POLITECNICO
MILANO 1863

TECHNOLOGY REPORT

HYPERMEDIA APPLICATIONS

Prof. Garzotto Franca – A.Y 2024/2025 15/07/2025

[YogaHeaven]

[GitHub Repository]

Submitted by Acceleration
Jiaxin Yang10719178
Run Jie Simone Dai 10766478

1. Work orgnization

Based on the project requirements, we divided the tasks into several key components. Throughout the process, we followed our plan diligently and supported each other in overcoming every technical challenge.

Member	Main Area of Work	%of Global Effort
Jiaxin Yang	UX Design, Low-Fidelity Wireframes,DB Data Insertion, Implementation, Styling, Report creation, Extra Functionalities, Interaction Scenarios, stripe setup.	50%
Run Jie Simone Dai	UX Design, Low-Fidelity Wireframes, DB Design, Implementation, Styling, Report creation, Accessibility, Interaction Scenarios, login/register.	50%

2. Our project

2.1 Project explanation

The aim of this project is to design and implement a fully functional website for a Yoga Center, following the specifications provided in the HYP 2024–2025 Design and Development Project. The website(YogaHeaven) is intended to present the center, its philosophy, its teachers, and the variety of activities it offers to the public.The website has been developed using Vue 3 and Nuxt 3, leveraging their modular and component-based architecture. The content is dynamically fetched from a Supabase database, ensuring scalability and easy content management. To enhance the functionality of our website, we also integrated Google Sign-In using Supabase authentication and implemented Stripe payments to allow users to book and pay for activities online in a secure and convenient way.

2.2 ChosenTheme

The theme we selected reflects the mission of the yoga center: promoting well-being, mindfulness, and physical balance through yoga lessons, meditation sessions, retreats, workshops, and seminars. The visual design adopts a green color scheme, symbolizing nature, tranquility, and inner peace—values that align perfectly with the essence of yoga.

2.3 Hosting service

The platform we adopted to host our website is Vercel, in combination with Supabase to store our data in a PostgreSQL database. This solution allowed us to build a more dynamic and interactive website compared to the alternative of GitHub Pages.

We opted for SSR (Server-Side Rendering) as the rendering mode, as we believe the server provided by Vercel is powerful and reliable. By offloading the rendering process to the server, we aim to reduce the computational load on client devices, ensuring smooth access for users regardless of their hardware.

Another advantage of using Vercel is its seamless integration with Git. Our project is automatically deployed every time we push changes to the repository, which simplifies the deployment workflow and keeps the online version always up to date.

2.4 Project Structure

2.4.1 Pages

Our website is structured in the following way:

• index.vue

• activities/type/[id].vue

/highlights.vue

/index.vue

/[id].vue

• teachers/[id].vue

/index.vue

• about_us.vue

• auth.vue

• checkout.vue

• contact_us.vue

• shoppingCart.vue

• success.vue

index.vue

The homepage of our website. It includes a slogan, highlighted activities, an overview of the team, and links to main sections of the site.

activities/type/[id].vue

Page for each individual activity. It displays activity details such as name, type, schedule, description, and the list of teachers involved.

activities/highlights.vue

This page presents a curated selection of highlighted activities the center promotes, featuring visual emphasis and quick access to detail pages.

activities/[id].vue

Displays the information of the activity with specific id, including more images and longer description. User can add to the course to cart or buy now.

teachers/[id].vue

The profile page of a teacher. It contains their picture, bio, specialization, and all the activities they are responsible for or teach.

teachers/index.vue

Shows a list of all teachers in the yoga center. Each card includes basic information and links to the detailed teacher profile.

about_us.vue

An informational page that introduces the yoga center, its mission, background, and core values.

auth.vue

Authentication page where users can log in or register, including integration with Google Sign-In via Supabase.

checkout.vue

The checkout page, where users can review selected items, enter billing information, and complete payment via Stripe.

contact_us.vue

Contact page containing phone, email, location map, and a form for messages or inquiries.

shoppingCart.vue

Displays all items the user has added to the cart. Users can modify quantities or proceed to checkout.

success.vue

Thank-you/confirmation page shown after successful payment. It confirms the booking and displays order summary.

2.4.2 ServerEndpoints

Activities

GET /api/activities/[id]
Endpoint that retrieves details of a specific activity.
The activity id is passed as a URL parameter.
Response: either an activity object (with its id, title, description, image, type, schedule, and related teachers) or an error message.

GET /api/activities/type/[id]
Endpoint that retrieves all the activities belonging to a specific activity type.
The activity type id is passed as a URL parameter.
Response: an object containing a list of activities (with id, title, short description, images) or an error message.

GET /api/starred.js
Endpoint that retrieves the list of highlight activities.
Response: a list of highlight activities or an error message.

Teachers

GET /api/teachers/[id]
Endpoint that retrieves full details of a specific teacher.
The teacher id is passed as a URL parameter.
Response: a teacher object (with id, name, image, cv, and the list of activities they are responsible for) or an error message.

Stripe / Payments

POST /api/stripe/paymentintent
Endpoint that creates a Stripe PaymentIntent for a checkout session.

Request body: contains cart items and total price.
Response: Stripe client secret or an error.

Orders / Cart

POST /api/create-order
Endpoint that creates an order after a successful Stripe payment.
Request body: contains user id, items, total, and payment confirmation.
Response: confirmation of order creation or an error.

2.5 Components Implemented

To reduce the redundancy of code, we decided to implement the following Vue components:

Content & Section Components

- ActivitiesSection.vue: It fetches data from Supabase, filters starred ones, randomly selects others, and shows them using ActivityComponent.
- ActivityComponent.vue: This component displays a single yoga activity in a card format. It accepts one prop named activity, which contains fields such as id, name, description, Type, img, price, oldPrice, star, and Teach. The component shows the activity's image, name, type label, price (including old price if available), and a list of teachers using TeacherAvatar.vue. If the activity is marked as starred, a yellow star icon appears in the corner. The image is clickable and links to the activity's detail page. This component does not emit any events.
- TeachersSession.vue:This component displays a grid of four teachers, fetched from the Supabase Teacher table. For each teacher, it retrieves basic information and their related courses via the Teach relationship. Each teacher is rendered using the TeacherComponent. The component does not accept props or emit events, and is used to highlight selected teachers, typically on the homepage.
- TeacherComponent.vue: This component displays a single teacher's profile card. It accepts one prop called teacher, which includes the teacher's id, name, surname, description, img, and a list of courses taught via the Teach relationship. The component shows the teacher's photo, name, a scrollable short description, and a list of their courses displayed as rounded tags. Each course tag is a link to the corresponding activity detail page. The component does not emit any events.

- TeacherAvatar.vue: This component displays a small, circular avatar of a teacher along with their name. It accepts one prop, teacher, which includes the teacher’s id, name, surname, and img. The avatar and name are wrapped in a link that navigates to the teacher’s detail page. It is typically used to display teaching staff in course cards or compact layouts. The component does not emit any events.

Layout & Structure Components

- TopBar.vue: This component implements the website's fixed navigation bar, visible across all pages. It adapts to both desktop and mobile views, offering a responsive layout with dropdown menus, cart item count, and user login/logout options. It dynamically renders links from a links array, supports authentication via Supabase, and merges the local cart with the server using overrideCartBeforeSignOut before sign-out. The design includes dropdown transitions, mobile navigation toggles, and dynamic cart updates from the Pinia store. No props or emits are used.

- Footer.vue: The Footer.vue component displays the website’s footer with navigation links, contact information, and social media icons. On large screens, the Activities section is split into main and sub-links across two columns, while on smaller screens it's grouped together. It uses computed properties to separate Activities from other links and adapts its layout responsively.

- MenuOverlay.vue: The MenuOverlay.vue component is a full-screen mobile overlay menu that displays navigational options like "My Orders," "Cart," and sign-in/sign-out based on user login status. It fetches user and cart state from the Pinia userStore, supports navigation using navigateTo, and dynamically closes the overlay when a link is clicked or the close button is pressed.

- Breadcrumbs.vue: The Breadcrumb.vue component dynamically generates a breadcrumb navigation based on the current route. It uses useRoute() to access the current path, splits it into segments, and constructs an array of breadcrumb items. Each segment (except the last one) is rendered as a clickable NuxtLink, while the final segment is displayed as plain text. This provides users with a clear, hierarchical view of their navigation path within the application.

Content Block/Band Components

- Gallery.vue: This component displays a responsive grid of interactive cards, each representing an experience (e.g. nature, city, stargazing). Each card has a background image, a tag, title, description, and a list of features. On hover, the card shows a brief description and slides up to reveal a full list of features and a call-to-action button. It uses props like title, tag, image, description, and features per card, and internally tracks isHovered for animation control. The design emphasizes visual storytelling and smooth transitions using Tailwind CSS.

- PageTitle.vue: The PageTitle.vue component is a reusable section header that displays a centered title and an optional short description. It accepts two props: title (required) and shortDescription (optional). This component helps standardize the appearance of section headings across different pages, enhancing visual consistency.

- SessionTitle.vue: This component displays a section title with a stylish left alignment and an animated underline that appears on hover. Below the title, it shows a brief description in a lighter font color. It accepts two props: title (required) and shortDescription (required), and uses a hover state to control the visual transition effect. The layout is optimized for readability and works well in content sections to highlight important headings and descriptions.

- BandForm.vue: This component renders a contact section that includes both an informative introduction and a contact form. On the left side, it displays a title, a helpful description message, an email address, and a set of social media icons with hover effects linking to Instagram, Facebook, and Twitter. On the right, it embeds a <ContactForm /> component for users to submit inquiries. The layout is responsive: on smaller screens, the two columns stack vertically to ensure accessibility and readability. The design uses a soft purple palette and the "Jost" font for a clean and modern look.

- BandLeftImg.vue: This component presents a responsive two-column layout combining an image and a text block with a call-to-action button. It receives five props: description (text content), namebutton (label for the button), imageName (image file name), link (button target URL), and aria_label (for accessibility). The image is displayed on the left, and the text with a violet-styled button (via <ButtonViolet />) is shown on the right. The layout adapts for smaller screens by stacking the columns vertically while keeping the style clean and centered. The image source is customizable through the imageName prop.

- BandLeftNoButton.vue: This component renders a responsive two-column layout where a description is displayed on the left and an image on the right. It accepts two props: description for the textual content and imageName for the image file name. Although the imageSrc is computed, the template mistakenly uses imageName directly in the src, which might cause issues if the image path isn't fully specified. On smaller screens, the layout stacks with the image on top and the text below, ensuring mobile-friendly presentation. There are no emitted events.

- BandRightNoButton.vue: This component displays an image and a text description side by side, with the image on the left and text on the right. It takes two props: description for the content and imageName for the image path. The layout is responsive and stacks vertically on smaller screens. No emits or interactivity are included.

- LeftImageRightText.vue: This component presents a responsive two-column layout with an image on the left and text content on the right. It accepts `imageUrl`, `imageAlt`, `title`, and `description` as props to display the main visual and content. If `features` are provided, they are shown as a list with checkmark icons. A call-to-action button is displayed if `ctaText` is set. No emits are used, and the logic is purely presentational.

- TextOverImg.vue: This component displays a full-width, full-height background image with a centered inspirational quote overlay. It accepts two props: img (the background image URL) and text (either a single string or an array of strings). Each line of text is rendered with a hover scaling effect, and if text is an array, each item is shown as a separate line. The layout includes a translucent background, centered content, and a decorative divider. It contains no emitted events and is purely presentational.

E-Commerce Components

- CartItem.vue: This component renders a single product item in the shopping cart with a custom circular selection checkbox, image, name, price, discount badge, and a delete icon. It accepts two props: product (an object with product details like id, name, image, price, oldPrice) and selectedArray (an array of currently selected products). It emits one event, selectedRadio, when the selection state is toggled. The component uses userStore to manage cart state and allows the user to remove the product from the cart via the delete icon. It also includes a hover effect for the custom checkbox.
- CheckoutItem.vue: This component displays a simple product summary for use in the cart or order review. It accepts a single product prop, which is an object containing fields like image, name, price, and oldPrice. It shows the product image, name, and current price, and

conditionally displays a "Discount" badge and old (strikethrough) price if applicable. There are no emitted events or internal logic beyond rendering the data passed via props.

Form & Input Components

- ContactForm.vue: This is a contact form component with input validation. It uses Vue's options API and accepts no props or emits. Internally, it handles name, surname, number, email, and text as form fields. Upon blur or form submission, it performs validation: the phone number must be exactly 10 digits, the email must match a standard pattern, and the message must not be empty. Validation errors are shown dynamically, and the submit button is disabled unless the form is valid. On successful submission, the form resets. The component uses scoped styles with responsive design for mobile.
- TextInput.vue: This is a reusable input field component that supports v-model via the input prop and update:input emit. It accepts the following props: input (String), placeholder (String), max (Number for maxLength), inputType (e.g. 'text', 'email'), and error (String for validation message). When focused, the input border becomes dark; when an error is present, the border turns red and an error message is shown below. The inputComputed computed property ensures two-way binding. It's wrapped in <client-only> to avoid SSR issues, especially when used with third-party form libraries.

2.6 Extra Modules

In this project, we integrated several extra modules to enhance functionality and user experience.

Supabase serves as our backend infrastructure, offering real-time database access, user authentication, and file storage.

Pinia is chosen for lightweight yet powerful state management, replacing Vuex and enabling a modular, composition-API-friendly architecture.

For payment processing, we implemented Stripe, allowing us to securely handle transactions with support for checkout, subscriptions, and webhook-based workflows.

2.7 Accessibility and SEO

2.7.1 Accessibility

Accessibility was a fundamental consideration throughout the development of our website. We ensured that the content is readable and usable for all users, including those relying on assistive technologies. Key practices include:

- Semantic Structure & Readability: All pages maintain clear layout hierarchy using semantic HTML elements. We adopted legible font sizes and ensured high contrast between foreground and background (e.g., white background with dark text or violet buttons with white text).
- Descriptive Labels: Every interactive component (buttons, links, form inputs) is enhanced with aria-label attributes to provide descriptive context. This supports screen readers in understanding the element's purpose.
- Alternative Text for Images: Each <img> element in the project includes a meaningful alt attribute or uses :alt="..." bindings in Vue components. This helps users who cannot visually access images.
- Keyboard Navigation & Hover States: Hover effects and focus outlines are implemented using Tailwind utilities and transitions to support both keyboard and mouse users.
- Assistive Feedback on Forms: All inputs in the contact form provide real-time validation feedback, helping users immediately understand and correct errors (e.g., email format or required fields).
- Tools Used: We tested accessibility using Lighthouse and WAVE, achieving high scores across all tested pages.

2.7.1.1 Responsiveness

Our entire layout is mobile-first and built using responsive units (vw, vh, %, max-w, min-h, etc.). We used Tailwind CSSand CSS media queries to ensure the design adapts across smartphones, tablets, and desktops. Flexbox and grid layouts were dynamically adjusted via utility classes based on screen size breakpoints, ensuring optimal readability and user experience on all devices.

2.7.2 SEO

To help our website reach as many users as possible, particularly clients who are interested in yoga or sports, SEO was carefully integrated into our Nuxt project. The measures taken include:

- Meta Information: We utilized the Nuxt head configuration to dynamically inject meta title, description, and keywords tags for each page, enabling search engines to better index and rank our content.
- Social Media Tags: Open Graph (og:) and Twitter card meta tags were included to enhance how pages appear when shared on platforms like Facebook, Instagram, or X (Twitter). This ensures thumbnails, titles, and descriptions appear correctly.
- Descriptive Links: All links use clear, informative labels (e.g., Explore Now, View Profile, Sign In / Register) instead of vague terms like "Click here", which improves both usability and SEO.
- Clean URLs and Semantic Routes: Nuxt's file-based routing allowed us to maintain SEO-friendly and human-readable URLs (e.g., /teachers/john-doe).
- Validation Tools: SEO setup was validated through Lighthouse audits, and structured content was ensured to support rich results where applicable.

2.8 Extra Functionalities

In addition to the mandatory requirements, we implemented several extra features that significantly enhance the website's usability and user experience.

2.8.1 E-commerce

We integrated a full e-commerce system allowing users to browse activities or courses and add them to a shopping cart. Users can view, update, and remove items in the cart before proceeding to payment. Payments are securely handled via Stripe, supporting smooth transaction flows. The cart state is managed globally using Pinia, with synchronization to the backend (Supabase) to persist user selections. The data of carts are saved to local storage even though the user does not login, and once user login, the local cart will be merged to the cart data saved in Supabase.

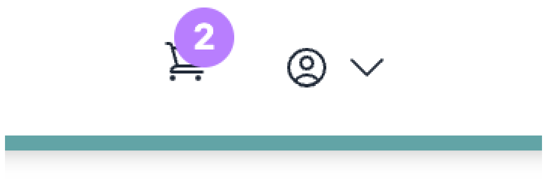


figure 2.8.1.1: Shopping cart Enter point

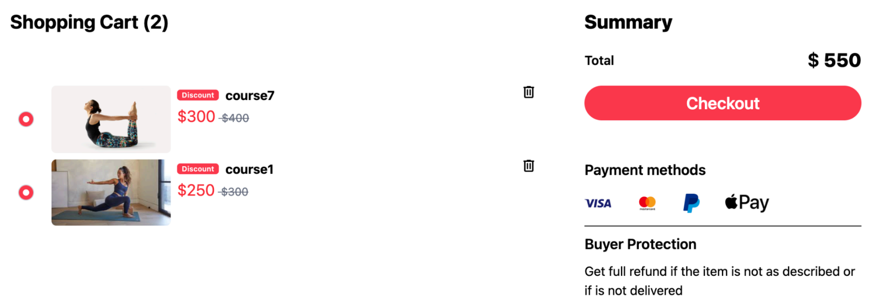


figure 2.8.1.2: Shopping cart

2.8.2 Login/Register with Google Account

To simplify user authentication, we implemented social login using Google accounts via Supabase's OAuth integration. This feature allows users to quickly register and log in without creating separate credentials, improving accessibility and reducing friction. The authentication state is centrally managed and reflects instantly across the UI.

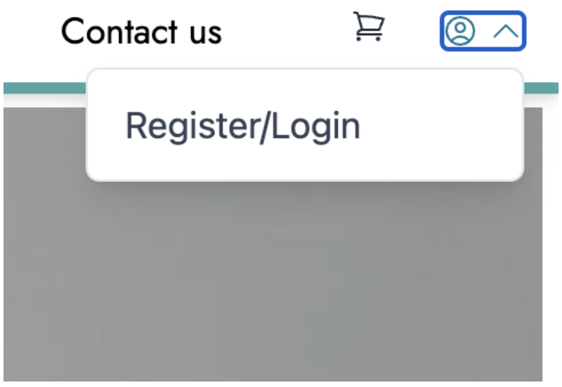


figure 2.8.2.1: Register/Login enter point

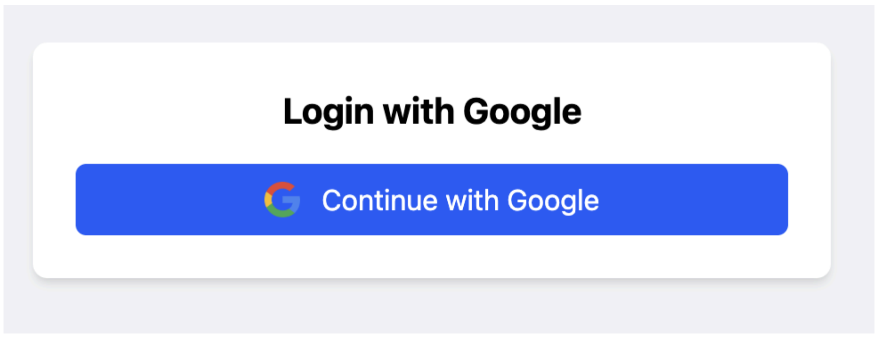
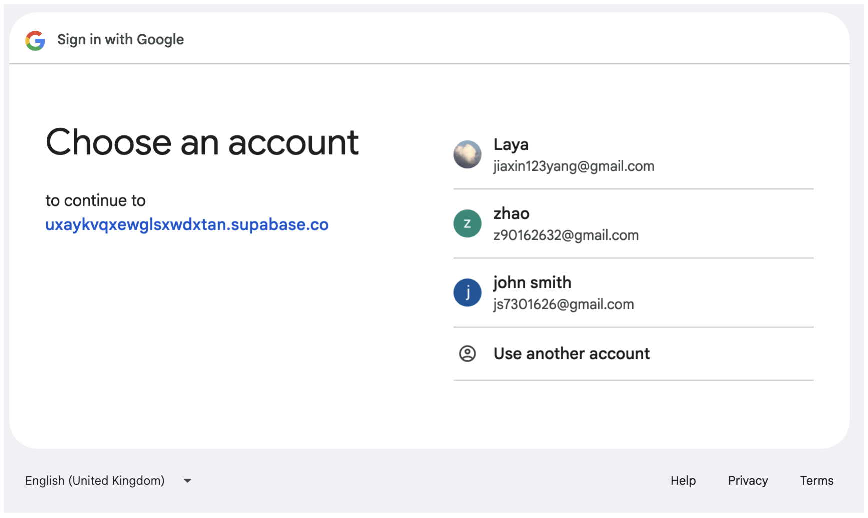


figure 2.8.1.2: auth page


Sign in with Google

Choose an account

to continue to uxaykvqxewglsxwdxtan.supabase.co

Laya
jiaxin123yang@gmail.com

zhao
z90162632@gmail.com

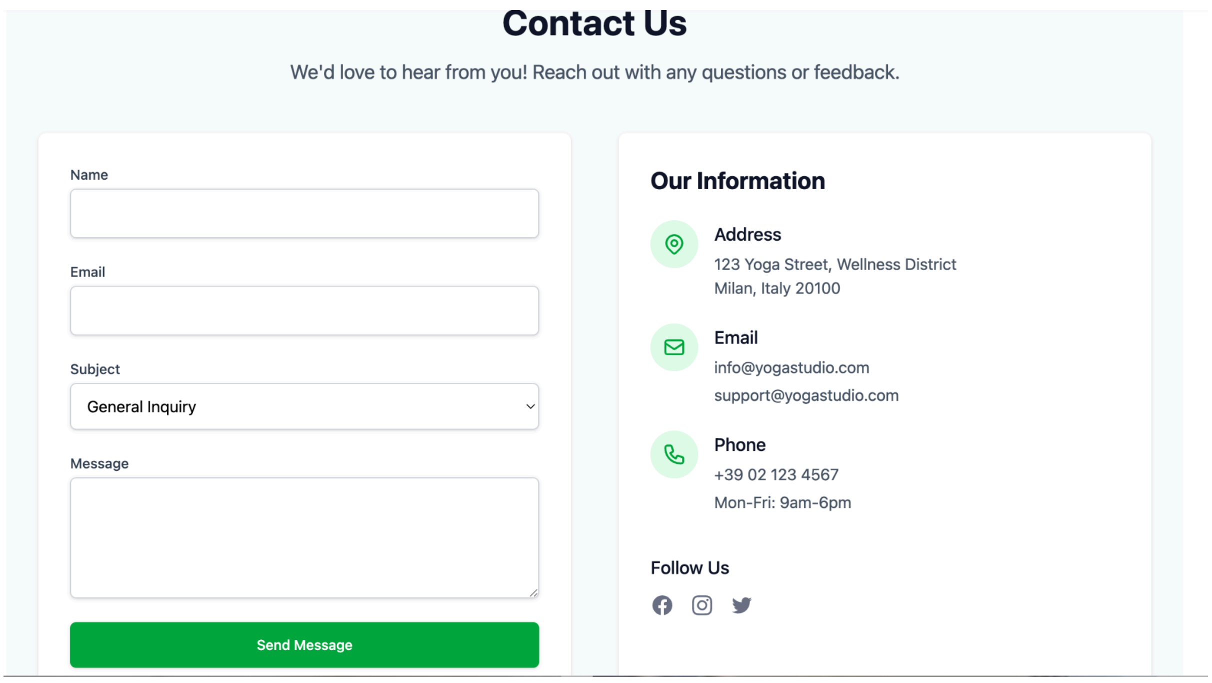
john smith
js7301626@gmail.com

Use another account

figure 2.8.1.2: Google authentification window

2.8.3 Contact Form

We developed a contact form enabling users to submit inquiries or requests directly through the website. The form includes fields for name, surname, phone number, email, and message, all with real-time validation to ensure data quality. Upon submission, the form resets and can be extended to send data to an API or email service for processing.


Contact Us

We'd love to hear from you! Reach out with any questions or feedback.

Name

Email

Subject: General Inquiry

Message

Our Information

Address
123 Yoga Street, Wellness District
Milan, Italy 20100

Email
info@yogastudio.com
support@yogastudio.com

Phone
+39 02 123 4567
Mon-Fri: 9am-6pm

figure 2.8.3: Contact Us page

2.8.4 Filter Activities by Type

Users can filter the list of available activities or courses by their type or category. This interactive filtering improves navigation and helps users find relevant content quickly. The filtering logic is implemented on the client side, dynamically updating the displayed list without page reloads, thus enhancing responsiveness and user experience.

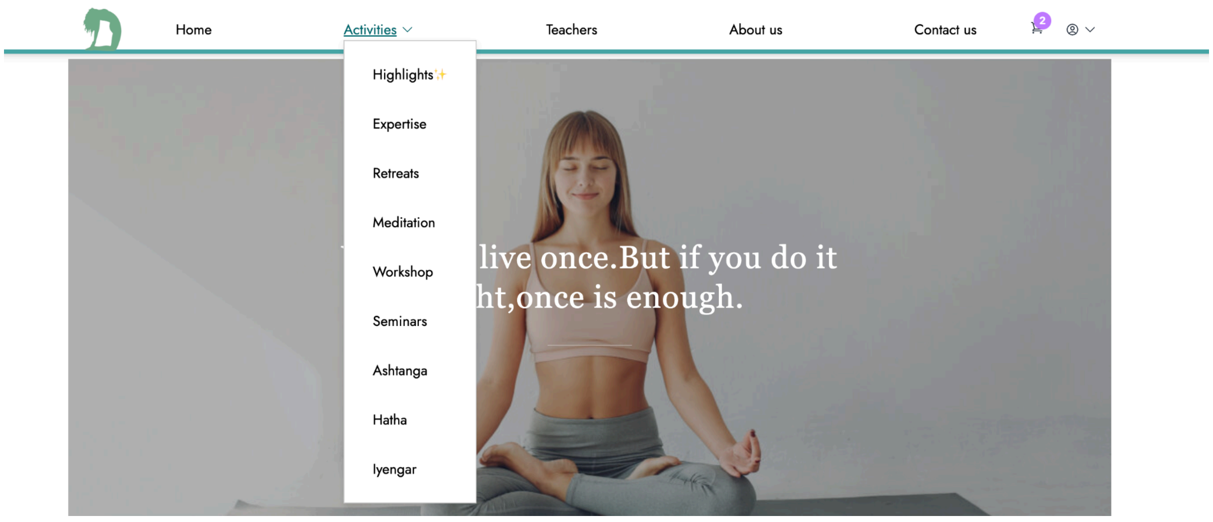


figure 2.8.3: Activity submenu