all of the readers have a header that gives some information about who wrote the article, when it was written, the newsgroup where the article comes from. Additionally, some of them have further fields such as subject of the article. In relation to the subscribing and unsubscribing process, Knode and Thunderbird have an additional window where the newsgroups can be found and where the user can subscribe see in Figure 2.1. Once the user is subscribed to a newsgroup it appears in the left sidebar. On the other hand, Pan has another kind of style (see in Figure 2.3) which consists of a sidebar in the left part that has the option to show the subscribed newsgroups or all the possibilities the user has. This last option helps to accelerate the process of subscribing and unsubscribing to newsgroup. To sum up, all of them have the possibility of filtering by names or keywords when the user is looking for a newsgroup.

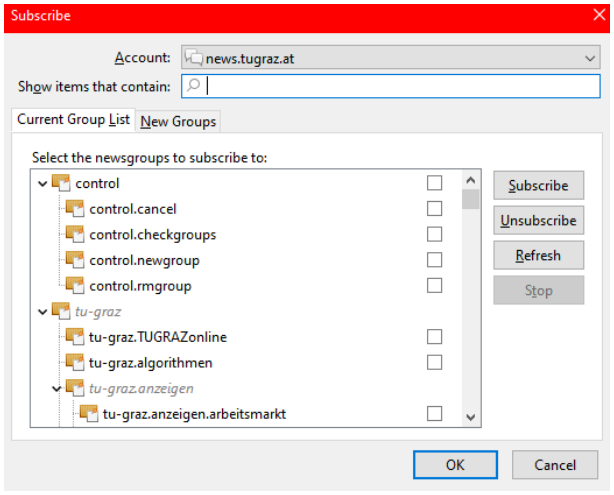

Figure 2.1: Thunderbird and Knode Subscribing to newsgroups process [Screenshot taken by the authors of this paper.]

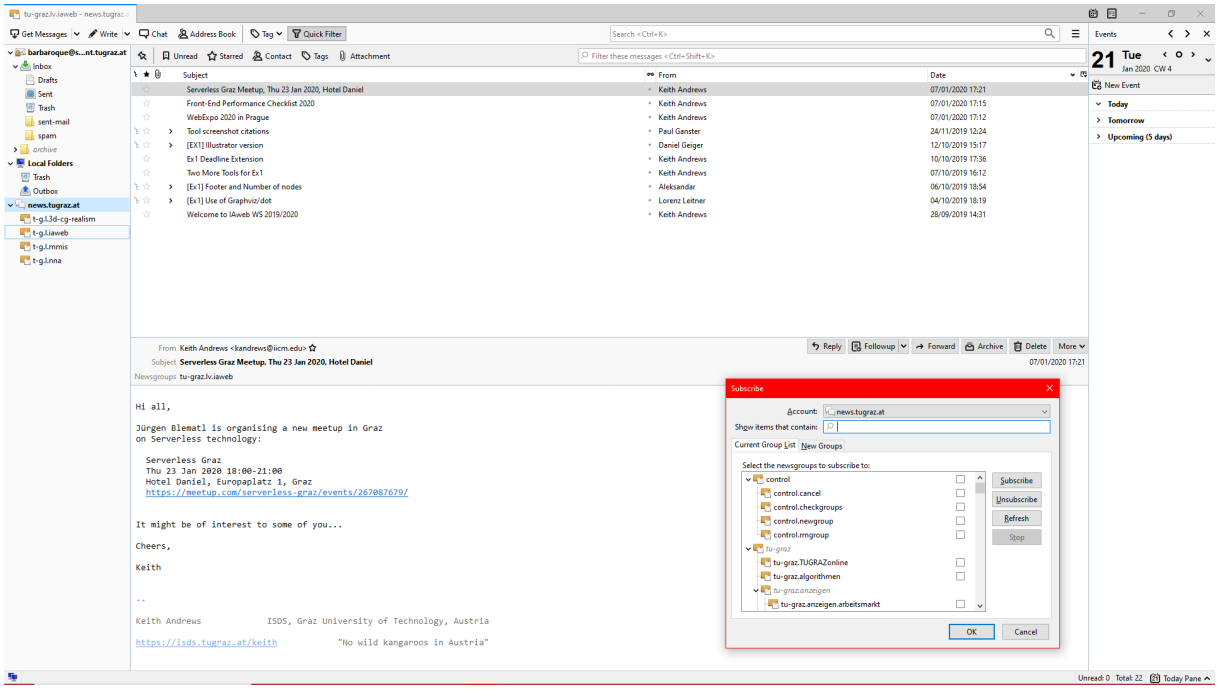

Figure 2.2: Thunderbird visualization model [Screenshot taken by the authors of this paper.]

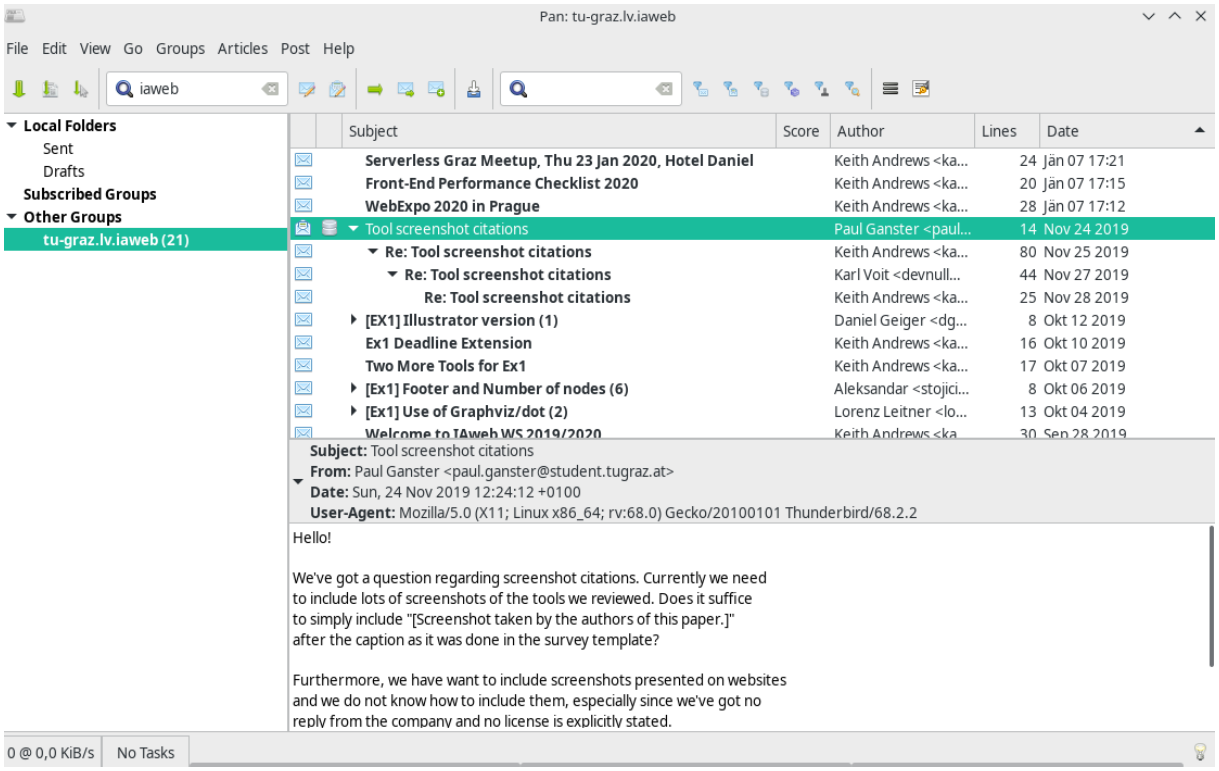

Figure 2.3: Pan windows visualization Subscribing to newsgroups process [Screenshot taken by the authors of this paper.]

Chapter 3

Technology

The following will explain the technologies used during the creation of the application.

1. Typescript is a free and open source programming language developed and maintained by Microsoft in 2012. It is a subset of JavaScript that specially add static types, objects based in classes. Typescript is compiled into Javascript to run in the client or server side.

2. React, also known as React.js or ReactJS is a popular JavaScript framework released in 2013 and developed by Facebook. Designed to create user interface and to facilitate the creation of single-page applications. It has Stateful Components, State and Lifecycle, Composition over Inheritance and JSX (syntax extension of JavaScript).

3. NNTP stands for "Network News Transfer Protocol" which is the protocol used to connect to Usenet servers and transfer newsgroup articles between systems over the Internet. The library Newsie was used to talk NNTP.

4. SCSS is metalanguage that adds some extra features on top of the original CSS to give more power and elegance to the basic language. It improves maintainability for larger projects, for example be allowing nesting statements.

5. Progressive Web App was rolled out by Google in 2015 that offers rich user experience, such as push notifications or Add to Home Screen (A2HS). It offers native app-level immersive experience on both Android and iOS.

3.1 Network News Transfer Protocol (NNTP)

To get a deeper understanding of the inner workings of a newsreader one needs to look at the underlying protocols and standards. The Network News Transfer Protocol (NNTP) is the language in which news clients talk to servers and that servers can also use to talk to each other and exchange messages. The news server holds all the articles that users write and send to a server directly or that a server receives from another server. Articles are organized in Newsgroups, or also abbreviated to groups.

NNTP was first standardized in 1986 as rfc977 RFC977 1986, but many more RFCs followed, as shown in Table 3.1. After this first standard more were created, to define the message exchange between servers RFC1036 1987 and common extensions Barber 2000. With rfc3977 Feather 2006 the old rfc977 was replaced. Later standards add TLS for encryption on transport layer Murchison et al. 2006, additional commands ÉLIE 2010 and compression Murchison and ÉLIE 2017. There is also a standard for two url schemes: `news://` and `nntp://`.

The NNTP protocol uses so called commands with responses to request information from the application programming interface (API) endpoint on the server or send information to the server. A few example commands are:

RFC number	RFC Name
977	Network News Transfer Protocol
1036	Standard for Interchange of USENET Messages
2980	Common NNTP Extensions
3977	Network News Transfer Protocol (NNTP)
4642	Using Transport Layer Security (TLS) with Network News Transfer Protocol (NNTP)
5538	The 'news' and 'nntp' URI Schemes
6048	Network News Transfer Protocol (NNTP) Additions to LIST Command
8054	Network News Transfer Protocol (NNTP) Extension for Compression

Table 3.1: RFCs about NNTP

- GROUP
- ARTICLE
- POST
- QUIT
- LIST NEWSGROUPS
- LISTGROUP
- OVER

For example `LIST NEWSGROUPS` returns a list of newsgroups that are provided by the server, `OVER` gives basic information on articles available in a specific group. Articles can be requested with `ARTICLE` but there are also commands to only request the header or body of an article. `POST` sends a new article to the server and `QUIT` closes the connection.

NNTP is not a fully stateless protocol as a developer would for example know and expect from a REST interface. Some commands depend on previously send commands in the same session, but most of them are independent of previous commands. For example posting articles needs two commands send after each other and depend on the group that was previously selected.

NNTP sits on top of the Transmission Control Protocol (TCP), a client like a newsreader needs to open a TCP socket to talk to the server. Websites unfortunately don't have the capability to make the browser open TCP sockets and therefore can't connect to NNTP servers directly. We came up with three different solutions to deal with this limitation:

1. Use an intermediate server that the browser can connect to via WebSockets that in turn opens a TCP socket and passes requests and responses to and from the NNTP server as shown in Figure 3.1 or
2. add WebSockets support directly to NNTP servers or
3. build a native app on desktop or mobile platforms instead of web app that does not have this limitation.

The first solution seems most feasible for quick implementation and widest compatibility with servers. Extending servers would be a longer process, as there are some different NNTP server applications that exist.

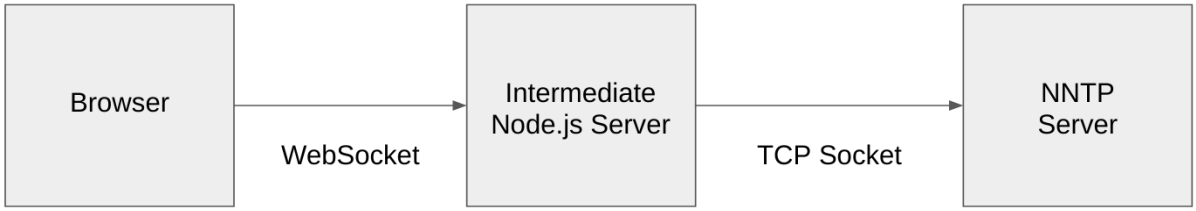

Figure 3.1: NNTP Intermediate Server

3.2 Progressive Web Apps

Progressive Web Apps MZ 2020 are web applications that use modern APIs to achieve a multitude of advantageous properties. MDN defines these advantages as safe, re-engageable, responsive, progressive, network independent, linkable, installable and discoverable. Instead of relying on a single API, a PWA depends on multiple technologies to achieve these properties.

- The **Service Worker API** is the core of a PWA. A service worker is responsible to intercept requests to the server and cache them. It furthermore handles offline availability and has to retrieve content from the cache if the user is currently online.
- The **Cache API** offers a possibility to cache `Request`, `Response` pairs directly that are returned by the **Fetch API**.
- Within a **Web App Manifest**, a developer can specify meta data about a web application that is then used for local installation. This meta information can consist of the short/long name of the app, high-definition logos and even background color for the splash screen during start up.
- Using the **Push API**, one can push notifications from the server to the application.
- These notifications can then be shown with the **Notifications API** which is responsible for displaying notifications on the device.
- **Add to Home Screen**, or short A2HS, is a feature in modern web browser that enables the user to add a PWA to the start screen of the smartphone. When the app is then launched from the start screen, it is contained in its own container and treated as an own app by the smartphone.
- Some features are **HTTPS** only, meaning APIs such as the service worker and A2HS will not work when the web app is launched on an insecure connection.

3.3 Create React App

Create React App FB 2020a is a powerful utility to easily bootstrap a React application. It comes as a bundle of necessary frontend build tools and predefined npm scripts. The only command that needs to be issued is

```
npx create-react-app my-app --template typescript
```

The build tools are composed of a development server, a typescript compiler, a production build (assets optimization), a test framework (jest FB 2020b) and much more. Even a service worker is provided by Create React App, which is not enabled by default. The predefined `package.json` comes with useful scripts, such as `npm run start` to start the development server with hot reloading, `npm run build` to create a production ready optimized build, `npm run test` to start the test suite and finally `npm run eject`. The latter removes the bundling of Create React App and adds to build tools directly to the project as dependencies. This offers the broadest customizability, for large applications.

Chapter 4

Features

4.1 Newsgroups List Views

In newsR there are three different views to display newsgroups: In Figure 4.1 the subscription view can be seen. If newsR is opened in the browser, this is always the first view the user sees. The subscription view is important, because it is not practical for the user to always need to scroll through all newsgroups to find the desired one. In this view, only the newsgroups which a user has subscribed to can be seen. The subscriptions are saved in the localStorage of the browser. If a user has not subscribed to any newsgroup a text is displayed, which tells the user what to do. The second newsgroup view can be seen in Figure 4.2. This is the view, where the user can manage the subscriptions by selecting or deselecting groups from a list with all groups. The third view displays all newsgroups, so a user does not need to subscribe to a group to visit it. This view can be seen in Figure 4.3.

4.1.1 localStorage

`localStorage` is a way to save data form a Website in the browser. One can write key-value pair strings to it. In newsR `localStorage` is used to save the subscriptions as well as the identifier of the already read threads. When a user posts to a newsgroup, the name and email is saved to `localStorage` too.

4.2 Thread Overview

A thread is a term not known in the NNTP standard. Articles belong to groups and can contain references to other articles for which they are a follow-up article. We consider a thread to be a collection of articles starting with a first article that does not follow any other article and all articles that follow it directly or indirectly in a chain of follow-ups. These hierarchies of articles are structured in the newsR client after they are loaded from the server or the local cache.

The view where the threads are displayed is shown after a user clicks on a newsgroup. In Figure 4.4 the split view, which is only available on bigger screens, can be seen. On the left side there is the overview of all threads, on the right side you can either see the selected thread or just a text with some information, when no thread is selected. On smaller screens (e.g. mobile screens) the split view would not fit anymore in a usable way. This is why the view is split into two separate screens. This can be seen in Figure 4.5. When a user clicks on a specific newsgroup the basic thread information for all threads are loaded from the newsgroup server. The content of the thread is not loaded until actually clicking on a specific thread. This is further explained in Section 4.3. In Figure 4.5 it can also be seen, how a read and a unread thread are displayed differently. All unread threads are in bold, while all read threads are displayed in normal font weight.

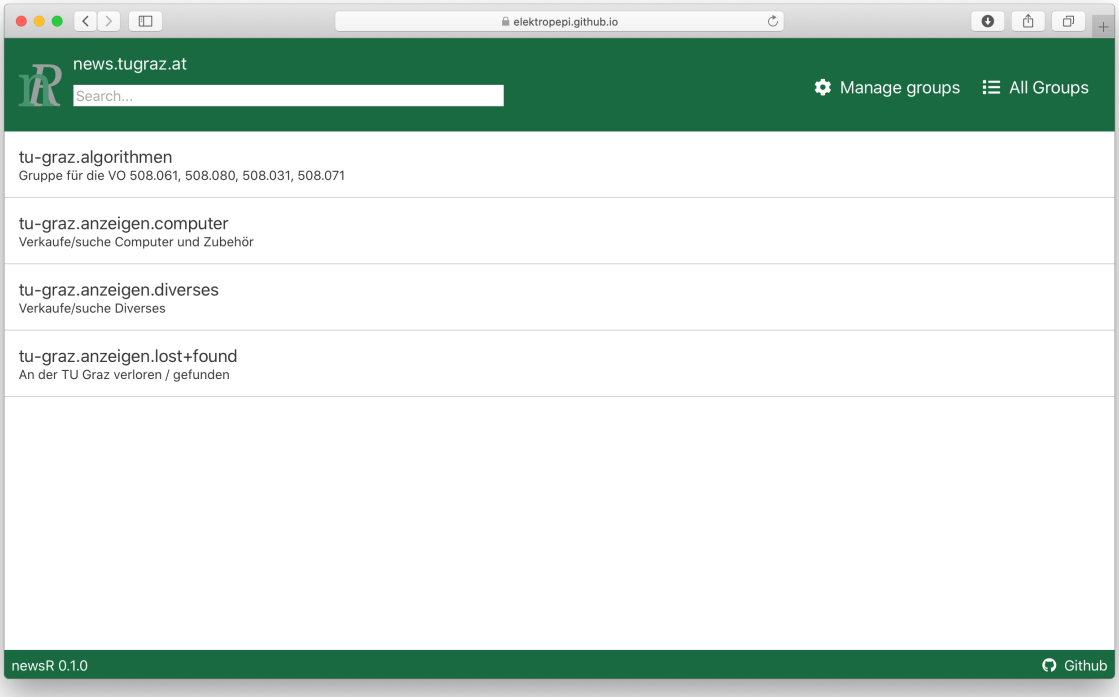

Figure 4.1: Subscription View

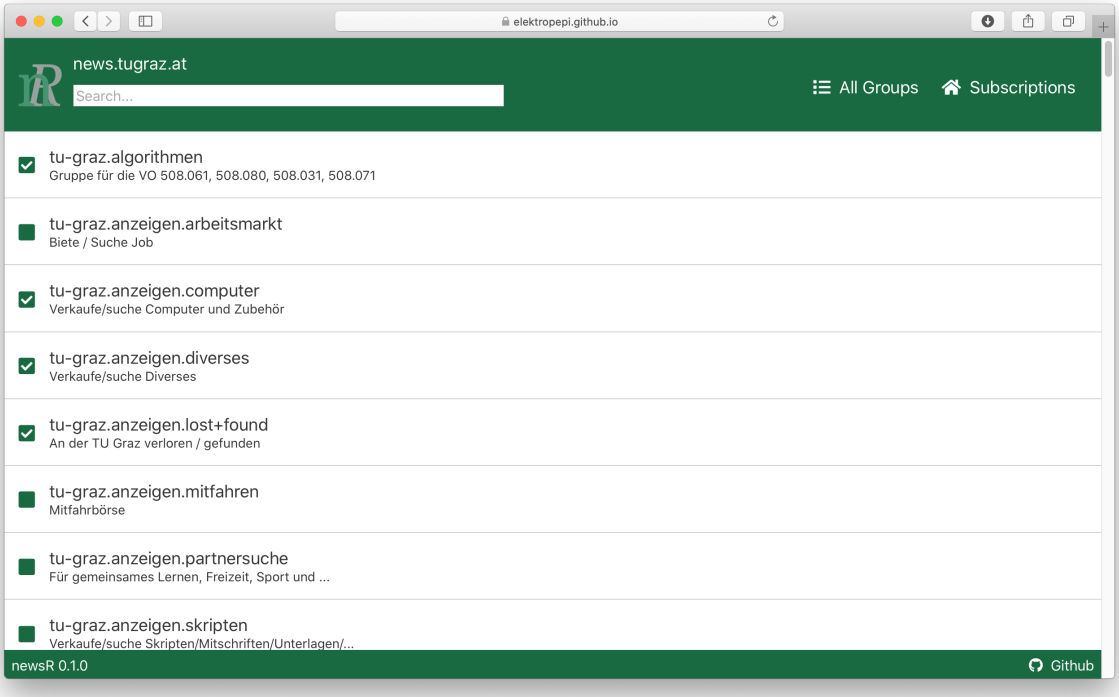

Figure 4.2: Manage Groups View

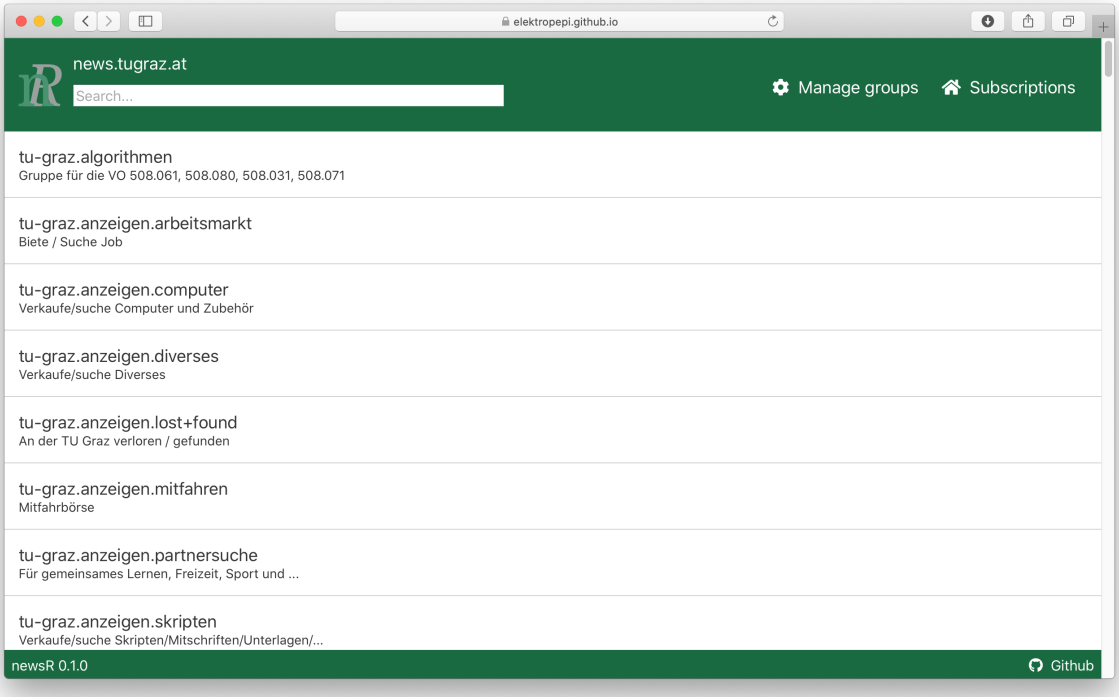

Figure 4.3: All Groups View

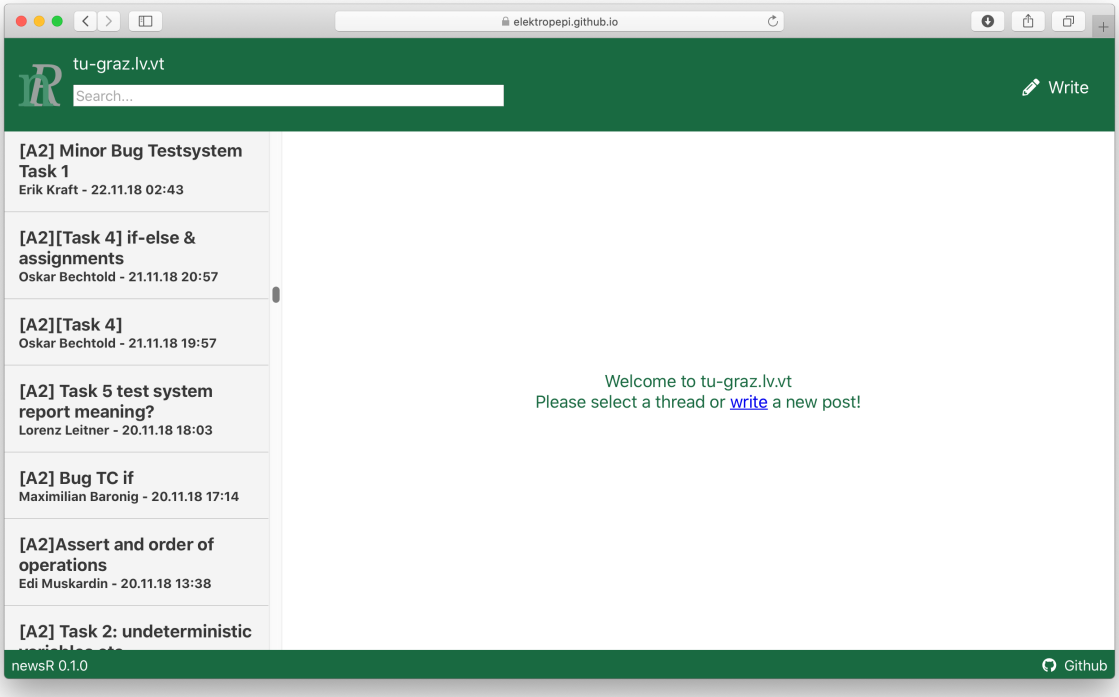

Figure 4.4: Thread Overview

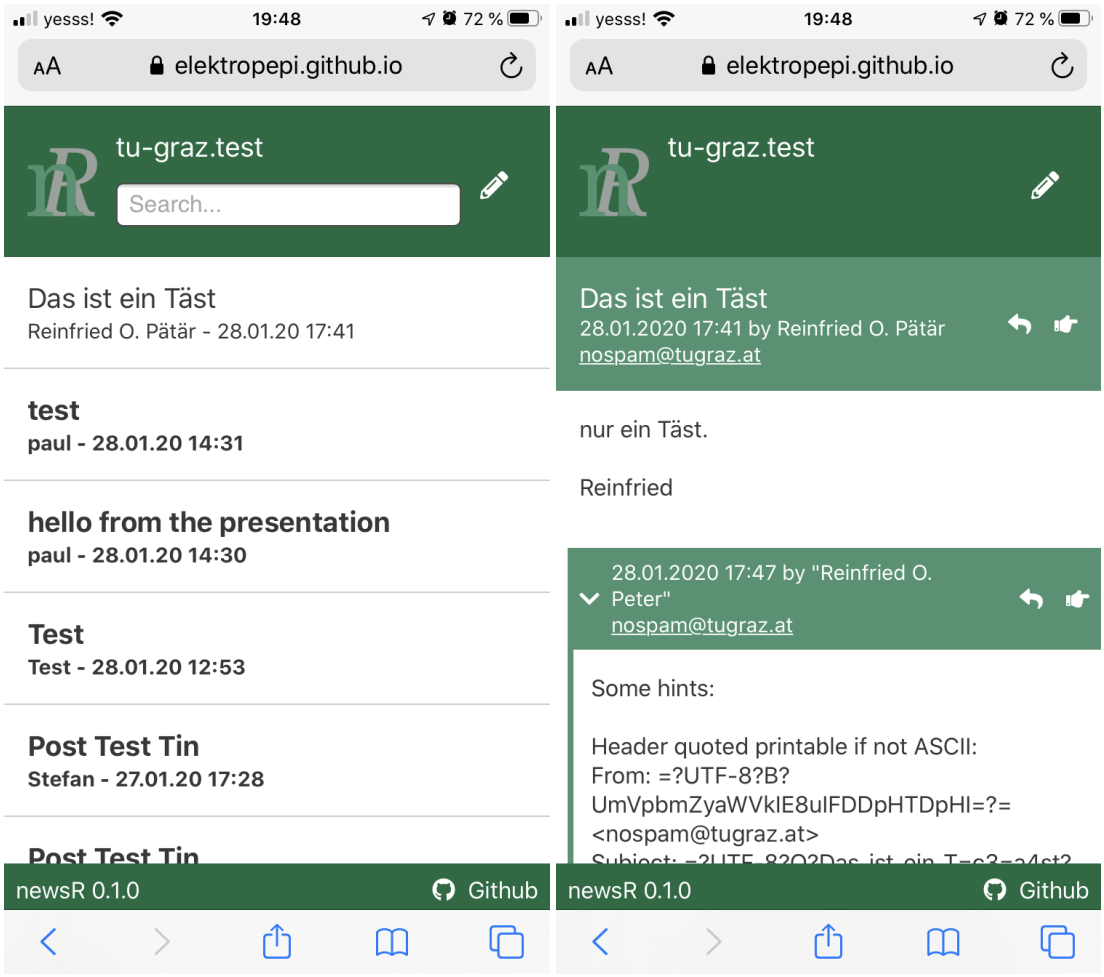

Figure 4.5: Mobile Thread Overview and Thread Content

4.3 Thread Content - Response Hierarchy

As it is common in social discourse, statements will yield responses and even turn into long lasting discussions. The same of course goes for newsgroup threads with their follow-ups. Any thread or follow-up may receive further follow-ups, therefore creating a hierarchical structure of follow-ups per thread. This constellation proved to be quite a challenge in displaying in a responsive matter. Small screen sizes have to be kept in mind. Also, some usability problems have to be taken into consideration, such as the user knowing which follow-up belongs to which thread / follow-up.

Upon selecting a thread, newsR dynamically loads its body content (only basic thread information is pre-loaded for every thread, such as subject, date, author etc.). The basic thread information is already shown in the header during loading to give the user immediate feedback that he correctly has selected a thread and now only needs to wait for the body to finish loading, as can be seen in Figure 4.6.

The thread information is shown in a header directly below the app header, which can be seen in Figure 4.8. This thread header has the color `$secondaryGreen` to indicate "less importance" than the `$primaryGreen` colored app header. Within the thread header, the subject is displayed at a slightly larger font-size, `1.2rem`. Lesser information such as thread date, author and mail, is displayed beneath the thread header in a slightly smaller font-size, `0.9rem`. The author mail is clickable and opens the native mail program using a `mailto:` anchor. On the right side of the thread header two buttons are shown: reply using the standard reply icon and follow-up using a hand icon. The first button also opens the native mail program, whilst the latter opens the post view, where the user than can write a follow-up on that thread. Section 4.5 will describe that in more detail.

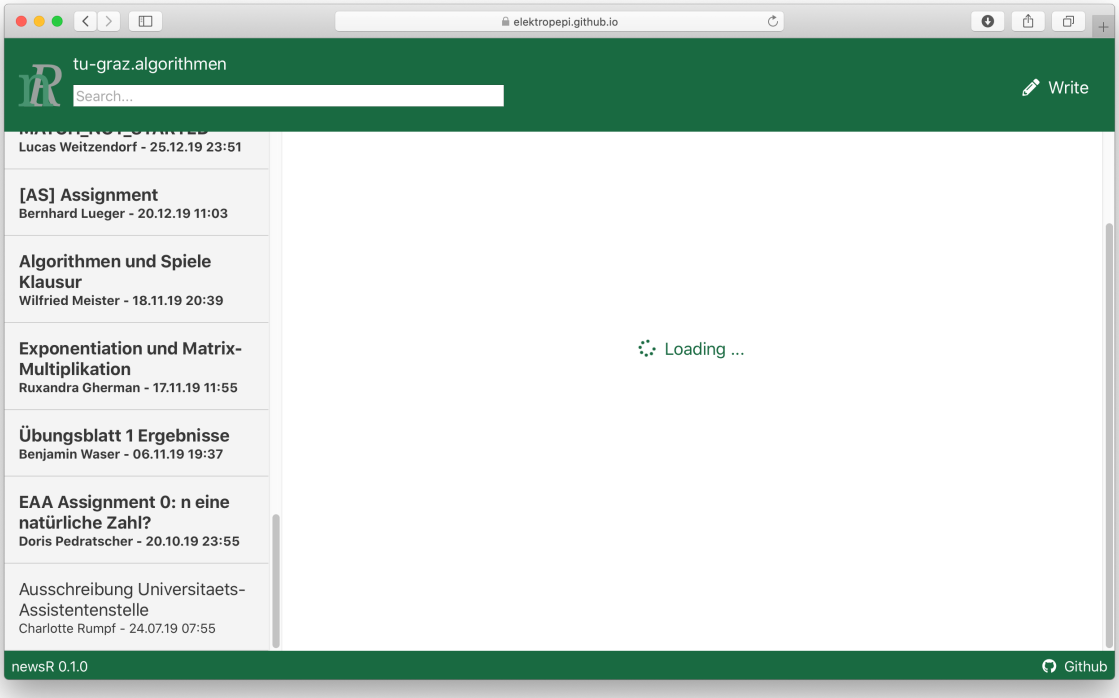

Figure 4.6: All Groups View

The thread body content, after it has been loaded, is displayed right beneath the thread header with a white background for better readability. Possible attachments will be displayed directly beneath the body, see Section 4.9. At the bottom of the thread detail view, the follow-up hierarchy is displayed. This hierarchy is a recursive inclusion of follow-ups with their follow-ups and so on. The deeper in the hierarchy, the more the indentation on the left side increases so the user can distinguish each follow-up to where it belongs. In addition to indentation, a `0.25rem` thin line is shown on the left side per level which assists the user in associating follow-ups with hierarchy levels.

In essence, each follow-up is a slimmed down version of the thread view: The thread header includes the same elements, such as follow-up date, author, buttons but without the subject (as it's mostly "Re: <thread subject>"). The padding in this follow-up header is decreased to emphasize lesser importance than the thread header. The smaller padding and exclusion of the follow-up subject enable the header to be of a compact size, which is especially useful for mobile devices. Below the header, the body content is displayed as usual with possible attachments. Afterwards, a hierarchy of possible follow-ups to that follow-up is shown again, but now with increased indentation. Clicking on the follow-up header will collapse everything belonging to it, such as body content, attachments and its follow-ups. This interaction enables to user to efficiently navigate through the thread hierarchy. A chevron on the left side of the follow-up header indicates whether or not the follow-up has been collapsed or not. Clicking on the thin line on the left side also collapses the corresponding follow-up. Figure 4.7 shows such a view of a thread hierarchy.

4.4 Article Caching

News client desktop applications normally load all articles with their body content for all subscribed groups on the start of the news client. The loaded content is retained on disk, doesn't need to be loaded from the server on the next startup and is saved in case it is no longer provided by the server.

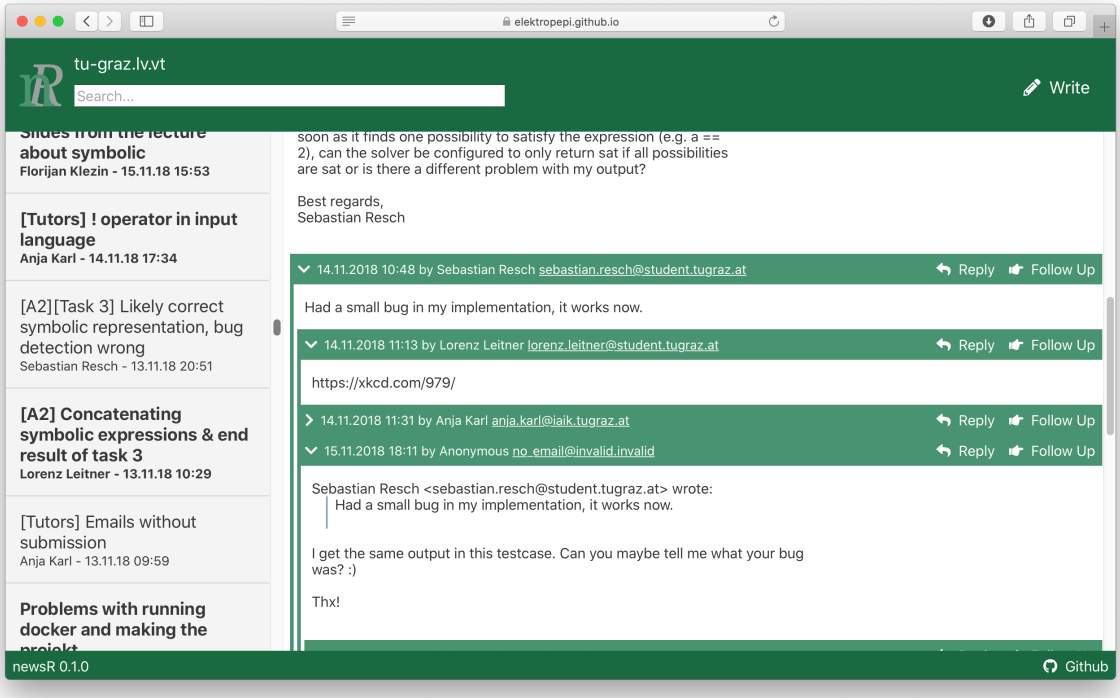

Figure 4.7: Thread Content View

Figure 4.8: Thread header, first on a desktop, then on a mobile device.

newsR on the other hand works more like a web user would expect from a website, loading content when browsing to it. When the user clicks on an group, the list of articles is loaded and structured as a hierarchy of threads and when a thread is opened the body content of each article in that thread is loaded. In addition both the list of articles and their content are stored in the browsers indexedDB as soon as they are loaded. This could be improved to provide full offline first capabilities.

4.5 Posting

Posting articles means letting the user write an article with a subject line and article body with multiple lines of content. The article must also include author information with a name and email address. Articles belong to groups and can contain references to other articles if they are a follow-up article.

newsR has two different entrances to the posting view, one for new threads and one for creating follow-up articles. In the thread overview, as can be seen in Figure 4.4, there are two links to post directly to the group, to start a new thread. In the mobile view only the write link in the header is shown. The link to

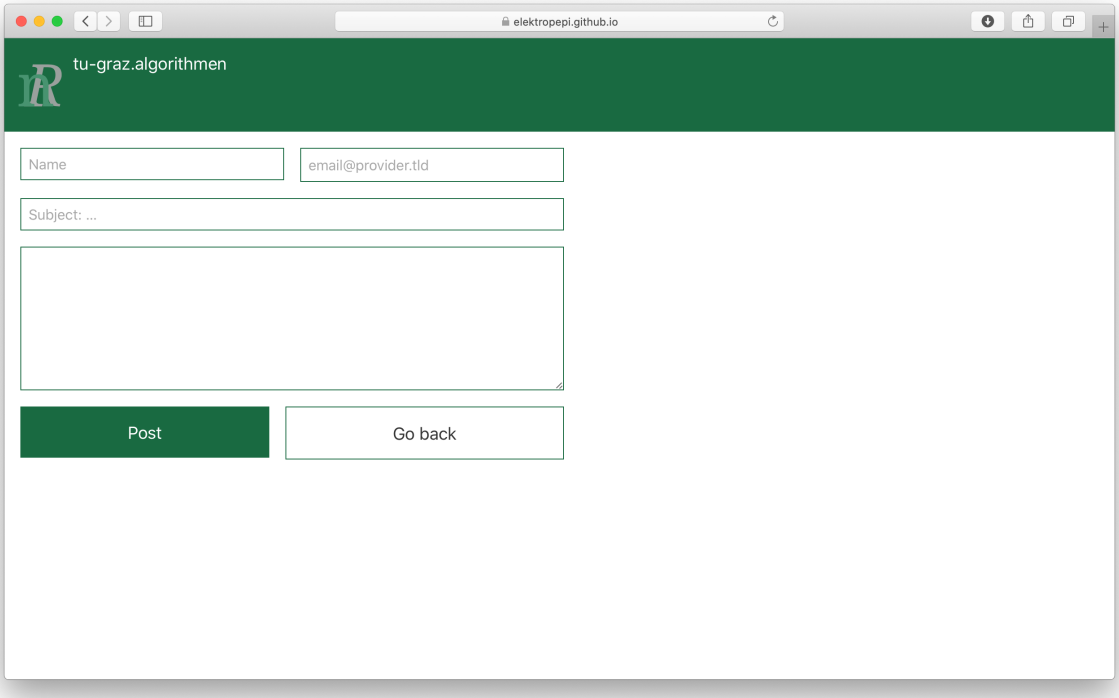

Figure 4.9: Post Article View

post as a follow-up is in the header of each article in the thread content view, as described in Section 4.3.

The post article view consist of a HTML form with input fields for author name and email, subject line and a text area for the article content, as can be seen in Figure 4.7. The user can decide to post the filled in information to the server or to go back to the previous view. The site header shows meta information with at least the group name that the article will be posted to. Author information only needs to be written for the first post a user submits, on subsequent posting the author fields will be automatically filled with the data from the last time. The user can still decide to use a different name or email with every new article.

When writing a follow-up article, the header contains more information on the article that is followed up on. The subject line is automatically filled with the subject line of the thread with up to one `"Re:"` prefixes. In the background the reference information is kept to be send to the server, for it to be inserted into a thread at the correct position.

4.6 React Router - Declarative Routes

With newsR, the current state of the application shall always be reflected by the URL. This means a URL will always be shareable and upon refreshing the page, the state must still be the same. This approach is also called deep linking W3C 2020. React router RT 2020b was used to achieve this functionality. It proved to provide an interesting approach to routing, called declarative routing. Instead of providing a centralized file with all routes, each component defines routing specific logic within its render function. This gives declarative a whole new meaning, because when one looks at the source code of a component, one can directly infer what is happening in regard to routing. If we refer to Listing 4.1, we can see the basic functionality of a `Switch` component provided by React Router. It contains multiple different `Route` components. Depending on the first match on the `path` property, react-router only renders the corresponding component. As we can see in the Listing, parameters in the URL are supplied with `:parameter` and optional parameters with `:parameter?`.

```
<div className="app">
  <Switch>
    <Route path="/groups/:name" component={GroupDetail}/>
    <Route path="/post/:name/:number?" component={Post}/>
    <Route path="/" component={StartPage}/>
  </Switch>
</div>
```

Listing 4.1: Basic route switch in root component

```
const {match} = this.props;
// (...)
<div className="app-grid-body">
  <SidebarContent
    sidebar={<List data={articleListData}/>}
    content={
      <Switch>
        <Route path={`${match.path}/:number`} render={props =>
          <ThreadDetail {...props} group={group}
            article={threads.find(thread => thread.number === parseInt(
                props.match.params.number))
            || null}/>
        }/>
        <NoThread url={match.path} groupName={group.name}/>
      </Switch>
    }/>
</div>
```

Listing 4.2: Further routing in child components

Further routing can be applied in child components. Minor parts of the GroupDetail component have been extracted in Listing 4.2. This component is used in the thread overview, as described in Section 4.2. In the first line, the important property `match` of the Route component is retrieved. `match.path` refers to the current match within the URL. For example, if "/groups/tu-graz.lv.iaweb/332" is currently opened, `match.path` would contain "/groups/tu-graz.lv.iaweb" since this was the URL part matched with "/groups/:name" in Listing 4.1. Using this value, one can further declare deeper routes within the GroupDetail component. In this view, during desktop mode, the list of threads of a group are always shown within a sidebar. The content on the right side of the sidebar now depends on further routes. By passing `path={‘$match.path/:number‘}` to the Route component, it is only rendered, if there is a number after the group name, like this: "/groups/tu-graz.lv.iaweb/332". If the number is missing, "/groups/tu-graz.lv.iaweb/", the `NoThread` component is rendered instead.

4.7 React Media - Declarative Responsiveness

Combining the declarative routing approach with React Media RT 2020a, which introduces CSS media queries to React, one has a powerful tool for responsive web development at hand. Both enable us to

```
export const SMALL_SCREEN_QUERY = "(max-width: 47rem)";
export const LARGE_SCREEN_QUERY = "(min-width: 47.1rem)";
// (...)
<Switch>
  <Route path={`${match.path}/:number`}>
    <Media queries={{small: SMALL_SCREEN_QUERY, large: LARGE_SCREEN_QUERY}}>
      {matches => (
        <Fragment>
          {matches.small && headerWithoutSearch}
          {matches.large && headerWithSearch}
        </Fragment>
      )}
    </Media>
  </Route>
  <Route path={`${match.path}`}>
    {headerWithSearch}
  </Route>
</Switch>
```

Listing 4.3: Displaying the header with or without a search bar depending on both screen width and URL.

render different components depending on both URL and screen size. On mobile for example, the URL "/groups/tu-graz.lv.iaweb" renders only the thread list, as can be seen in Figure 4.5. But on desktop, this URL must resolve to the `SidebarContent` component, having the `NoThread` component as content. Listing 4.3 shows another example how this combined approach can be used. The goal here is to hide the search bar in the header only for small screens viewing a thread body. With the `Media` component one can perform queries similar to CSS media queries. It then renders specific components only when a certain screen size is reached. Combining with the `Route` component, `Media` is only rendered for thread detail URLs. The result can be seen in Figure 4.5.

4.8 Progressive Web Apps capabilities

newsR already has minor PWA capabilities, such as A2HS and a service worker caching JS, CSS scripts in the background. Most of these capabilities came already integrated with the application bootstrapped by Create React App FB 2020a, which is described in more detail in Section 3.3. In Figure 4.10, one can see the install prompts offered by modern mobile browsers on Android. The first screenshot is of Firefox, where one has to open the settings pane and then an entry "Install" can be found. The second one is of Chrome, where directly upon opening the web application a prompt with "Add newsR to Home screen" is shown. Upon installation a start icon can be found on the launcher, as shown in Figure 4.11. The features offered by a A2HS enabled PWA can be seen in Figure 4.12. Upon launching the application from the start screen, it is launched in full screen mode. Depending on the settings in the `manifest.json`, even the top bar in Android is colored accordingly. Furthermore, does Androids task manager recognize it is a separate application, making split screen usage possible.

4.9 Attachments

Newsgroup articles can contain attachments like emails can, some servers or some newsgroups don't allow for attachments to be posted as they can be great in file size compared to text only articles. Attachments are contained within the body of the article, the whole body is encoded in the Multipurpose Internet

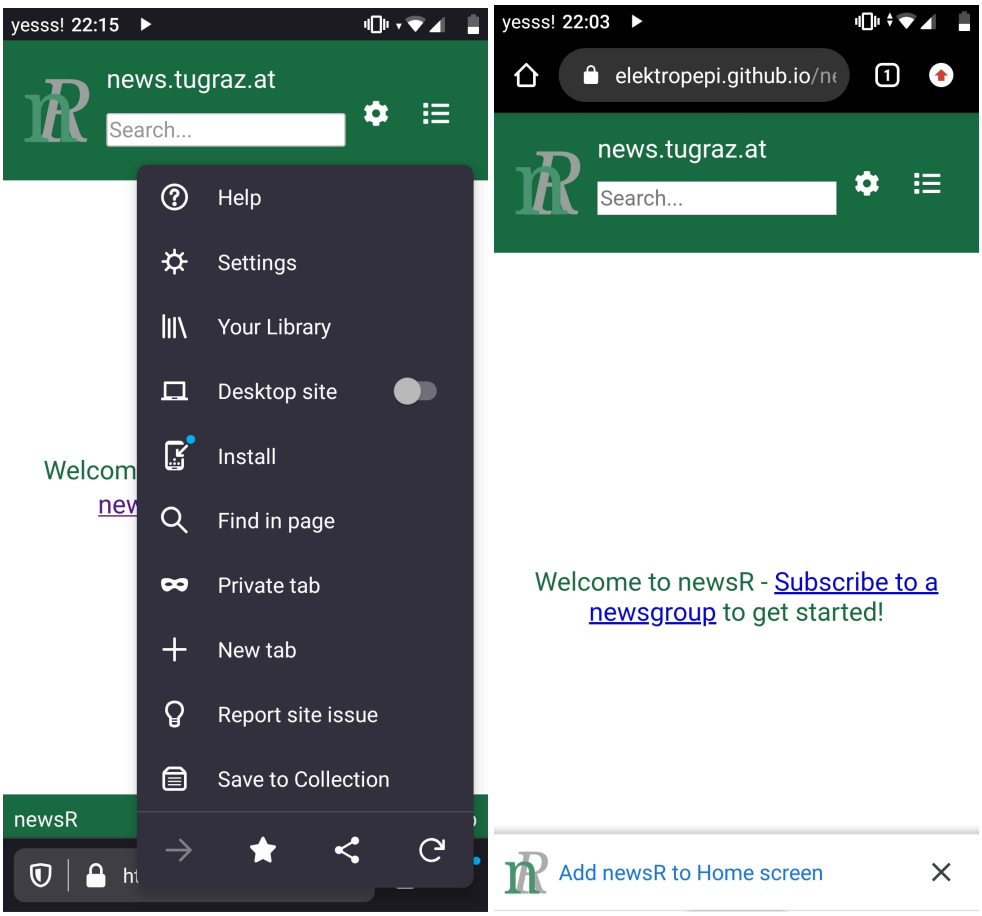

Figure 4.10: Install prompts for a PWA offered by mobile browsers, first being Firefox and second being Chrome.

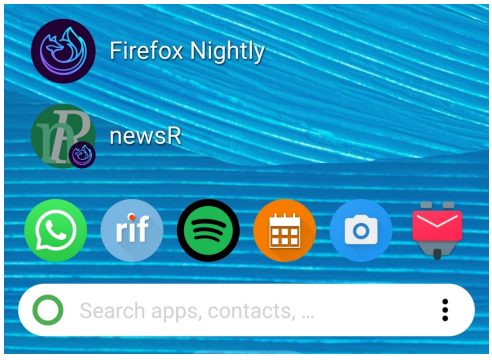

Figure 4.11: A2HS start icon added by Firefox. [Screenshot taken by the authors of this paper.]

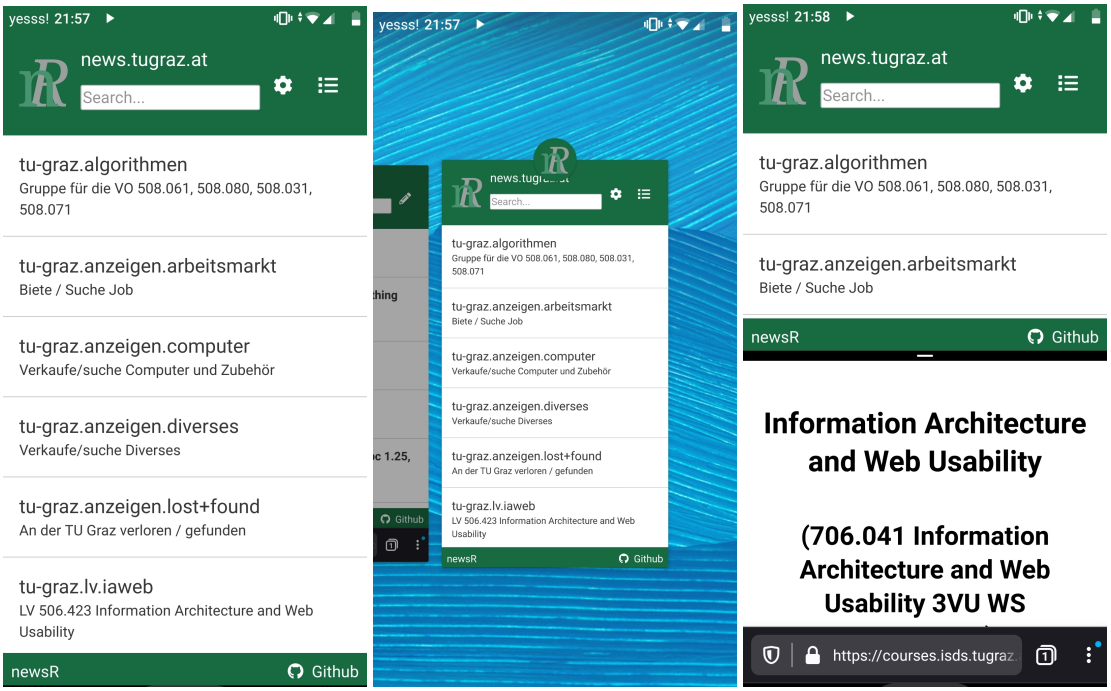

Figure 4.12: A2HS feature on Android in action: 1.) the web app is launched in full screen 2.) the app is shown as a separate application in the task manager 3.) even split screen is possible.

Mail Extensions (MIME) Freed and Borenstein 1996 format. The body is split in multiple parts, one of them contains the text content other parts contain attachments encoded in a base64 string. When leaving MIME unprocessed, the user is presented with a lot of meta information and text strings that mark each message part. The attachments themselves are unidentifiable and seemingly random strings to the user and also take up a lot of visual space.

newsR finds MIME multipart messages and parses them. The text part of the article is used as the regular article content as in articles without attachments. At the end of the content a list of attachments is shown with every attachment as a download link with the file name displayed. The links are Data URLs Masinter 1998 starting with `data://` containing the file type and the base64 encoded file within the link itself. Attachments with an image `Content-Type` like `image/png` or `image/jpeg` are directly shown as images, as can be seen in Figure 4.13. The image can also be downloaded by clicking it.

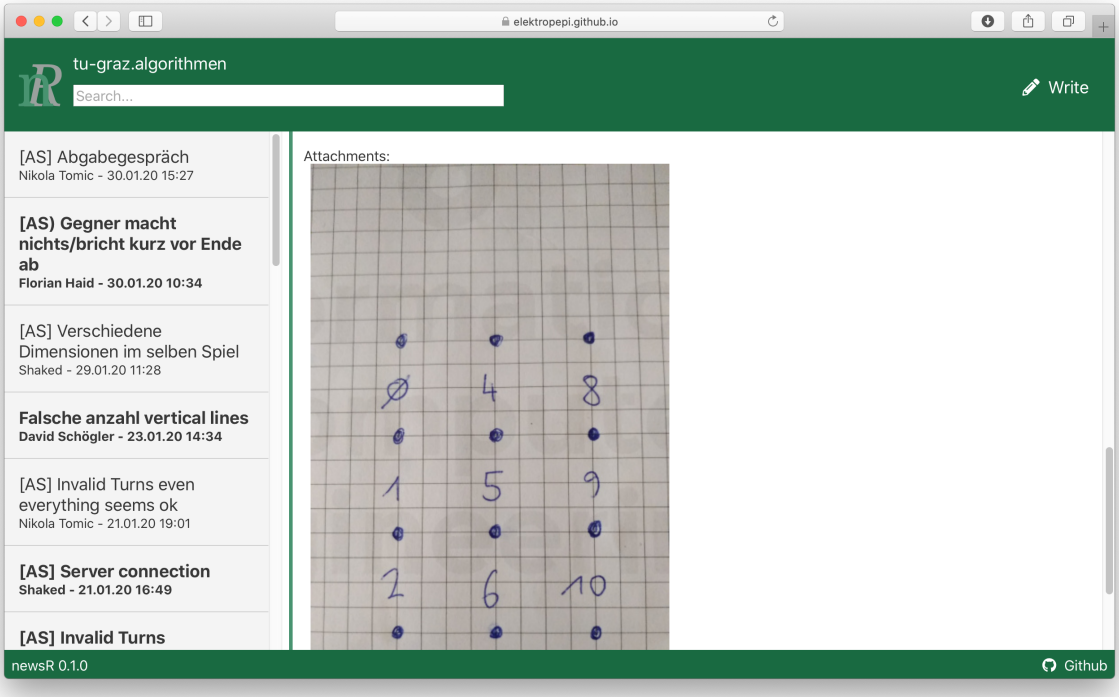

Figure 4.13: Image Attachment

Chapter 5

Conclusion

newsR is a full responsive web-app with basic features. Due to the fact that browsers cannot use raw TCP connection, an intermediate server is needed. This could be avoided by programming a native app. On the plus side, newsR is a website, which makes it compatible with every operating system. Because of the highly responsive and mobile optimized views, newsR can be used very well on every smartphone. With the implemented PWA features it can behave pretty much like a real app.

5.1 Future work

1. Posting like an instant messenger: reply at the bottom
2. Sorting by newsgroups
3. Fix charset troubles
4. Lazy loading of threads, newsgroups (for big numbers, e.g. VT has threads up until 2016, long loading times)
5. News server configuration

Bibliography

Barber, Stan O. [2000]. *Common NNTP Extensions*. RFC 2980. Oct 2000. doi:10.17487/RFC2980. https://rfc-editor.org/rfc/rfc2980.txt (cited on page 7).

ÉLIE, Julien [2010]. *Network News Transfer Protocol (NNTP) Additions to LIST Command*. RFC 6048. Nov 2010. doi:10.17487/RFC6048. https://rfc-editor.org/rfc/rfc6048.txt (cited on page 7).

FB [2020a]. *Create React App - Set up a modern web app by running one command*. Facebook. 25 Jan 2020. https://create-react-app.dev (cited on pages 9, 18).

FB [2020b]. *Jest is a delightful JavaScript Testing Framework with a focus on simplicity*. Facebook. 25 Jan 2020. https://jestjs.io (cited on page 9).

Feather, Clive [2006]. *Network News Transfer Protocol (NNTP)*. RFC 3977. Oct 2006. doi:10.17487/RFC3977. https://rfc-editor.org/rfc/rfc3977.txt (cited on page 7).

Freed, Ned and Dr. Nathaniel S. Borenstein [1996]. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045. Nov 1996. doi:10.17487/RFC2045. https://rfc-editor.org/rfc/rfc2045.txt (cited on page 20).

Masinter, Larry M [1998]. *The "data" URL scheme*. RFC 2397. Aug 1998. doi:10.17487/RFC2397. https://rfc-editor.org/rfc/rfc2397.txt (cited on page 20).

Murchison, Ken and Julien ÉLIE [2017]. *Network News Transfer Protocol (NNTP) Extension for Compression*. RFC 8054. Jan 2017. doi:10.17487/RFC8054. https://rfc-editor.org/rfc/rfc8054.txt (cited on page 7).

Murchison, Ken, Chris Newman, and Jeffrey M. Vinocur [2006]. *Using Transport Layer Security (TLS) with Network News Transfer Protocol (NNTP)*. RFC 4642. Oct 2006. doi:10.17487/RFC4642. https://rfc-editor.org/rfc/rfc4642.txt (cited on page 7).

MZ [2020]. *Progressive web apps*. Mozilla. 25 Jan 2020. https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (cited on page 9).

RFC1036 [1987]. *Standard for interchange of USENET messages*. RFC 1036. Dec 1987. doi:10.17487/RFC1036. https://rfc-editor.org/rfc/rfc1036.txt (cited on page 7).

RFC977 [1986]. *Network News Transfer Protocol*. RFC 977. Feb 1986. doi:10.17487/RFC0977. https://rfc-editor.org/rfc/rfc977.txt (cited on page 7).

RT [2020a]. *React Media: CSS media queries for React*. React Training. 25 Jan 2020. https://github.com/ReactTraining/react-media/ (cited on page 17).

RT [2020b]. *React Router: Declarative Routing for React.js*. React Training. 21 Jan 2020. https://github.com/ReactTraining/react-router/ (cited on page 16).

W3C [2020]. *Deep Linking in World Wide Web*. W3C. 21 Jan 2020. https://w3.org/2001/tag/doc/deeplinking.html (cited on page 16).

Wikipedia [2019]. *List of Usenet newsreaders*. 10 Nov 2019. https://en.wikipedia.org/wiki/List_of_Usenet_newsreaders (cited on page 4).