POLITECNICO
MILANO 1863

Technology Report

Hypermedia Applications

A.Y. 2024-2025

Team

Matteo Civitillo	matteo.civitillo@mail.polimi.it
Mattia Vicenzotto	mattia.vicenzotto@mail.polimi.it
Nicolò Gandini	nicolo.gandini@mail.polimi.it
Xin Ye	xin.ye@mail.polimi.it

Professor: Franca Garzotto
Assistant Professors: Giovanni Caleffi and Matteo Secco

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Version: 1.0
Date: 15-July-2025

Contents

1 Abstract

The aim of this technical document is to present the key technological choices made during the development of the Serendipity Yoga website. It describes the site structure and the custom components developed, and outlines the tools, modules, and libraries used. Finally, it highlights how accessibility and SEO guidelines were addressed in the design and implementation.

2 Introduction

2.1 Group information

Group name: Bada Bastu

Team members:

Matteo Civitillo	matteo.civitillo@mail.polimi.it
Mattia Vicenzotto	mattia.vicenzotto@mail.polimi.it
Nicolò Gandini	nicolo.gandini@mail.polimi.it
Xin Ye	xin.ye@mail.polimi.it

2.2 Project information

- Link to our website:
  https://hypermedia-applications-rho.vercel.app/
- Link to our GitHub repository:
  https://github.com/yexin01/hypermedia_applications

2.3 Work Breakdown

Due to the high number and complexity of tasks involved, we successfully managed to distribute the workload evenly. This allowed each team member to actively contribute to all areas of the project, including implementation, database modeling, technical documentation, and the final report. This balanced approach ensured full engagement and comprehensive learning across all dimensions of the development process.

3 Documentation

3.1 Chosen Theme

We have decided to create a website for a yoga center called **"Serendipity Yoga"**, based in Milan, Italy. The yoga center follows a holistic approach that blends yoga, meditation, and activities to guide body, mind, and spirit.

3.2 Technological Choices

- **Frontend framework: Nuxt 3.**
  According to project requirements, the website uses Nuxt 3 (Vue 3-based) to conform to the specified framework and leverage its hybrid rendering, file-based routing, and modular component conventions.

- **Application hosting: Vercel (frontend) and Render API (backend).** The Nuxt app is deployed on Vercel to benefit from automatic builds on Git pushes and global edge caching. Backend logic is handled via Python scripts deployed through Render API, allowing for a serverless-like experience without maintaining dedicated infrastructure.

- **Database hosting: Supabase.** Supabase's managed PostgreSQL instance is used to support relational data (activities, teachers, rooms, etc.) and provides built-in REST and Realtime APIs. These are consumed directly by the Python backend deployed on Render API.

- **Styling: Tailwind CSS.**
  Tailwind was integrated via the `@nuxtjs/tailwindcss` module to facilitate rapid, utility-first styling and consistent responsive layouts without custom stylesheets, streamlining the design workflow.

- **Rendering mode: SSR.**
  Server-side rendering was adopted so that essential content is delivered fully formed to users and search engines, improving SEO and reducing "time to first meaningful paint." CSR was not used since the site's informational nature does not require heavy interactivity.

- **Routing: File-based routing (Nuxt/Vue Router).**
  Each `.vue` file in `pages/` generates a corresponding route (e.g., `pages/activities/[id].vue` $\Rightarrow$ `/activities/:id`), providing predictable URL structures without manual configuration.

- **Scripting language: TypeScript (frontend) and Python (backend).** Vue components and composables are written in TypeScript for type safety and improved developer experience. The backend is implemented in Python 3 and deployed via Render API, which exposes RESTful endpoints to the frontend.

- **Backend deployment: Render API.** The backend is deployed using Render API, which handles infrastructure, routing, and scaling. Python code defines the RESTful logic, and Render serves the application directly through HTTP endpoints without relying on a dedicated web framework.

3.3 Project structure

3.3.1 Links and page structure

Page	URL	Description
Home	/	It is the Serendipity Yoga home page. It summarizes content from other pages and provides various links to them and general information.
The center	/center	It provides detail about the center, the activities and the available rooms and areas.
Activities	/activities	It provides an overview of all the available activities in the center.
Activity	/activities/[name]	It describes the activity in detail providing the following information: level, teachers, schedule, benefits, the room where it will take place and other related activities.
Teachers	/teachers	It displays all the teachers.
Teacher	/teachers/[name]	It provides in detail a description of the teacher with the main expertise areas and their role.
Prices	/prices	It displays the membership plans.
Contact	/contact	It displays all the contact information like email, telephone number and the map to show where the center is located. A FAQ section is also present and a contact form.

3.3.2 Available server endpoints

API call	Parameters	Description
/teachers	language	Get all teachers information
/teacher/{teacher_id}	teacherId, language	Get information from a specific teacher
/teacher/activity/ {activity_id}	language	Get teacher name and surname by activity ID
/teacher/{teacher_id}/ activities	teacherId	Get all activities for a specific teacher
/activities	language	Get all activities information
/activity/{activity_id}	activityId, language	Get a specific activity by ID
/activity_by_name	activityName	Get a specific activity by name
/activity_id_from_name	activityName	Get activity ID from name
/rooms	language	Get all rooms information
/room/{room_id}	roomId, language	Get a specific room by ID
/areas		Get all areas information

/area/{area_id}	areaId	Get a specific area by ID
/reviews		Get all the reviews

3.4 Custom types

We have defined and used several custom data structures throughout the codebase to facilitate the developer experience and maintain clarity in data handling. These can be distinguished into two categories:

Data/Domain Types

These include: `Teacher`, `Activity`, `Room`, `Breadcrumb`, and `Language`. They represent core entities used across pages and components and are tied to data fetched via the API.

- `Teacher`: /pages/teachers.vue, /services/api.js
- `Activity`: /pages/activities.vue, /services/api.js, /components/misc/ActivityCardSuggestion.vue
- `Room`: /components/misc/RoomCard.vue
- `Breadcrumb`: /components/home/BreadCrumbs.vue
- `Language`: /components/home/LanguageSelector.vue

Utility Types

This category includes `ContactFormData` and several props objects used across components (e.g., `activity`, `room`, `breadcrumbs`, `languages`) for managing UI state and structured data passing.

- `ContactFormData`: /components/misc/ContactForm.vue

3.5 Custom component

Most of our website is built from components that we have designed and developed according to two guiding principles:

- **Reusability**: Many elements, such as buttons, cards, and carousels, appear throughout the site. To maintain consistency, we encapsulated these into standalone Vue components. When minor variations are needed, we rely on optional props or boolean–flag props instead of duplicating code.
- **Modularity**: Breaking the interface into discrete components keeps each page's code clean and easy to navigate. Even components used only once were separated if they simplified the parent page's logic or enhanced readability.

Below, we describe our custom components in detail.

3.5.1 Buttons

Our application includes custom-built buttons that go beyond simple interactivity, offering dynamic behavior, accessibility considerations, and support for localization and theming.

- **LanguageSelector Dropdown**: A language selector dropdown that displays the current language with its flag and allows switching to other available languages. The component supports keyboard accessibility and dynamic labels based on the current locale.

  File: `components/home/LanguageSelector.vue`

  Data Source: props (languages, modelValue) + internal translations object

  Controls: manual (click to toggle dropdown and select language)

- **ThemeToggle**: A circular button that toggles between light and dark themes. Displays a sun icon in light mode and a moon icon in dark mode. Includes transition effects for smooth icon switching.

  File: `components/home/ThemeToggle.vue`

  Data Source: reactive state from composable `useTheme()`

  Controls: manual toggle (click event)

3.5.2 Cards

Reusable card components that visually and interactively present structured information. Each card is designed with accessibility, responsiveness, and theming in mind. Cards consume props-driven data, support localization when needed, and often serve as entry points to more detailed content via navigation.

- **ActivityCard**: A prominent card used in activity listings. It displays the activity's image, title (or name), level, and a short description. The card features staggered entrance animations based on index, smooth hover effects (e.g., shadow, scale, opacity), and a graceful image-loading fallback with a loading label. Clicking the card navigates to the activity's detail page.

  File: `components/misc/ActivityCard.vue`

  Data Source: passed via props (activity object with title or name, image, level, short_description; index for staggered animation delay; t() for translations)

  Controls: manual (click to navigate, hover effects)

- **TeacherCard** A visually rich card representing a teacher. It displays the teacher's image, full name (from name and surname), role, and up to two areas of expertise (split by). It includes staggered entrance animations based on index, graceful image loading with error fallback (including an SVG-based placeholder), and interactive hover effects like scale-up, elevation, and text movement. Clicking the card navigates to the teacher's detail page.

  File: `components/misc/TeacherCard.vue`

Data Source: passed via props (teacher object with name, surname, image, role, main_expertise; index for animation delay; t() for localization)

Controls: manual (click to navigate, hover for visual feedback)

- **ActivityCardSuggestion** A compact card that showcases an activity, including its image, title (or name), level, schedule, and a short description. It supports smooth hover effects (e.g., scale, shadow, label transitions) and navigates to the activity's detail page when clicked.

  File: `components/misc/ActivityCardSuggestion.vue`

  Data Source: passed via props (activity object with id, title/name, image, level, schedule, short_description)

  Controls: manual (click to navigate, hover for visual effects)

- **RoomCard** A content-rich card that displays details about a yoga center room. It includes the room's image, name (or title), description, optional features, and related activities (with navigation links). It is purely visual and does not support hover animations.

  File: `components/misc/RoomCard.vue`

  Data Source: passed via props (room object with name or title, image, description, optional features array, optional activities array of activity objects, optional legacy_activities array of strings, optional quote)

  Controls: none (purely display, no user interaction except navigation links for activities)

3.5.3 Carousels

The following carousels have been custom-built using Vue and Tailwind CSS (no external carousel library). For accessibility, all are manually controlled (no auto-scroll by default, except HighlightedActivities which supports optional auto-slide).

- **HighlightedActivities Carousel** A carousel that displays highlighted activities (e.g., Ashtanga Yoga, Gentle Water Yoga, Sunset Yoga) in sliding cards. Users navigate via arrows or dots; auto-slide every 5s can be toggled.

  File: `components/home/HighlightedActivities.vue`

  Data Source: fetched filtered internally from API

  Controls: manual (arrows dots) + optional auto-slide

- **Room Carousel** A carousel that displays each room of the yoga center as a card. Navigation via arrows or dots.

  File: `pages/center.vue`

  Data Source: passed in via props (fetched from API)

  Controls: manual (arrows dots)

3.5.4 Containers

Rather than standalone container components, this project groups related content using Section Components. Each section bundles its own markup and sub-components for clarity and modularity:

- **HeroSection**: Groups the main hero content for the homepage.
  File: `components/home/HeroSection.vue`
- **ContactSection** Groups contact information and the contact form.
  File: `components/home/ContactSection.vue`
- **YogaClasses**: Groups the yoga class cards and related information.
  File: `components/home/YogaClasses.vue`
- **HighlightedActivities**: Groups the highlighted activities carousel and navigation.
  File: `components/home/HighlightedActivities.vue`

3.5.5 Other Components

We will now provide a brief description of the remaining custom components that do not fit into any previously mentioned category.

Footer

The footer, located at the bottom of each page, shows the most useful information:

- Logo
- Quick links: displays about us, highlights, activities and teacher links.
- Contact us: useful information about contact info and location
- Follow us: social media links

Navigation bar

It includes all the landmarks to navigate through the site, plus light/dark mode button and language switch button.

Loader

This component serves as a loading bar, temporarily replacing database data whenever it is being loaded, indicating that the information is being fetched.

SocialWall

The SocialWall component displays a live feed of posts and updates from the platform's official social media accounts.It is designed to foster engagement and keep users informed about the latest news, events, and announcements. Posts are dynamically fetched and presented in a scrollable layout, enriched with media content (images or videos) and interaction buttons such as likes or shares, where applicable.

3.6 Extra modules

Module / Service Name	Usage Description
Tailwind CSS	Provides the core Tailwind CSS framework
Autoprefixer	Adds vendor prefixes to compiled CSS for cross-browser compatibility
Postcss	Runs CSS through plugins (e.g., Tailwind, Autoprefixer)
Vue-router	Manages client-side routing in the Vue / Nuxt application
Supabase	Serves as a Backend-as-a-Service for database
Vercel	Hosts and deploys the frontend (Nuxt) application
Render	Hosts and deploys the backend API (FastAPI)

3.7 External libraries

As per project rules, we used only the **Tailwind CSS** framework to handle all styling. Tailwind was imported via the `@nuxtjs/tailwindcss` Nuxt module to provide utility-first classes for layouts, spacing, and responsive design. No other external CSS or JavaScript libraries were added. To customize default Tailwind values (colors, breakpoints, etc.), we included a `tailwind.config.js` file at the root of the project.

4 Extra

This chapter will cover the website's adherence to accessibility and SEO guidelines, with compliance evaluated using the WAVE and Lighthouse web evaluation tools.

4.1 Accessibility Compliance

- High contrast: tools like Deque University Color Contrast Analyzer are used to balance the contrast ratio.
- Clear structure: the website uses spatial allocation and headings to clearly differentiate between various pieces of content.
- Coherence: similar pages are structured in the same way, to reduce cognitive overload over the pages.

- Alternative text: when required, alternative text is provided for non-textual elements.
- Standards compliance. HTML and CSS content is accessible from the browser.

4.2 SEO Guidelines Compliance

The following Search Engine Optimization (SEO) good practices have been implemented:

- Clear URLs: the website's URLs are descriptive and contain relevant keywords that accurately reflect the content of their corresponding pages. Name of the teacher for the teacher page, name of the activity for the activity page etc.
- Effective meta descriptions: each page of the website also has a short but informative description that represents what to expect when navigating the page. Global settings are also applied.
- Unique title tags: each page of the website has a unique title tag that summarizes in a couple of words the content.
- Clear content hierarchy: different headers are used to be clear which content is more important.
- Mobile friendliness: the website adapts its layout based on the maximum available screen width.

4.3 Performance

To improve the performance of our Nuxt 3 application, we relied heavily on Lighthouse to identify bottlenecks and optimize critical loading metrics.

By running audits throughout development, we focused on improving metrics such as **Time to First Byte** (TTFB), **First Contentful Paint** (FCP), and **Largest Contentful Paint** (LCP). Nuxt 3's **server-side rendering** (SSR), combined with Vercel's global edge network, allowed us to serve pre-rendered pages quickly and benefit from automatic caching strategies.

We optimized images, minimized JavaScript payloads, and leveraged code-splitting and lazy loading to reduce initial load times. Tailwind CSS, integrated via `@nuxtjs/tailwindcss`, ensured that our styles were lightweight and compiled efficiently without unused CSS. Additionally, we reviewed our dependency tree and removed unnecessary packages to keep the frontend lean.

These optimizations, guided by Lighthouse insights, helped us deliver a performant, fast-loading experience across devices and network conditions.