Hypermedia Applications
Technology report

POLITECNICO
MILANO 1863

Benelle Francesco	10727489@polimi.it
Cavicchioli Michele	10706553@polimi.it
Lo Presti Irene	10765687@polimi.it
Lodelli Riccardo	10790201@polimi.it

Our Website
GitHub Repository

17/07/2024

Table Of Contents

1 Description of the Project

The purpose of the project is to create a fully functional and accessible website for an imaginary anti-violence centre for women, that proposes some Activities, divided into Projects and Services. The name of the centre is **HERmet** as we aimed to recall the concept of protection (helmet) emphasizing our focus on our target audience: women.

The objectives of the proposed activities are broad: from basic services to support survivors of any kind of violence against women to activities that aim to help women with everyday problems, and from specific projects for particularly difficult situations to projects that promote awareness about gender disparities. We have fully developed the content for each activity, including names, descriptions, and service logos. Each activity is managed by an employee of HERmet.

We also managed the personnel of the center. Our idea was to create an environment where every woman, regardless of her particular story, could feel safe. To achieve this, we decided to create an all-female team. The women we chose are mainly fictional characters from films and TV series. We developed their role inside the center, personal descriptions and curricula vitae.

2 How we split the work

In this section we'll briefly explain how we divided the workload to complete this project. It's necessary to say that each and every member of the group worked on almost every part of the project, as we wanted to get some experience in everything. The workload of the project can be divided in at least 2 phases, the first one had the main goal to organise the design of the website, creating the C-IDM, the content tables, the abstract pages, the low fidelity wireframes and the ER diagram of the database. Whereas the second step was the actual development of the website.

When starting the first phase of the project we all agreed on the fact that it was better to do this *together* and then divide the coding in individual jobs, and that's exactly what we did. In fact, once the *design* phase was completed we started to split evenly the components/pages between the members of the group.

Member	Work area
Benelle Francesco	UX design, Content creation and insertion in DB, Website implementation, Pages' styling, Extra functionalities (Text To Speech, Speech To Text, Volunteer Form), reports.
Cavicchioli Michele	UX design, Content creation and insertion in DB, Website implementation, Pages' styling, Extra functionalities (Breadcrumbs, People Sorting), Report
Lo Presti Irene	UX design, Content creation and insertion in DB, Images searching and logos, Website implementation and Pages' styling of Home Page, Single Service Page, and Single Project Page, only styling of single Person Page, chatbot's basic funcionalities and style, user testing, reports.
Lodelli Riccardo	UX design, Content creation and insertion in DB, Website implementation, Default page and About Us styling, SEO implementation, reports.

3 Hosting Service

During classes, two possible hosting services were suggested to us: GitHub Pages and Vercel+Supabase.

We chose the solution of **Vercel** and **Supabase** because we wanted to develop a dynamic website, and GitHub Pages is a static site generator.

We connected Vercel with our GitHub repository to automatically update the website after every pushed commit, and we used a *Postgres* database offered by Supabase. Regarding Supabase, each person in the group created a local *.env* file (which was put inside the *.gitignore* file) where they stored the proper credentials (*url*, *password*) used to access the online database.

4 Structure of the project

The purpose of this section is to describe in detail how we organized the project directory, explaining where each component is located. Here's a condensed version of the folder hierarchy of the project (for the complete hierarchy, please refer to the GitHub repository):

```
Website
├── assets
│   ├── css
│   ├── imgs
│   └── svgs
├── components
├── layouts
├── pages
│   ├── about_us
│   ├── activities
│   ├── contact_us
│   ├── our_women
│   ├── projects
│   ├── services
│   └── volunteer
├── public
└── server
    └── api
        ├── activities
        │   ├── leader
        │   ├── mostRecentActivity
        │   ├── projects
        │   └── services
        ├── chatbot
        ├── our_women
        └── testimonials
```

./assets

The *assets* folder contains files such as images and svgs, it also contains a css folder used to store the *main.css* file for the setup of *tailwind*.

./components

The *components* folder stores the *.vue* files that implement the components we need throughout the entire website. Down here you can find the list of components we implemented and a brief description of their purpose and how we used them (for each component the extension of the file, *.vue*, is omitted as it's the same for every item):

- **Chatbot**: displays the front-end of the chat window and handles the calls to the *OpenAI* API to get the answer from the chatbot (see the chatbot section in the *Design Report*).
- **Navbar**: simply displays the navigation bar.
- **Person**: groups together the image of a person, its name, surname and role, into a single container used in the */our_women/* page. The outer container is also a link to redirect the user to the page of the relative person.
- **Activity**: similar to the *Person* component but used to handle the activity of the center. This component was not merged together with the *Person* component due to differences in the code structures required to implement them, even though logically they are quite similar. Furthermore, this handles both projects and services.
- **Section**: used in the */our_women/* page to display the various roles of the personnel. There is no JavaScript code inside the corresponding file, as all interactions related to these sections, specifically the click events, needed to be managed directly within the */our_women/* page.
- **FormVolunteering**: this component displays the volunteering form and handles the parameters check on client-side.

./layouts

This folder contains only one file, called *default.js*. Its purpose is to redefine the default layout of Nuxt.js. In this case, we modified it to include the navbar, the footer, the chatbot button, and, depending on the screen size, the social buttons.

./pages

For each directory inside this folder, a new route is built, where the *URL* is defined as the relative path of each page (starting from *./pages*). Here's the complete list:

- **/index.vue**: Homepage
- **/activities/index.vue** displays two carousels, one for the projects and one for the services, to show all the activities of the centre.
- **/projects**
  - **/index.vue**: page containing the "cards" to display all the projects proposed by the centre (components: *Project*);

  - **/[id].vue**: page of an individual project, the parameter *id* is used to retrieve the proper information about the requested project.

- **/services**
  - **/index.vue** displays the ”cards” to show the services offered by the centre (components: *Service*);
  - **/[id].vue**: page of an individual service, the parameter *id* is used to retrieve the information about the requested service.
- **/our_women**
  - **/index.vue**: group page to show the employees of the center. They can be seen grouped by role or all together with the possibility of being alphabetically ordered (components: *Person*, *Section*);
  - **/[id].vue** displays the information of a specific person (identified by the *id* parameter). If the person is in charge of an activity there's also a link to its page.
- **/volunteer/index.vue**: page used to gather volunteering requests from the users. It contains a brief description of what *volunteering* means for the centre, how you could help, and a form to send the request.
- **/about_us/index.vue**: *About us* page containing some specifics about the centre, its purpose and a couple of the proposed activities.
- **/contact_us/index.vue**: usual *Contact Us* page with the coordinates to contact anyone in the centre, its geographical location, socials and so on.
- **FormVolunteering**

./stores

Together with Nuxtjs we had to use the ***pinia*** module to handle some states. More specifically, we had to use *pinia* to share the employees' list and the chosen section between the */our_women/* page and the individual page of an employee (this data is used in the breadcrumbs of the *./our_women/[id].vue* page).

./server/api

The purpose of this folder is to contain the APIs we developed to get all the information we needed throughout the entire website. As we did for the components and the pages, here's a list of all the APIs (in some files is specified the HTTP method but not in others because if not specified in the file's name the default method used is GET):

- **/activities**
  - **GET /leader/[leaderId].js**: retrieve the activity lead by a certain person defined by its

person_id;

- **GET /mostRecentActivity/index.get.js**: get the latest started activity;
- **GET /projects/index.js**: retrieve all the projects handled by the centre;
- **GET /services/index.js**: retrieve all the services offered by the centre;
- **GET /[activityId].js**: get the information of a single specific activity (defined by the *activityId*);
- **GET /index.js**: fetch **all** the activities of the centre.

- **POST /chatbot/index.post.js**: this endpoint handles the conversation with the chatbot, implemented by the *gpt-3.5-turbo* model of *OpenAI*. We send the entire prompt of messages (the one to fine-tune the model) at each request, the body contains a list of messages starting with existing prompts, and each new message is appended to this body. This approach allows the model to process the entire chat history with the user during each interaction. Ultimately, this API sends a POST request to the *OpenAI* API, awaiting the model's response. The object returned by this API endpoint is the answer generated by the ChatGPT model.
- **/our_women**
  - **GET /[id].js** fetches the information of a certain person given it's *person_id*;
  - **GET /index.get.js** retrieves all the employees of the centre.
- **GET /testimonials/[activityId].js** gets the testimonials of a certain activity based on its *activity_id*.

Extra Modules

Here we explain what extra modules we had to use to implement the website.

- **Supabase**: the supabase module is used to retrieve the *createClient* function to get a supabase client through which we query the supabase database. The credentials to authenticate the client are retrieved from a *.env* file [@nuxtjs/supabase]
- **Google Fonts**: this module has been installed with the purpose of importing new fonts coming from *Google Fonts* [@nuxtjs/google-fonts]
- **Tailwind CSS**: module to import the css properties of *tailwindcss* [@nuxtjs/tailwindcss]
- **Pinia**: used to import the *Pinia store manager* [@pinia/nuxt]

Even though it technically isn't a new module, it's worth specifying that we also imported *Flowbite*, an open source collection of UI components built with the utility classes from *Tailwind CSS*. The purpose of flowbite in our project was to implement the carousels used in the following pages:

- /actitivities
- /projects
- /services
- /about_us

The flowbite components that involve already-written javascript code (like dropdowns, carousels and so on) must be initialized on the mount of the page; this is to check that the only components we initialized are the carousels.

5 Responsiveness

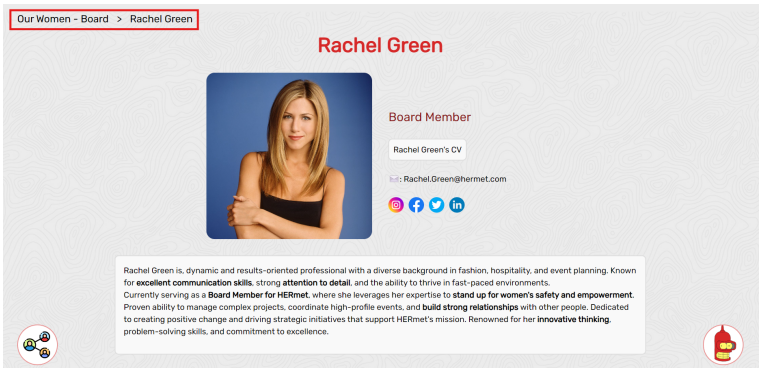
We designed our website in such a way to maximize its usability and we made it responsive and adaptable to every possible device (either computers, tablets and smartphones) with the highest possible variety of screen or window size. In order to accomplish this goal we used CSS features as the viewport units (as *vh* or *vw*) that help adjusting dimensions with respect to the window's height and width, we exploited *rem* and percentages to make the elements scalable with respect to the *father* element in order to adjust spacing and dimensions (in a lot of instances, instead of manually write media queries we used the utility classes of *Tailwind CSS*, like *sm:*, which enables a class only if the screen size is at least *small*). Here's an example of a media query we implemented:

```
@media (max-width: 768px) { // 768px = md (medium)
  .person-card {
    flex-direction: column;
    padding: 1rem;
    gap: 1rem;
  }
}
```

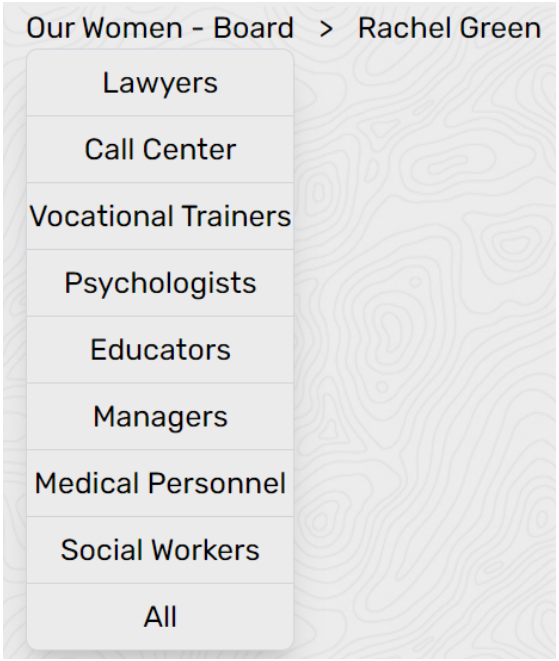
6 Extra Functionalities

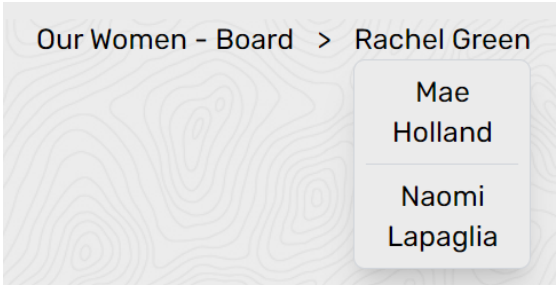
6.1 Bread Crumbs

We decided to have bread crumbs in the pages relative to people and activities. This feature enhances the ease of navigation of the website. They are made as interactive labels that show the hierarchy of the website and also act as links to redirect to pages of the same group. They're shown in the upper-leftmost part of the page:



When the user hovers the cursor on them, a dropdown menu is opened, showing all the links to the related pages of the same level of the one which the bread crumb's label is about.





6.2 People Sorting

Inside the ***Our Women*** page you can find a list of all the employees of the centre. As you can see from the below screenshots you can filter the employees based on their role (*Lawyers*, *Vocational Trainers*, ...), with the *Board* section being the default one, or list all the people at once. Regarding this last option, you can sort the people based on their surname, either in alphabetic order or reverse alphabetic (once you select a sorting method the relative button will highlight that it's the active method).

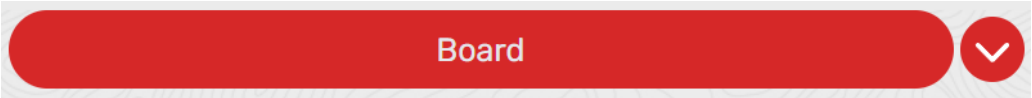


Figure 1: Default appearance

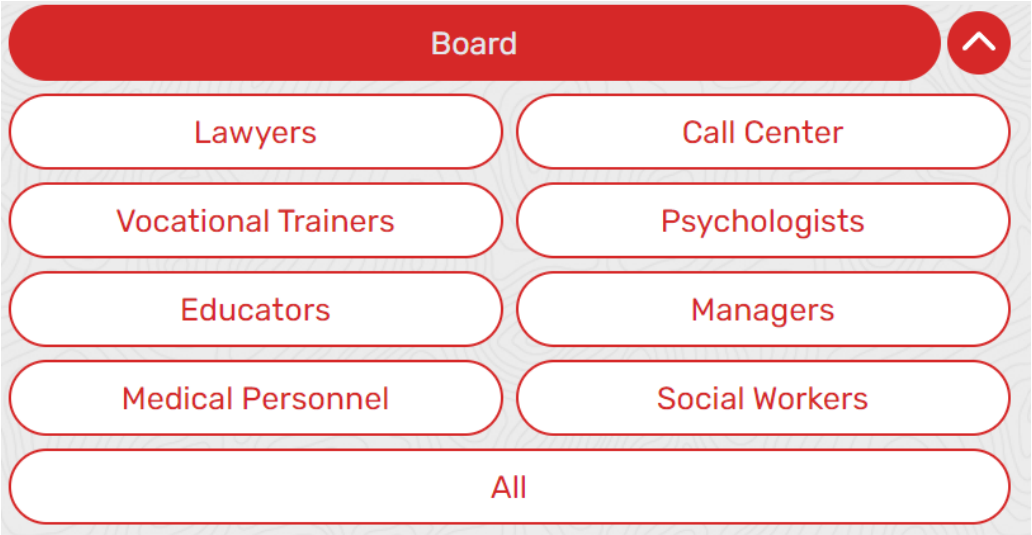


Figure 2: Section selection

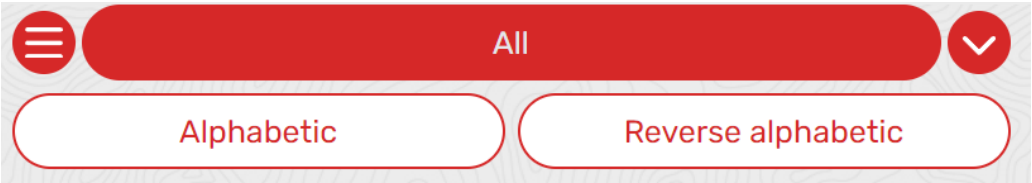


Figure 3: Sorting selection

6.3 Speech To Text and Text To Speech

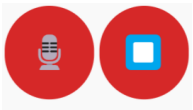
As described in the chatbot's report, STT and TTS functionalities were added to the chatbot, aiming to enhance the accessibility and to give the possibility to write a message without the keyboard and hearing the text message that comes out as response.

6.3.1 Text-to-Speech

The Text-to-Speech (TTS) feature allows the chatbot to convert its text responses into spoken words. This functionality has been implemented mostly in order to provide an audible form of communications for those users who may have difficulty reading text on a screen.
The feature has been implemented using the voice of an English-speaking woman, in order to try to make as most at ease as possible the designed target of the chatbot. It has also been implemented a filter that parses the text messages coming out from the chatbot in order to avoid the reading of html tags or similar that would be inevitably present due to the choice of making Jarvis provide html links to the pages of the projects and services present in the website.
All the final user will have to do in order to activate the Text-to-Speech function will be clicking on the icon (provided below) that will show up at the end of the the incoming chatbot's messages and consent the use of the external audio devices to the browser.

6.3.2 Speech-to-Text

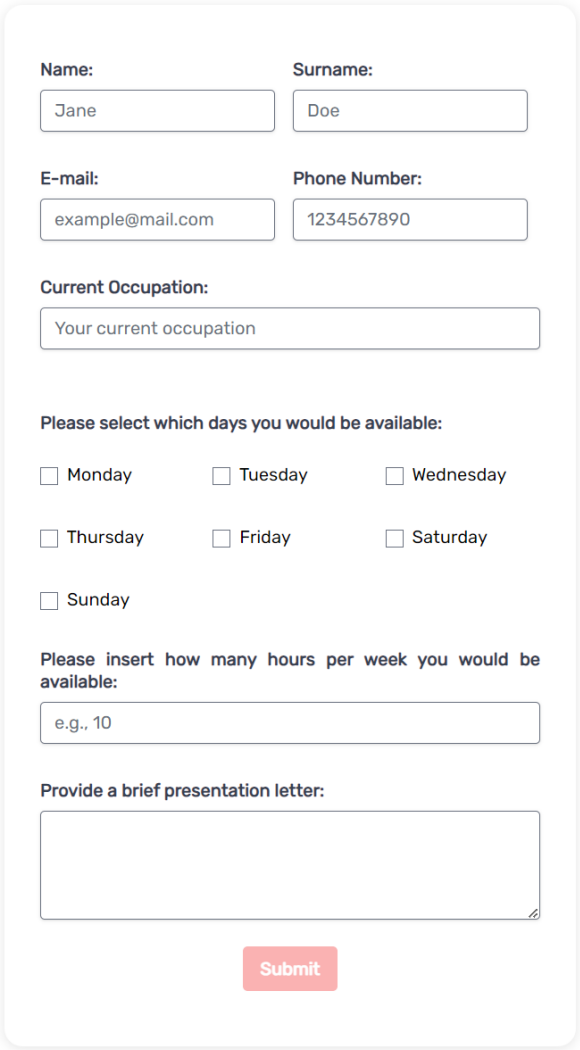
The Speech-to-Text (STT) feature enables users to input their queries and commands to the chatbot using their voice. This functionality has been implemented mostly in order to provide an alternative form of communications for those users who may have difficulty typing on the keyboard their messages or just feel more at ease expressing their requests vocally.
At the condition of the user granting the browser access to their microphone, they will be able to start the vocal recognition by clicking on the apposite *start* button (on the left) and by clicking on the *stop* button (on the right) once they finish.



The voice recognition, whose default language is set to English, will capture the users' voice messages and parse them as texts, before feeding them to the chatbot as if they were common written messages.

6.4 Volunteer Form

We decided to add a page that provides information on how to volunteer with HERmet. It contains an introduction encouraging community involvement, followed by information on volunteering opportunities. Furthermore, there is a **form** for whoever is interested to provide their personal information and contacts. The content of the form is posted in an ulterior database table, which is exclusive for this functionality and just stores the content of the form submission. Upon submission of the form, a popup will show up to show the user the status of the process. The submission button will be rendered impossible to be clicked until all the fields are completed and the checks on the content of the fields are passed.



7 SEO

Search Engine Optimization (SEO) is the practice of enhancing a website to **improve its visibility and ranking on search engine results pages**. The primary goal of SEO is to attract more organic traffic to a website by ensuring that it appears prominently in search engine results for relevant queries.
SEO is a fundamental process in website development because it allows a website to **rank higher** when users search for similar topics, and **being at the top means being more frequently visited**.
To achieve a high ”score” from crawlers—automated programs used by search engines to scan and index web pages—the **<head>tag** of each webpage should include specific **metadata** that helps crawlers analyze and understand the page.
In particular, there are three main elements that must be inserted into the <head>tag:

- **Title**: describes the page in a few words
- **Description**: complements the title by providing more details about the page
- **Keywords**: a list of words that should appear when searching for the page with search engines

Designing the website with Nuxt generates a few challenges, such as the use of Single File Components (SFC) instead of plain HTML (there is no ¡head¿ tag) and the fact that a Nuxt website is a Single Page Application, making it seem difficult to distinguish different pages. Luckily, Nuxt provides tools to achieve SEO, mainly through two different methods:

7.1 SEO - Global

By adding tags in the **nuxt.config.ts** file, a default bunch of metadata is created inside the <head>tag of each website page. Here is our implementation:

```
app: {
    head: {
        title: 'HERmet',
        htmlAttrs: {
            lang: "en",
        },
```

```
        meta: [
            { charset: 'utf-8' },
            { name: 'description', content: 'HERmet website, where you can
            find all the information you need about us: people, projects,
            services, contacts.' },
            { name: 'keywords', content: 'HERmet, Antiviolence Center,
            People, Projects, Services, Contacts' },
            { name: 'viewport', content: 'width=device-width,
            initial-scale=1' },
            //social media meta tags
            { name: 'og:title', content: 'HERmet' },
            { name: 'og:description', content: 'HERmet website, where you can
            find all the information you need about us: people, projects,
            services, contacts.' },
            { name: 'og:url', content: 'https://hypermediaapplication
            project2024-benels-projects.vercel.app/' },
            { name: 'og:image', content: 'https://imgur.com/a/3XjyyVq' },
            { name: 'author', content: 'TheCEOs' },
        ],
    }
}
```

In this script, there are the three tags previously mentioned and some other important tags, such as default language, charset, and viewport to ensure correct device adaptability, along with a series of "og:xxxxx" tags that are meaningful when the website is shared among Social Networks.

7.2 SEO - Local

In addition to the general global SEO approach - which sets the same tags for ALL the pages of the website - Nuxt offers the opportunity to **dynamically** set some SEO tags for specific pages. This way, if a user is interested in a specific page of the website, search engines will propose that **specific page** instead of the homepage, which can lead to confusion for some users. To achieve local SEO, we inserted the following script under the <script> tag:

```
//Search Engine Optimization
const description = ref('Specific page description')
const keywords = ref('Specific page keywords')

useHead({
  meta: [
    { name: 'description', content: description },
    { name: 'keywords', content: keywords }
```

```
    ]
})
```

The script above declares two const variables for the current page, containing the specific description and keywords. Then, it uses these variables to replace the default description and keywords tags from the global approach, providing a dynamic SEO solution.