POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Title

Tesi di Laurea Magistrale in
Xxxxxxx Engineering - Ingegneria Xxxxxxx

Author: **Name Surname**

Student ID: 000000
Advisor: Prof. Name Surname
Co-advisors: Name Surname, Name Surname
Academic Year: 20XX-XX

Abstract

Here goes the Abstract in English of your thesis followed by a list of keywords. The Abstract is a concise summary of the content of the thesis (single page of text) and a guide to the most important contributions included in your thesis. The Abstract is the very last thing you write. It should be a self-contained text and should be clear to someone who hasn't (yet) read the whole manuscript. The Abstract should contain the answers to the main scientific questions that have been addressed in your thesis. It needs to summarize the adopted motivations and the adopted methodological approach as well as the findings of your work and their relevance and impact. The Abstract is the part appearing in the record of your thesis inside POLITesi, the Digital Archive of PhD and Master Theses (Laurea Magistrale) of Politecnico di Milano. The Abstract will be followed by a list of four to six keywords. Keywords are a tool to help indexers and search engines to find relevant documents. To be relevant and effective, keywords must be chosen carefully. They should represent the content of your work and be specific to your field or sub-field. Keywords may be a single word or two to four words.

Keywords: here, the keywords, of your thesis

Contents

1 | Introduction

1.1. Group Information

Group Name: HyperDaGiaZeySe

Group Composition:

Group Composition

Name Surname	Person Code
Zeynep Erbaysal	11035715
Sebastian Perea	10986638
Daniel Ruiz	10988760
Giacomo Scampini	10764570

1.2. Project Information

Website: `https://lotus-haven.vercel.app/`
GitHub Repository: `https://github.com/sebaxe07/LotusHaven`

1.3. Work Breakdown

Throughout the project, we aimed to maintain a balanced and collaborative workflow, ensuring that each team member could contribute and learn from different aspects of the development process. As the project progressed, we naturally divided responsibilities to leverage individual strengths and improve efficiency. Sebastian Perea and Daniel Ruiz primarily focused on the core development and implementation of the application, while Zeynep Erbaysal and Giacomo Scampini concentrated on the website's styling and user interface design. Despite this division, all members participated in discussions and contributed ideas to both the technical and design components, resulting in a cohesive and well-rounded final product.

2 | Documentation

2.1. Chosen Theme

This project presents the design and development of a yoga center website focused on promoting wellness, clarity, and connection. The website architecture is built around key entities such as instructors, activity types, and scheduled sessions, with main navigational pages including the homepage, About Us, and Contact Us. The design process included structured content organization, detailed wireframes with design rationale, and realistic user interaction scenarios to ensure intuitive navigation. The underlying database schema supports efficient data flow and user interaction, providing a cohesive and user-friendly experience for visitors seeking information about yoga activities and instructors.

2.2. Technological Choices

- **Hosting: Vercel** — The website is deployed and hosted on Vercel, a cloud platform optimized for frontend frameworks and static sites. Vercel provides seamless integration with GitHub for continuous deployment, fast global CDN, and automatic SSL, ensuring high availability and performance.
- **Database: Supabase** — Supabase is used as the backend database solution. It offers a scalable, open-source alternative to Firebase, providing a PostgreSQL database, authentication, and real-time capabilities. Supabase enables secure storage and efficient retrieval of data related to instructors, activities, and sessions.
- **Rendering Mode: Server-Side Rendering (SSR)** — The project is configured to use Nuxt's server-side rendering mode, as specified in the configuration file. All routes are rendered on the server by default, providing improved SEO, performance, and dynamic content handling. This ensures that users receive fully rendered pages from the server, with client-side hydration for interactivity.
- **Main Framework: Nuxt** — The application is built using Nuxt, a powerful Vue.js framework for building modern web applications. Nuxt provides features

such as file-based routing, automatic code splitting, and an intuitive module system, streamlining development and improving maintainability.

- **Programming Language: TypeScript** — TypeScript is used throughout the codebase to provide static typing, improved code quality, and better developer tooling. TypeScript helps catch errors early and enhances the maintainability of the project.
- **Component-Based Architecture** — The project is organized using a modular, component-based structure, with reusable UI components for buttons, cards, carousels, and more. This promotes code reuse and simplifies updates and testing.
- **Version Control: GitHub** — Source code is managed using Git and hosted on GitHub, enabling collaboration, version tracking, and integration with CI/CD pipelines.

2.3. Project Structure

2.3.1. Links/Pages Structure

Page	URL	Description
Home	/	Landing page with hero section, featured activities, teacher highlights, and call-to-action sections providing an overview of the yoga studio.
Highlights	/highlights	Displays featured yoga classes and special activities, including a prominently featured activity and a carousel of additional highlighted options.
Activities	/activities	Lists all available yoga classes with search functionality and filtering options. Users can browse and find detailed information about each activity.
Activity Detail	/activity/[id]	Shows detailed information about a specific yoga class, including description, difficulty level, duration, and teacher information.
Teachers	/teachers	Displays all yoga instructors with their specialties and a brief introduction. Users can click to view detailed profiles.
Teacher Profile	/teacher/[id]	Provides comprehensive information about a specific teacher, including their bio, expertise, contact information, and scheduled classes.
About Us	/about	Contains information about the yoga studio philosophy, mission, location, operating hours, and other facility details.
Contact Us	/contact	Features contact information, staff directory, location map, and a contact form for inquiries and class registrations.

Table 2.1: Website navigation structure showing main pages and their purposes

2.3.2. Database Structure and Data Access

This project utilizes Supabase as a backend-as-a-service solution rather than implementing traditional REST API endpoints. The data layer is structured around three main database tables:

Table	Description
Teachers	Stores information about yoga instructors, including name, bio, expertise, and contact details.
Activities	Contains data about yoga classes and activities, including name, description, difficulty level, duration, and whether they are highlighted.
TeacherActivities	Junction table managing the many-to-many relationship between teachers and activities, including scheduling information (days and times).

Table 2.2: Supabase Database Structure

Data is accessed through Supabase client queries in dedicated composables:

- **useTeachers.ts** — Provides functions for fetching all teachers (`fetchTeachers()`) and retrieving a specific teacher by ID (`fetchTeacherById()`) along with their associated activities.
- **useActivities.ts** — Handles fetching all activities (`fetchActivities()`), highlighted activities (`fetchHighlightedActivities()`), and retrieving specific activities by ID (`fetchActivityById()`).

These composables encapsulate the data access logic, transforming raw database records into domain models used throughout the application.

2.4. Custom Types

The project uses TypeScript to define and enforce a clear type system throughout the application. Types are organized into two main categories: database-related types that represent data models and UI component types used for presentation.

2.4.1. Database and Domain Model Types

These types represent the core data structures of the application, including raw database formats and their transformed domain models:

Type Name	Description
`Teacher`	Core teacher model containing personal and professional information including ID, name, surname, biography, contact details, and associated activities.
`Activity`	Core activity model representing a yoga class including title, descriptions, schedules, difficulty level, and visual assets.
`TeacherActivity`	Junction model representing the relationship between teachers and activities, including scheduling information.
`Schedule`	Represents a class schedule with time, days, and assigned teacher information.
`RawTeacherData`	Database response type for teacher records before transformation to domain models.
`RawActivityData`	Database response type for activity records before transformation to domain models.
`RawTeacher-ActivityData`	Database response type for the teacher-activity relationship records.

Table 2.3: Database and Domain Model Types

2.4.2. UI and Presentation Types

These types are designed for optimized display in UI components, often with simplified or enhanced properties:

Type Name	Description
`TeacherCardItem`	Simplified teacher presentation model optimized for card display with essential information and formatting properties.
`teacherToCardItem2`	Enhanced teacher card model that includes a list of activities taught by the teacher.
`ClassCardItem`	Presentation model for displaying activities in card format with visual properties like colorVariant.

Table 2.4: UI and Presentation Types

The application uses TypeScript interfaces to define these types, providing strong typing throughout the codebase. This approach enhances code quality through compile-time type checking and improves developer experience with better autocomplete and documentation.

TypeScript's structural typing system allows for flexible type transformations between database representations and UI components, maintaining type safety while accommodating different data formats needed at various application layers.

2.5. Custom Components

The project implements a comprehensive set of custom components to ensure consistency, reusability, and maintainability throughout the application. These components serve as the building blocks of the user interface, each designed with specific functionality while maintaining a cohesive visual language. By developing these custom components, the project achieves several key objectives:

- **Design Consistency** — Custom components ensure a unified look and feel across all pages, reinforcing brand identity and improving user experience through visual coherence.
- **Code Reusability** — Components are designed to be reused across different pages and contexts, reducing code duplication and development time while ensuring consistent behavior.
- **Maintainability** — Centralizing UI logic in discrete components simplifies debugging, updates, and feature additions by isolating changes to specific modules rather than scattered throughout the codebase.
- **Responsiveness** — Components are built with responsive design principles, adapt-

ing to different screen sizes and device types to provide an optimal viewing experience.

- **Type Safety** — All components leverage TypeScript for prop validation and interface definitions, preventing runtime errors and providing better developer tooling support.

The component library is organized into functional categories, each serving distinct interface needs while maintaining consistency with the overall design system of the yoga studio website.

2.5.1. ActivityCard Component

The `ActivityCard` component serves as a reusable display element for yoga activities throughout the website. Its primary goal is to present activity information in a consistent and visually appealing card format, with support for images, descriptions, difficulty indicators, and navigation to detailed views.

Prop	Type	Required	Short Description
activity	Activity	Yes	The activity object containing all necessary data to display, including title, description, images, and difficulty level.

Table 2.5: ActivityCard Component Props

The component is used throughout the application to display activities on the activities listing page, teacher profiles, and featured sections on the homepage.

2.5.2. Button Component

The `Button` component provides a highly configurable button element that adapts its appearance based on provided props. Its primary goal is to create consistent, accessible, and visually appealing interactive elements with support for multiple visual variants, sizes, and states.

Prop	Type	Require	Short Description
text	string	No	Button text content displayed when no slot content is provided.
color	"primary" | "secondary" | "white" | "outline" | "outline-white"	No	Visual style variant applying different color schemes.
disabled	boolean	No	Controls whether the button is clickable or in disabled state.
extraClasses	string	No	Additional CSS classes for custom styling beyond variants.
fullWidth	boolean	No	When true, the button expands to fill its parent container width.
size	"sm" | "md" | "lg"	No	Controls button size, affecting padding and text size.

Table 2.6: Button Component Props

The component is used extensively throughout the application for calls to action, form submissions, navigation triggers, and interactive elements.

2.5.3. CarouselArrow Component

The `CarouselArrow` component provides a circular navigation button with directional arrow icons. Its primary goal is to offer consistent carousel controls with directional indicators that can be placed anywhere within carousel components to navigate through content.

Prop	Type	Require	Short Description
direction	"left" | "right" | "up" | "down"	No	Determines the direction the arrow points using icon rotation.
size	number	No	Controls the diameter of the circular button in pixels.

Table 2.7: CarouselArrow Component Props

The component is used within carousel interfaces throughout the application, particularly in activity and teacher display carousels on the home page and highlights section.

2.5.4. ClassCard Component

The `ClassCard` component presents yoga class information in an elegant card format. Its primary goal is to display class details with visual distinction and provide a consistent interface for users to learn more about specific classes or activities.

Prop	Type	Require	Short Description
image	string	No	URL to the class image; if not provided, a colored background is used.
title	string	Yes	Title of the yoga class or activity.
description	string	Yes	Short description of the class content or benefits.
schedules	Schedule[]	No	Array of available class schedules with time, days, and instructor information.
id	string | number	Yes	Unique identifier for the class, used for navigation.
colorVariant	"primary" | "secondary" | "third"	No	Color theme for cards without images.

Table 2.8: ClassCard Component Props

The component is used on class listing pages and in featured sections to display yoga classes with their available sessions and provide navigation to detailed class information.

2.5.5. ClassesCarousel Component

The `ClassesCarousel` component provides a responsive horizontal scrolling container for displaying multiple class cards. Its primary goal is to create an interactive browsing experience for yoga activities with intuitive navigation controls and smooth scrolling behavior.

Prop	Type	Require	Short Description
activities	ClassCardItem[]	Yes	Array of activity/class items to display in the carousel with their details.

Table 2.9: ClassesCarousel Component Props

The component is used on the homepage and highlights section to showcase multiple activities in a space-efficient, horizontally-scrollable interface with navigation arrows and mousewheel support.

2.5.6. ContactCard Component

The `ContactCard` component displays staff contact information in a consistent card format. Its primary goal is to present contact details with visual prominence while maintaining the design language of the application.

Prop	Type	Require	Short Description
name	string	Yes	The name of the contact person to display.
role	string	Yes	The role or position of the contact person.
description	string	Yes	Brief description or additional information about the contact.
imageUrl	string	Yes	URL to the contact's profile image.

Table 2.10: ContactCard Component Props

The component is used on the Contact page to display yoga studio staff information, including instructors and administrative personnel.

2.5.7. DecorativeIllustration Component

The `DecorativeIllustration` component provides a stylized lotus flower SVG illustration. Its primary goal is to add visual interest and reinforce the yoga studio brand identity through decorative graphical elements using CSS variables for consistent theming.

Prop	Type	Require	Short Description
No props required - zero-configuration component			

Table 2.11: DecorativeIllustration Component Props

The component is used as a decorative element on various pages, particularly in the About section and on content pages where visual interest is needed without requiring additional imagery.

2.5.8. FeaturedHighlight Component

The `FeaturedHighlight` component presents a prominent, responsive display for featured yoga activities. Its primary goal is to showcase a selected activity with enhanced visual treatment, comprehensive information, and upcoming sessions to drive user engagement.

Prop	Type	Require	Short Description
id	string | number	Yes	Unique identifier for the featured activity.
title	string	Yes	Title of the featured class or activity.
description	string	Yes	Descriptive text about the featured class or activity.
image	string	No	URL to the featured item's image; if not provided, a colored background is used.
schedules	Schedule[]	No	Array of available session schedules with time, days, and instructor information.
colorVariant	"primary" | "secondary" | "third"	No	Color theme variant controlling background colors and accents.
difficultyLevel	number	No	Difficulty level of the class (1: Beginner, 2: Intermediate, 3: Advanced).

Table 2.12: FeaturedHighlight Component Props

The component is used on the homepage and highlights page to prominently feature special activities or promotions with enhanced visual treatment and scheduling information.

2.5.9. Navbar Component

The `Navbar` component provides the main navigation interface for the website. Its primary goal is to offer consistent site-wide navigation with responsive behavior that adapts between a desktop sidebar and a mobile top navigation menu with dropdown.

Prop	Type	Require	Short Description
No props required - handles responsive behavior internally			

Table 2.13: Navbar Component Props

The component is used throughout the application as part of the default layout, providing navigation links, logo display, and social media icons in both desktop and mobile views.

2.5.10. SearchBar Component

The `SearchBar` component provides a clean search input interface with an integrated search button. Its primary goal is to offer a consistent search experience with proper accessibility features and visual feedback.

Prop	Type	Require	Short Description
modelValue	string	Yes	Current search text value that binds with v-model for two-way data binding.

Table 2.14: SearchBar Component Props

The component is used on the activities and teachers listing pages to enable filtering and searching through available content.

2.5.11. SectionCard Component

The `SectionCard` component displays detailed class schedule information with instructor details. Its primary goal is to present session scheduling data and instructor contact information in a visually organized card format with interactive elements.

Prop	Type	Require	Short Description
schedule	Schedule	Yes	Schedule object containing time, days, and instructor information with contact details.

Table 2.15: SectionCard Component Props

The component is used on activity detail pages to display class sessions with instructor information, including contact details and schedule times.

2.5.12. SvgIcon Component

The `SvgIcon` component provides a flexible SVG loading and rendering utility. Its primary goal is to dynamically fetch and display SVG icons with configurable sizing and coloring while maintaining accessibility and visual consistency.

Prop	Type	Require	Short Description
icon	string	Yes	Path to the SVG file to be loaded and displayed.
width	number | string	No	Icon width in pixels.
height	number | string	No	Icon height in pixels.
color	string	No	Color to apply to SVG elements; can use CSS color values or "currentColor".

Table 2.16: SvgIcon Component Props

The component is used throughout the application to display icons in navigation elements, cards, buttons, and informational sections with consistent styling and color theming.

2.5.13. TeacherCard Component

The `TeacherCard` component provides a compact visualization of instructor information. Its primary goal is to display teacher profiles in a visually appealing card format with image, name, and optional description in an efficient space-conscious layout.

Prop	Type	Require	Short Description
id	number	Yes	Unique identifier for the teacher.
name	string	Yes	Teacher's display name.
imageUrl	string	Yes	Path to the teacher's profile image.
description	string	No	Optional short description or bio text.
cardWidth	string	No	Optional width specification (CSS value).

Table 2.17: TeacherCard Component Props

The component is used on the teachers listing page and in carousels on the homepage to display available instructors in a grid or scrollable layout.

2.5.14. TeachersCarousel Component

The `TeachersCarousel` component provides a responsive horizontal scrolling container for displaying multiple teacher cards. Its primary goal is to create an interactive browsing experience for yoga instructors with intuitive navigation controls, dynamic card sizing based on screen width, and smooth scrolling behavior.

Prop	Type	Require	Short Description
No direct props			

Table 2.18: TeachersCarousel Component Props

The component is used on the homepage and teachers listing page to showcase multiple instructors in a space-efficient, horizontally-scrollable interface with navigation arrows and enhanced mousewheel support. It dynamically adapts to screen size, showing between 1.5 (mobile) and 4 (desktop) cards at once.

2.5.15. TeacherSchedule Component

The `TeacherSchedule` component displays a teacher's class schedule in a compact, interactive card format. Its primary goal is to present scheduled activities with time, day

information, and visual yoga icons in an easily scannable layout that supports navigation to detailed schedule views.

Prop	Type	Require	Short Description
id	number | string	Yes	Unique identifier for the schedule.
title	string	Yes	Title of the scheduled activity.
icon_id	number	Yes	ID for the yoga icon to display.
time	string	Yes	Time of the scheduled activity (formatted string).
days	string[]	Yes	Array of days when the activity occurs.

Table 2.19: TeacherSchedule Component Props

The component is used on teacher profile pages to display class schedules and in schedule listing interfaces, providing users with interactive access to detailed schedule information with consistent visual representation.

2.6. Extra Modules

In addition to the core Vue.js and Nuxt frameworks, the LotusHaven project utilizes several specialized modules and integrations to enhance functionality, improve developer experience, and extend the application's capabilities. These modules are integrated through Nuxt's modular architecture, providing seamless extensions to the core functionality.

2.6.1. Database and Backend Integration

Module	Implementation and Usage
Supabase	Integrated through the official `@nuxtjs/supabase` module to provide a serverless backend solution. The application uses Supabase for: • **Database Services**: PostgreSQL database for storing teachers, activities, and scheduling information. • **Data Relationships**: Managing many-to-many relationships between teachers and activities via junction tables. • **Query API**: Custom queries with nested relationship data using Supabase's query builder in the `useActivities` and `useTeachers` composables. The database design follows a normalized structure with foreign key relationships, and the Supabase client is used to fetch and transform data into typed domain models.

Table 2.20: Database and Backend Modules

2.6.2. Content Management

Module	Implementation and Usage
Nuxt Content	Implemented via `@nuxt/content` to provide a file-based content management system for static content. The module is used to: • **Manage Markdown Content**: Store and render rich text content for static pages like About Us. • **Content Querying**: Enable dynamic fetching of content based on page requirements. • **Content Presentation**: Transform content into Vue components with proper formatting and styling. This approach separates content from code, allowing for easier updates to text-heavy sections without requiring code changes.

Table 2.21: Content Management Modules

2.6.3. Asset and Media Management

Module	Implementation and Usage
Nuxt Image	Integrated through `@nuxt/image` to provide advanced image processing capabilities. The module is used for: • **Responsive Images**: Automatically generating different image sizes for various device widths. • **Lazy Loading**: Optimizing page load performance by deferring image loading until visible. • **Format Optimization**: Converting images to modern formats like WebP when supported. Particularly important for optimizing the numerous images used throughout the yoga studio website, including teacher photos, activity images, and decorative graphics.
Nuxt Icon	Used via `@nuxt/icon` to manage icon assets throughout the application. This module: • **Centralizes Icon Management**: Provides a consistent API for using SVG icons. • **Optimizes SVG Assets**: Reduces icon file sizes through optimizations. • **Simplifies Implementation**: Works with the custom `SvgIcon` component to standardize icon usage. The website extensively uses icons for navigation, feature indicators, and visual communication of yoga poses.

Table 2.22: Asset and Media Management Modules

2.6.4. Development and Build Tooling

Module	Implementation and Usage
Tailwind CSS	Integrated using `@tailwindcss/vite` as a Vite plugin. Tailwind CSS is used throughout the application for: • **Utility-First Styling**: All component styling uses Tailwind's utility classes. • **Responsive Design**: Viewport-based responsive layouts using Tailwind's breakpoint system. • **Theming**: Custom color palette and typography configuration to match yoga studio branding. Tailwind significantly accelerated UI development by eliminating the need for custom CSS in most cases.
Nuxt ESLint	Implemented via `@nuxt/eslint` to ensure code quality and consistency. The module provides: • **Nuxt-Specific Rules**: Enforcing best practices specific to Nuxt applications. • **TypeScript Integration**: Enhanced linting for typed code, reducing potential errors. • **Consistent Code Style**: Enforcing style guidelines across the codebase. ESLint configurations are extended with TypeScript-specific rules to ensure type safety in the codebase.

Table 2.23: Development and Build Tooling Modules

2.6.5. User Interface Enhancement

Module	Implementation and Usage
Nuxt UI	Utilized through `@nuxt/ui` to provide a foundation of accessible, pre-styled components. This module: • **Extends Component Library**: Provides base components that are extended with custom styling. • **Ensures Accessibility**: Built-in accessibility features like proper ARIA attributes. • **Integrates with Tailwind**: Seamlessly works with Tailwind for custom styling. While many components are custom-built, Nuxt UI provides a foundation for common UI elements and interactions.
Nuxt Fonts	Integrated via `@nuxt/fonts` to manage web fonts throughout the application. The module: • **Optimizes Font Loading**: Implements best practices for font loading performance. • **Manages Font Files**: Handles font file formats and browser compatibility. • **Implements Font Display Strategies**: Controls font rendering behavior. Typography is an essential element of the yoga studio's visual identity, and this module ensures consistent, performant font rendering.

Table 2.24: UI Enhancement Modules

The integration of these extra modules significantly enhances the application's capabilities beyond what's available in the core frameworks. Each module was selected to address specific requirements of the yoga studio website, from content management to visual presentation to backend integration, creating a cohesive and performant user experience.

2.7. External Libraries

The LotusHaven project leverages several external libraries and frameworks to enhance development efficiency, application performance, and user experience. These dependencies

provide crucial functionality ranging from core framework capabilities to specialized UI components and development tools.

2.7.1. Core Framework and Runtime

Library	Description
Nuxt (v3.16.2)	Meta-framework for Vue.js that provides server-side rendering, automatic code splitting, static site generation, and other performance optimizations. Serves as the foundation of the application architecture.
Vue (v3.5.13)	Progressive JavaScript framework for building user interfaces, featuring a component-based architecture, reactive data binding, and compositional API. All UI components in the project are built with Vue.
Vue Router (v4.5.0)	Official router for Vue.js applications, providing navigation capabilities and dynamic route matching. Handles all client-side navigation between pages.

Table 2.25: Core Framework Libraries

2.7.2. Nuxt Ecosystem

Library	Description
@nuxt/content (v3.4.0)	File-based CMS for Nuxt that allows reading and displaying content from Markdown, JSON, YAML and CSV files. Used for managing static content such as yoga descriptions and studio information.
@nuxt/image (v1.10.0)	Image optimization and transformation module for Nuxt applications. Provides responsive images, automatic optimization, and format conversion for improved performance and user experience.
@nuxt/ui (v3.0.2)	UI component library for Nuxt that provides pre-built accessible components. Offers a foundation for consistent styling and interactions.
@nuxt/fonts (v0.11.1)	Font management module for Nuxt applications, handling font loading optimization and configuration.
@nuxt/icon (v1.12.0)	Icon management module simplifying the use of SVG icons throughout the application.
@nuxtjs/supabase (v1.5.0)	Official Supabase integration for Nuxt, providing simplified access to authentication, database, and storage features. Manages the connection to backend services.
@unhead/vue (v2.0.7)	Head management for Vue and Nuxt applications, allowing control of document `<head>` metadata for SEO optimization.

Table 2.26: Nuxt Ecosystem Libraries

2.7.3. Development and Styling Tools

Library	Description
TypeScript (v5.8.3)	Superset of JavaScript that adds static types, enhancing code quality, developer experience, and maintainability. Used throughout the project for type safety.
Tailwind CSS (v4.1.4)	Utility-first CSS framework for rapid UI development. Provides low-level utility classes for building custom designs without leaving the HTML. Powers the site's responsive design and styling system.
@tailwindcss/vite (v4.1.4)	Vite plugin for Tailwind CSS, enabling efficient compilation and hot module replacement during development.
ESLint (v9.24.0)	Static code analysis tool for identifying problematic patterns in JavaScript and TypeScript code. Ensures code quality and consistency.
@nuxt/test-utils (v3.17.2)	Testing utilities for Nuxt applications, providing tools for unit and integration testing.

Table 2.27: Development and Styling Tools

2.7.4. Utility Libraries

Library	Description
cookie (v0.7.2)	Basic cookie parsing and serialization library for handling client-side cookies. Used for managing user preferences and session data.

Table 2.28: Utility Libraries

The combination of these libraries creates a robust development environment that prioritizes performance, developer experience, and maintainability. The selection reflects modern web development best practices, with a focus on type safety (TypeScript), component-based architecture (Vue), server-side rendering (Nuxt), and utility-based styling (Tailwind CSS).

3 | Extras

3.1. Extras

In this chapter, we will discuss how our website is compliant with the accessibility and SEO guidelines. In order to evaluate such compliance, we have used the WAVE and Lighthouse web evaluation tools.

3.1.1. Accessibility Implementation

Ensuring web accessibility was a core priority in the development of Lotus Haven's website to provide an inclusive experience for all users regardless of their abilities or disabilities. Our implementation follows the Web Content Accessibility Guidelines (WCAG) standards and has been validated using Lighthouse evaluation tools.

Lighthouse Accessibility Evaluation

According to our most recent Lighthouse report, Lotus Haven achieved an accessibility score of **96%** which demonstrates our strong commitment to accessible design. This high score confirms that the website implements most of the recommended accessibility practices and ensures a smooth experience for users with various disabilities.

Implemented Accessibility Features

Semantic HTML Structure The website uses semantic HTML elements throughout to ensure that content is properly structured and navigable by screen readers and other assistive technologies:

- **Proper heading hierarchy** with logical progression from H1 to lower-level headings
- **Landmark regions** such as navigation, main content, and complementary areas
- **Semantic elements** like `<section>`, `<article>`, and `<nav>` to organize content

Image Accessibility All images throughout the website follow accessibility best practices:

- **Alternative text**: Informative images include descriptive alt text (e.g., "Lotus Haven Logo" for the site logo)
- **Decorative images**: Properly marked with empty alt attributes to prevent unnecessary screen reader announcements
- **SVG accessibility**: Icons include appropriate ARIA attributes when needed for context

Color Contrast and Visual Design The design ensures readability for users with visual impairments:

- **WCAG AA compliant color contrast** between text and background
- **Visual indicators** beyond color alone for interactive elements
- **Consistent visual styling** for similar interactive elements
- **Customized scrollbars** with enhanced visibility for easier navigation

Keyboard Navigation The website is fully navigable without requiring a mouse:

- **Logical tab order** following the visual flow of the page
- **Focus indicators** that are clearly visible when elements receive keyboard focus
- **Skip-to-content** functionality available for keyboard users
- **Interactive carousels** with keyboard controls (arrow navigation)

Responsive Design Considerations Accessibility extends to different viewport sizes and devices:

- **Viewport meta tag** properly configured to support mobile devices and prevent zooming issues
- **Responsive layout** that adapts to different screen sizes
- **Zoom detection** that intelligently adapts the interface when users have zoomed in
- **Touch-friendly targets** of appropriate size for mobile users

ARIA Attributes ARIA (Accessible Rich Internet Applications) attributes are used judiciously to enhance accessibility:

- **ARIA labels** for elements where the visible text is insufficient
- **ARIA roles** to clarify the purpose of custom widgets and components
- **Dynamic content updates** that are announced appropriately to screen readers

Forms and Interactive Elements All interactive elements are designed with accessibility in mind:

- **Properly labeled form controls** with explicit associations between labels and inputs
- **Clear error messages** for form validation
- **Button states** (disabled, active, hover) that are clearly distinguishable
- **Focus management** for modal dialogs and other interactive components

Implementation in Code

Our accessibility features are not just superficial additions but are deeply integrated into the codebase:

- All components include careful consideration of keyboard navigation, screen reader announcements, and visual focus states
- The `ClassesCarousel` and `TeachersCarousel` components implement enhanced keyboard accessibility and mouse wheel interaction for horizontal scrolling
- Viewport meta tags ensure proper display on mobile devices and prevent the 300ms tap delay
- Form components include built-in accessibility features like proper labeling and ARIA attributes
- The UI component library (`@nuxt/ui`) provides a foundation of accessible components that we've extended

Remaining Accessibility Challenges

While our Lighthouse score of 96% is excellent, we acknowledge a few remaining areas for improvement:

- Complex interactive elements like carousels could benefit from additional ARIA live regions for dynamic content updates
- Further enhancements to keyboard navigation in some complex widgets
- Additional testing with various screen readers to ensure consistent experiences

Our commitment to accessibility is ongoing, with regular testing and improvements planned as part of our development roadmap.

3.1.2. SEO Compliance

Search Engine Optimization (SEO) is crucial for ensuring that our website is discoverable by potential users through search engines. We've implemented several SEO strategies and best practices, validated through Lighthouse's SEO evaluation.

Lighthouse SEO Evaluation

The Lighthouse SEO analysis shows that our website follows modern SEO best practices, ensuring good visibility in search engine results. The implementation of proper meta tags, semantic HTML, and mobile-friendly design contributes to the website's search engine ranking potential.

Implemented SEO Features

Meta Tags and Document Structure We've implemented comprehensive meta tags across all pages to improve search engine understanding of our content:

- **Title tags**: Each page has a unique, descriptive title that includes relevant keywords (e.g., "About Us | Our Studio & Philosophy | Lotus Haven")
- **Meta descriptions**: Custom descriptions summarize each page's content and purpose for search engine results
- **Viewport meta tag**: Properly configured to ensure mobile-friendly display with "width=device-width, initial-scale=1"
- **Document `<head>`**: Well-structured with appropriate tags and prioritized loading

Semantic Content Structure Our content is organized with search engines in mind:

- **Proper heading hierarchy**: Logical H1-H6 structure that helps search engines understand content importance

- **Descriptive link text**: Links use meaningful text rather than generic phrases like "click here"
- **Semantic HTML elements**: Content is structured with appropriate semantic tags that communicate meaning
- **Crawlable anchors**: All links are properly formatted to be discoverable by search engine crawlers

Social Media Integration We've enhanced social sharing capabilities through proper metadata:

- **Open Graph Protocol**: Implementation of og tags (og:title, og:description, og:type, og:image) for optimized sharing on platforms like Facebook
- **Twitter Card markup**: Including twitter:card, twitter:title, and twitter:description tags for Twitter sharing optimization
- **Consistent branding**: Social metadata maintains consistent branding and messaging across platforms

Mobile Optimization Our mobile-first approach benefits SEO:

- **Responsive design**: Content adapts seamlessly to various screen sizes
- **Touch-friendly elements**: Proper sizing and spacing of interactive elements
- **Fast loading on mobile**: Performance optimizations for mobile network conditions

Technical SEO Elements Several technical implementations enhance our SEO compliance:

- **HTTPS implementation**: The entire site is served over secure HTTPS connections
- **Canonical URLs**: Properly defined to prevent duplicate content issues
- **Server-side rendering**: Nuxt's SSR capabilities ensure content is immediately available to search engine crawlers
- **robots.txt**: Properly configured to guide search engine crawlers

Implementation in Code

Our SEO features are implemented throughout the codebase:

- **Dynamic meta tags**: Using Nuxt's `useHead` composable to set page-specific metadata based on content
- **Structured data**: Implementing JSON-LD for certain content types to enhance rich results in search engines
- **Image optimization**: Using Nuxt Image module for responsive image sizing and format optimization
- **SEO-friendly URLs**: Clean, descriptive URLs that follow best practices

Example from our `about.vue` page showing dynamic meta tag implementation:

```
useHead({
  title: "About Us | Our Studio & Philosophy | Lotus Haven",
  meta: [
    {
      name: "description",
      content:
        "Learn about Lotus Haven yoga studio, our philosophy, location, and mission.
        Discover how we integrate ancient wisdom with modern practice to create
        a nurturing wellness environment.",
    },
    { name: "viewport", content: "width=device-width, initial-scale=1" },
    { property: "og:title", content: "About Lotus Haven Yoga Studio" },
    {
      property: "og:description",
      content:
        "Learn about our yoga philosophy, studio location and hours at Lotus Haven.",
    },
    { property: "og:type", content: "website" },
  ],
});
```

SEO Monitoring and Future Improvements

While our implementation of SEO best practices is comprehensive, we recognize that SEO is an ongoing process:

- **Regular monitoring**: Tracking search performance through analytics tools
- **Keyword optimization**: Continuing to refine content based on relevant search

terms

- **Performance optimization**: Further improving page load speeds to benefit search rankings
- **Structured data enhancements**: Expanding schema markup for more rich search results

Our approach to SEO is holistic, focusing not just on technical implementation but also on creating high-quality, relevant content that provides value to our users while following search engine guidelines.