AI Nutritionist chatbot

Headless CMS with Next JS and Wordpress

CSIS-4495-050

Nyamkhuu Enkhbat #300388408
Argho Chakma #300367039

11-27-2025

TABLE OF CONTENTS

Table of Figures

1 INTRODUCTION

Happy nutrition provides consultations to improve gut health, digestion, hormone balance, and stress management through personalized diet, targeted supplementation, and transformative lifestyle changes. The main website, www.happynutrition.ca, is currently used for advertising and selling supplements.

There are 2 websites under development. An online learning platform, www.healthacademy.ca, is published and it will be promoted to public soon. However, the client wants changes on the website by improving performance, fixing some bugs, making user-friendly, and more importantly adding AI chatbot that allows users to have informative conversations regarding the course materials and nutrition advice.

There is another web app named AskNutritionist.ai which is not published yet. The amount of work to be done is still unclear to the clint as the work of building the site is still in progress.

2 PROPOSED RESEARCH PROJECT

We proposed four main things that need to be done at the end of term, AI powered Chatbot, Performance optimization, Usability and SEO, and fixes bug and maintenance.

AI powered Chatbot: We proposed a personalized AI-powered health chatbot that helps students find answers about nutrition and course contents. The chatbot remembers the previous chat and answers the questions regarding topics even though it is not a general question. Also, it should be able to easily connect fast, only registered users can see the chatbot so that we can check which users asked about which topics. We proposed OpenAI, since it is very easy, fast, and common in business intelligence.

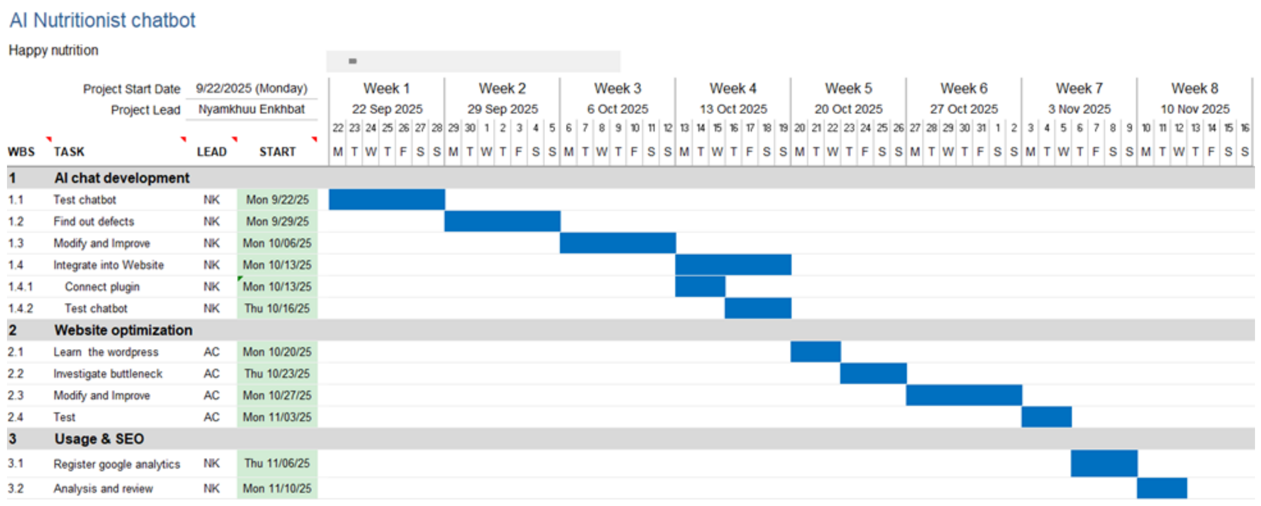
Performance optimization: We proposed NextJS in front end and WordPress will be headless CMS as backend. However, the Riipen client did not like this idea, so we decided to do it anyway and check the performance and show the results to the client after the implementation. Our connection would be GraphQL, and more secure interaction between client and server.

Usability and SEO: Ensuring the website is more user friendly, SEO optimized, responsive, and visually appealing. All images and videos have alt text that describes them. Define the title and description of the header. In the body part, use h1, h2, and h3 sections to describe every page.

Bug fixes: We will ensure that every request from Riipen clients will be done during the session. That can be any request like changing UI in WP, fixing errors, and removing unnecessary plugins etc.

3 PROJECT PLANNING AND TIMELINE

Our schedule to develop was AI chatbot in 4 weeks, Website Optimization 3 weeks, SEO in 1 week, and documentation in 1 week.


AI Nutritionist chatbot

Happy nutrition

Project Start Date 9/22/2025 (Monday)

Project Lead Nyamkhuu Enkhbat

Week 1 22 Sep 2025 | Week 2 29 Sep 2025 | Week 3 6 Oct 2025 | Week 4 13 Oct 2025 | Week 5 20 Oct 2025 | Week 6 27 Oct 2025 | Week 7 3 Nov 2025 | Week 8 10 Nov 2025

WBS	TASK	LEAD	START
1	**AI chat development**		
1.1	Test chatbot	NK	Mon 9/22/25
1.2	Find out defects	NK	Mon 9/29/25
1.3	Modify and Improve	NK	Mon 10/06/25
1.4	Integrate into Website	NK	Mon 10/13/25
1.4.1	Connect plugin	NK	Mon 10/13/25
1.4.2	Test chatbot	NK	Thu 10/16/25
2	**Website optimization**		
2.1	Learn the wordpress	AC	Mon 10/20/25
2.2	Investigate buttleneck	AC	Thu 10/23/25
2.3	Modify and Improve	AC	Mon 10/27/25
2.4	Test	AC	Mon 11/03/25
3	**Usage & SEO**		
3.1	Register google analytics	NK	Thu 11/06/25
3.2	Analysis and review	NK	Mon 11/10/25

Student	Task
Nyamkhuu Enkhbat	Chatbot prototyping, fine-tuning, and integration Website performance analysis & optimization Weekly client meetings and reporting
Argho Chakma	SEO & Analytics setup WordPress optimization and bug fixing Security and compliance testing Research & documentation

However, there were unplanned tasks that I came up with. It was to migrate WP into the development environment. It was so challenging.

4 IMPLEMENTED FEATURE

4.1.1 Summary of Completed Tasks

As part of this project, we successfully completed three major components: the development of automated migration scripts, the implementation of an AI-powered chatbot, and the creation of a redesigned Next.js frontend with improved and modern UI enhancements.

4.1.2 Migration Scripting

A significant portion of the project involved building custom migration scripts capable of transferring a live WordPress server into controlled test environments. These scripts were designed to handle the complexity of a large, rapidly changing WordPress installation and to facilitate consistent replication across multiple hosting platforms. Using these scripts, we were able to migrate the full WordPress instance into both a local hosting environment and an external test server hosted on WHC (www.dataofattraction.com). The migrated site is functional and reflects the original environment accurately; however, it currently experiences intermittent crashes due to resource limitations on the target hosting infrastructure rather than issues with the migration process itself. This confirms that the migration scripts are effective, but the hosting environment must be scaled appropriately to support such a large and dynamic WordPress installation.

4.1.3 AI Chatbot WordPress Plugin

The second major component of the project involved the development and integration of an AI-powered chatbot within the WordPress environment. This feature was implemented as a custom WordPress plugin that communicates directly with the OpenAI API to generate personalized nutrition and health-related responses. The plugin exposes a dedicated REST API endpoint that receives user queries from the frontend, processes those inputs, and constructs prompts suitable for large-language-model interpretation. The server-side PHP code then forwards these prompts to OpenAI and formats the returned output into a JSON response that can be consumed by the Next.js frontend.

This architecture ensures that all AI-related logic, including API keys and context handling, remains securely on the server side, preventing exposure on the client. The chatbot is fully integrated into the user dashboard of the Next.js application, where the React-based chat interface allows users to interact with the model in real time. This implementation demonstrates a practical example of providing users with an intelligent assistant that enhances the educational and wellness capabilities of the platform.

4.1.4 Next.js as the Frontend Layer

In the headless CMS architecture adopted for this project, the Next.js application functions as the complete frontend ecosystem. It serves as the user interface, headless client, application layer, rendering engine, and authentication proxy while also supporting improved SEO performance. In contrast, the existing WordPress installation operates exclusively as the backend, acting as the primary database layer, authentication provider, API provider, and overall content management system. Our implementation of the Next.js frontend strictly follows this headless model.

The Next.js application communicates with WordPress through REST and GraphQL endpoints to retrieve menus, posts, LearnDash courses, events, and user information. It uses server components to render data securely and efficiently, enabling faster load times and enhanced SEO by generating HTML on the server in serverless hosting. Authentication is handled through a hybrid approach in which the Next.js API routes forward login credentials to the WordPress JWT authentication endpoint. Upon successful login, the system stores the token inside an HTTPOnly cookie, ensuring that sensitive authentication data is inaccessible to client-side JavaScript and protected against cross-site scripting attacks.

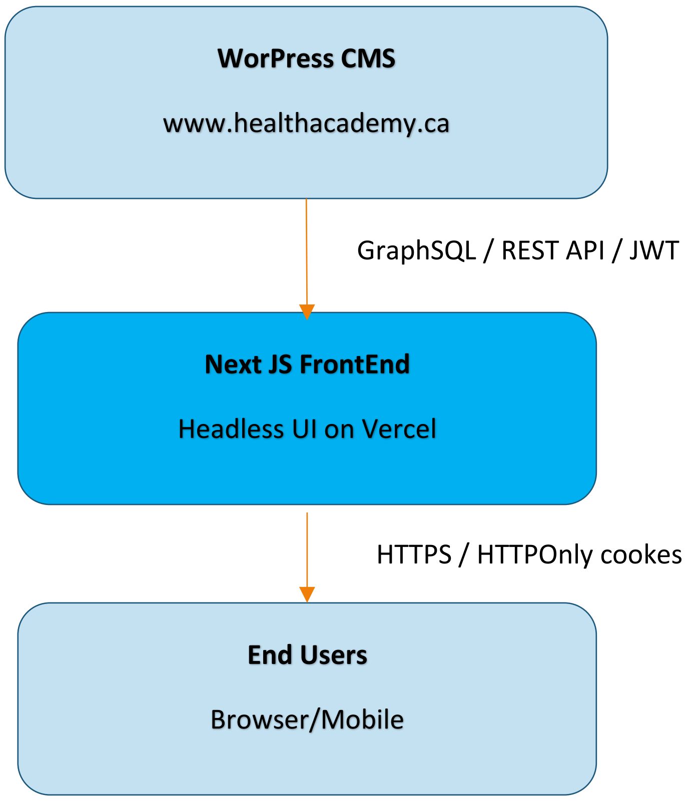


Figure 1(FrontEnd)

4.2 Details of Implementation 2

Overview of the Implemented Feature

This implemented feature transforms the existing WordPress-powered learning platform into a fully decoupled system by introducing a modern Next.js 16 frontend that replaces all public-facing pages. The original WordPress instance continues to store content, users, LearnDash courses, and events, while the new frontend handles routing, rendering, authentication, and interactive functionality. This headless architecture preserves WordPress as the content management system but relocates presentation and user interaction to Next.js, resulting in improved performance, maintainability, and user experience.

4.2.1 System Architecture and Component Structure

The architecture follows a layered model in which the WordPress backend and MySQL database form the content and data layer, while the Next.js application functions as the presentation and control layer. All WordPress content is accessed through REST and GraphQL endpoints and then rendered through Next.js server components. Custom WordPress REST routes allow the frontend to perform operations such as user registration, and LearnDash REST endpoints supply structured course data. A dedicated custom REST namespace handles the nutrition chatbot, which uses OpenAI for generating responses. These elements work together to create a cohesive system where WordPress maintains data integrity while Next.js provides a modernized interface.

4.2.2 Server-Side Layout and Global State Initialization

The global application shell is established in the `app/layout.tsx` file, where server components retrieve the WordPress navigation menu and authenticated user information before rendering. The layout loads global styles, fonts, the navigation bar, and the footer, ensuring a consistent interface across all pages. By retrieving authentication and menu data at the layout level, the system avoids redundant network calls and guarantees that each page loads with the correct contextual information. This central layout also wraps the entire application inside an authentication provider so that client components can access user state at any time.

4.2.3 Page Rendering and Content Assembly

Each page in the app directory loads content by assembling modular React components that correspond to visual sections of the interface. For example, the homepage retrieves a subset of LearnDash courses and passes them to the course-highlight component, while the About page loads timeline sections, health-wheel cards, and animated statistics. By fetching data directly inside server components, the system ensures fast, SEO-friendly performance and reduces API calls from the browser. This modular composition allows individual sections—such as videos, sliders, and testimonials—to be updated independently without affecting the overall page.

4.2.4 Data Integration with WordPress and LearnDash

All data access logic resides in the `lib` directory and interacts with WordPress and LearnDash REST APIs. LearnDash data is retrieved using App Password authentication and delivered as typed objects to the frontend. GraphQL is used to load WordPress menus and restructure them into a navigation tree. User information is retrieved through the WordPress `/users/me` endpoint using JWT tokens stored in secure cookies. This modularized data architecture ensures predictable, type-safe data handling across the application.

4.2.5 Authentication, JWT Workflow, and Protected Routes

Authentication is managed through custom API routes inside the Next.js application. When a user submits their credentials, the `/api/login` endpoint forwards them to WordPress's JWT authentication endpoint. Upon success, the backend stores the returned token inside an HTTP-only cookie, allowing the frontend to stay secure from cross-site scripting attacks. The signup process is handled through a custom WordPress REST route that sanitizes, validates, and registers new accounts. Middleware enforces access restrictions by ensuring users cannot access protected routes such as the dashboard unless authenticated, and that logged-in users cannot revisit the login page unnecessarily.

4.2.6 Dashboard Functionality and Personalized Course Loading

The dashboard represents one of the central implemented features. When a user accesses the dashboard, the application first verifies authentication by reading the JWT token from cookies. It then retrieves LearnDash enrollment and course progress through REST endpoints. These datasets are merged to produce a personalized course overview, which is displayed through structured React components. The dashboard also houses the nutrition chatbot panel, providing users with real-time conversational assistance. User identity, course listings, and chatbot interaction all work together to create an interactive and personalized environment.

4.2.7 Events Calendar Integration and Dynamic Rendering

The events page integrates with The Events Calendar plugin via its REST API. The system retrieves upcoming and past events and passes them into a client-side calendar interface. The calendar supports month, list, and day views, with the `CalendarClient` component managing view state transitions. The `MonthView` component computes calendar geometry, determines weekday offsets, and organizes events into the correct grid positions. As users navigate through months or switch between views, the interface updates dynamically without requiring additional data fetching, delivering a smooth and responsive experience.

4.2.8 Interactive Components and Frontend Enhancements

Interactive UI elements—including sliders, animated counters, accordions, hero banners, and timeline components—are implemented as modular React components. Each section uses TailwindCSS styling and, when needed, CSS Modules to create visually rich experiences. The image slider uses CSS transforms to animate slides horizontally, while animated number components rely on `requestAnimationFrame` to create progressive numerical transitions. These UI enhancements replicate and improve upon the dynamic elements of the original WordPress site while ensuring performance efficiency.

4.2.9 AI Chatbot Integration Using OpenAI

A major aspect of this implementation is the integration of an AI-powered nutrition chatbot inside the dashboard. User messages are captured through a client-side React component and sent to a custom WordPress REST route. WordPress processes the input, constructs a prompt, forwards the request to the OpenAI API, and returns the model's generated response. The frontend updates the chat window by appending both user and assistant messages, creating an interactive conversational interface. This design keeps OpenAI credentials securely stored on the server while enabling seamless communication from the frontend.

4.2.10 Summary of the Implementation Approach

Overall, the implemented feature replaces the traditional WordPress interface with a modern, highly responsive Next.js frontend that interacts with WordPress through secure REST and GraphQL endpoints.

Authentication, course loading, event rendering, chatbot functionality, and UI components are fully integrated into a cohesive system. WordPress continues to manage data and content, while Next.js delivers an optimized, modular, and scalable presentation layer. This implementation demonstrates a complete transformation from a monolithic site to a headless architecture built with contemporary web technologies.

5 Lessons Learned and Future Work

Throughout the development of this applied research project, our team gained extensive hands-on experience in working with modern web technologies and addressing the practical challenges of system integration. One of the most significant lessons involved establishing and maintaining a dependable local development environment while working alongside a live WordPress production site. Configuring XAMPP, resolving MySQL port conflicts, managing phpMyAdmin access, and handling database imports provided us with deeper insight into server operations and database administration in real-world settings.

Implementing Next.js as a headless frontend over WordPress allowed us to better understand contemporary web architectures and the advantages of decoupling content management from presentation. This architectural approach not only improved performance but also demonstrated how modern frontend frameworks can coexist with legacy systems.

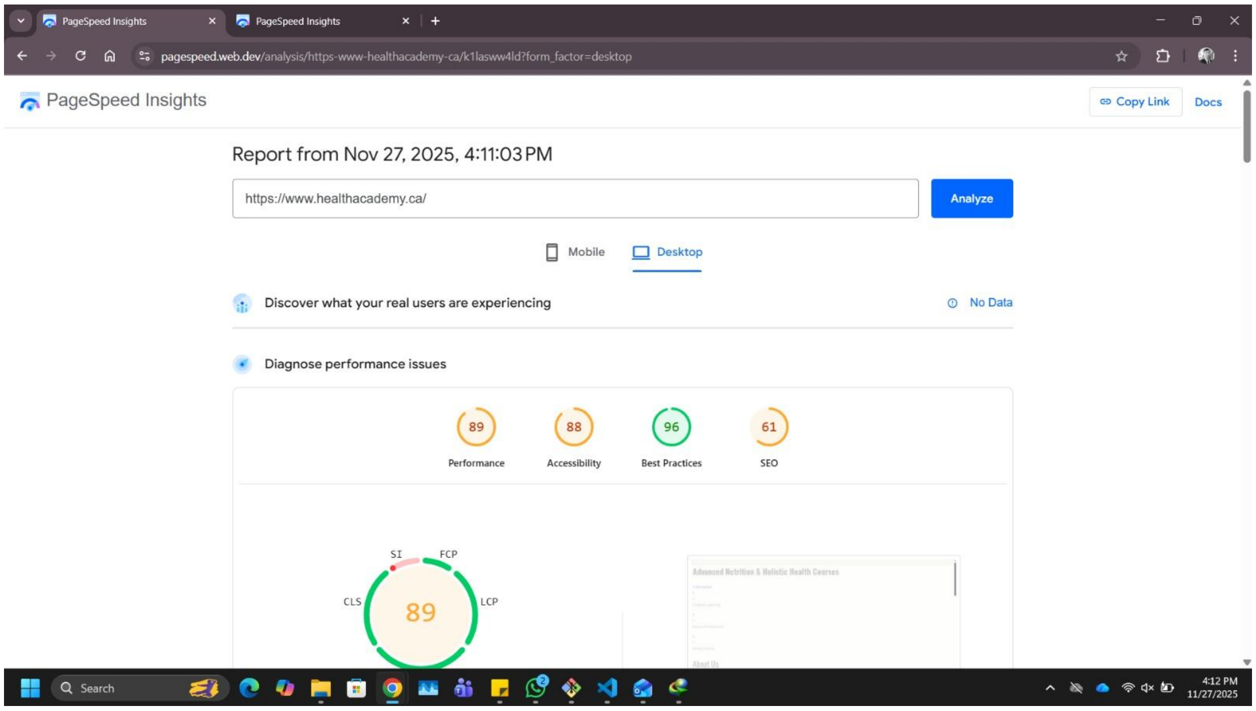
From a project workflow perspective, we realized the importance of disciplined project management practices such as organized version control, structured development procedures, and frequent backups. These habits played a crucial role in maintaining stability throughout the project, minimizing disruptions, and enabling safe experimentation during integration and debugging.

The project also reinforced concepts learned throughout our academic program. Managing and migrating a MySQL database directly connected to our coursework in database administration. Building a fully responsive Next.js interface drew upon skills from front-end development and UI design. Planning, documenting, and iterating on the system reflected core software engineering principles, while managing a local hosting environment related closely to cloud computing and web deployment topics covered in class.

Future development will focus on improving the chatbot's capabilities, strengthening overall site performance, and optimizing search engine visibility through SEO best practices. Additionally, once the WordPress Next.js hybrid architecture becomes fully stable, we plan to migrate the system to a cloud-based environment with continuous deployment pipelines. This transition will support better scalability, streamlined updates, and a more robust long-term operational workflow.

6 Concluding Remarks

So far, the Health Academy project has been an invaluable experiential learning opportunity that allowed our team to bridge theoretical concepts with real-world implementation. Migrating the live WordPress website into a controlled local environment exposed us to the complexities of database administration, environment configuration, and hosting limitations. Integrating the Next.js frontend taught us how a modern headless CMS architecture operates in practice, highlighting the importance of performance optimization, structured development workflows, and API-driven systems. This headless transition not only enhanced our understanding of web engineering but also demonstrated significant performance improvements, as seen in the PageSpeed Insights results: the Next.js version of the site achieved near-perfect scores across Performance, Accessibility, Best Practices, and SEO, vastly outperforming the original WordPress implementation. These metrics validate the technical decisions we made and confirm that our redesigned interface successfully delivers a faster, more accessible, and more search-optimized user experience.



PageSpeed Insights
pagespeed.web.dev/analysis/https-www-healthacademy-ca/k1lasww4ld?form_factor=desktop
Copy Link
Docs
Report from Nov 27, 2025, 4:11:03 PM
https://www.healthacademy.ca/
Analyze
Mobile
Desktop
Discover what your real users are experiencing
No Data
Diagnose performance issues
89
Performance
88
Accessibility
96
Best Practices
61
SEO
SI
FCP
CLS
LCP
89
4:12 PM
11/27/2025

Figure 2(Live Site)

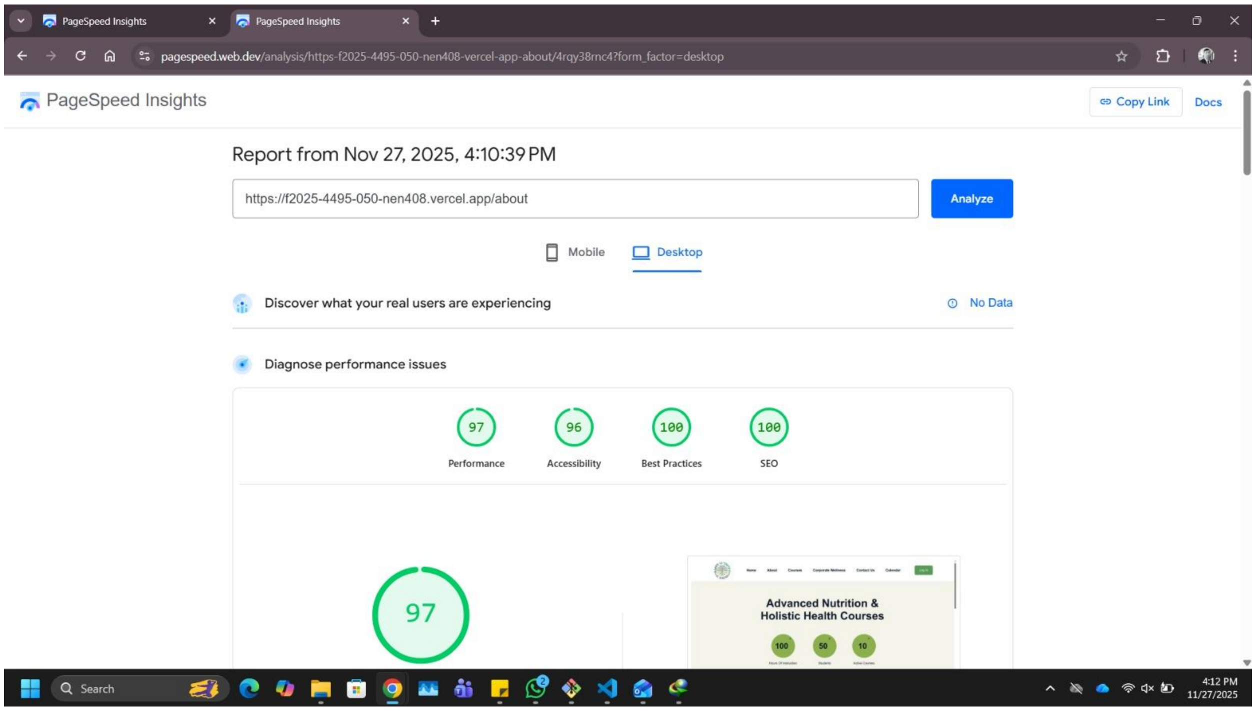

PageSpeed Insights
pagespeed.web.dev/analysis/https-f2025-4495-050-nen408-vercel-app-about/4rqy38rnc4?form_factor=desktop
Copy Link
Docs
Report from Nov 27, 2025, 4:10:39 PM
https://f2025-4495-050-nen408.vercel.app/about
Analyze
Mobile
Desktop
Discover what your real users are experiencing
No Data
Diagnose performance issues
97
Performance
96
Accessibility
100
Best Practices
100
SEO
97
Advanced Nutrition & Holistic Health Courses
100
50
10

Figure 3(Developed Site)

Furthermore, the successful development of the AI-powered chatbot using OpenAI's API marks a major milestone in expanding the platform's capabilities. It introduces an intelligent, personalized information layer that supports users directly from within the dashboard. This confirms that we not only modernized the frontend but also enhanced the platform with meaningful, future-oriented features.

Overall, we have successfully achieved the goals we set at the beginning of the project: migrating the system safely, modernizing the frontend architecture, improving performance and SEO, and integrating an intelligent chatbot feature. As we move forward, the lessons learned will continue to guide us in refining the platform, scaling it to cloud environments, and building even more innovative, user-centered digital solutions.

7 Appendix

7.1 Appendix A: Installation Guide

Install xamp and mysql.

Install node 24.11.1 or later version

In bash terminal:

git clone https://github.com/narkmn/F2025_4495_050_Nen408.git

7.1.1 WordPress CMS (Backend) installation guide

cp r F2025_4495_050_Nen C:\xampp\htdocs

Run xampp and start Apache and Mysql

Go to http://localhost/phpmyadmin/

Import database (The database is not provided due privacy, also WP files are not fully uploaded into github due to storage size)

Go to http://localhost/healthacademy

To migration to webserver please read wordpress documentation.

https://developer.wordpress.org/advanced-administration/upgrade/migrating/

7.1.2 Next.js Frontend

After cloning the repository

cd F2025_4495_050_Nen408/Implementation/wpnextjs/

To test: **npm run dev**

To build: **npm run built**

To start: **npm run start**

.env: (due to purpose of security, it is not uploaded in Github)

```
# WordPress GraphQL Endpoint
WORDPRESS_URL=https://healthacademy.ca/graphql

# WordPress API Credentials
WP_USER="nyamkhuu.en"
WP_APP_PASSWORD="Rmv2 s6xI KlNu 45dR c53L Bm07"
```

```
WP_API_URL="https://healthacademy.ca"

JWT_SECRET=cc470e540913174915cc3692bc8576e1431d3541f089a9cd5ab73844ccd7d6246080a9f1f9b5b6206d8ce744b17dd5e07d546c1f6dd4a929873e97008583efd4
```
to deploy on Vercel, please read documentation.

https://vercel.com/docs/deployments

7.1.3 Test deployments website

www.healthacademy.ca

https://f2025-4495-050-nen408.vercel.app/

7.2 APPENDIX B: USER GUIDE

This section provides a clear and accessible overview of how users interact with the system from the moment they arrive on the website to the point where they explore courses, register an account, access their dashboard, and use the integrated features such as the events calendar and the AI-assisted nutrition chatbot. The user journey begins on the homepage, where visitors are greeted with key sections including the hero banner, introductory information, featured videos, course highlights, application forms, and student testimonials. These sections are designed to offer an immediate sense of the academy's mission, available programs, and student experiences. From this main page, users can navigate to other areas of the website using the global navigation bar, which remains consistent throughout the platform.

To explore educational offerings, users can visit the course listing page, which presents all available LearnDash courses with their titles, images, and short descriptions. By clicking on a course card, users may view additional details provided by the WordPress backend. Users may also browse the About page for information about the academy's background, values, and structure, and access the Corporate Wellness and Contact pages for specific programs or inquiries. The Events page presents upcoming and past events in calendar or list format, allowing users to interact with dates, switch between month, list, and day views, and explore event details fetched directly from The Events Calendar plugin.

If users wish to create an account, they can proceed to the login page, which provides tabs for both login and signup. During signup, users enter a username, email, and password; the system validates these fields and registers the new account through a custom WordPress REST API. Once successfully registered, users may log in by entering their credentials into the login form, after which the system authenticates them using WordPress's JWT authentication mechanism. Upon successful login, users are redirected to their personalized dashboard.

Within the dashboard, users can view their enrolled courses and track progress retrieved in real time from LearnDash. Each course entry displays essential information, allowing users to easily navigate to their content. The dashboard also contains an embedded nutrition chatbot positioned in the lower-right corner of the interface. By clicking the chat button, users activate a conversational panel where they can ask questions related to health, food, or nutrition. The chatbot processes messages using the OpenAI API and returns helpful responses while maintaining a history of the conversation for the duration of the session.

Throughout the website, the navigation bar dynamically adapts based on whether the user is logged in or not. Authenticated users see a “Dashboard” link, while unauthenticated users see a “Log In” button. A logout option is also available, which clears the user’s authentication cookie and redirects them to the homepage. The footer at the bottom of each page provides quick access to essential pages such as About, Contact, Privacy Policy, Terms, and other resources.

Overall, the user interface is designed to remain intuitive, responsive, and accessible across devices. Every major function course browsing, user login, event viewing, and chatbot interaction—is integrated seamlessly into a clean and modern design powered by Next.js. This user guide ensures that both new and returning users can effortlessly navigate the site and take full advantage of its educational and interactive capabilities.

7.3 Appendix C: Dataset and API Used

The data foundation of this project is built entirely on the WordPress content management system, which stores all site content—including posts, pages, course information, media files, user accounts, and configuration settings—in a MySQL relational database. This structured data source serves as the primary dataset for the custom Next.js interface, allowing the project to reuse existing content rather than relying on external datasets. Through its relational schema, WordPress provides consistent and well-organized access to information, making it a reliable backend for both content delivery and user management features.

To access and manipulate this data, the project extensively uses the WordPress REST API, which enables programmatic retrieval of posts, pages, media, menus, and user-related information through HTTP-based endpoints. The REST API plays a key role in bridging the WordPress backend and the modern Next.js frontend, allowing dynamic rendering of site content and seamless synchronization between the two layers. In addition to the standard REST endpoints provided by WordPress, the project incorporates several **custom-built API routes** to support specific functionalities that are not available in the default WordPress API. For example, custom endpoints were created to handle user registration, form submissions, and other workflow-specific operations. These endpoints are implemented in the WordPress theme or plugin directory using the `register_rest_route()` function, allowing server-side logic to be executed securely and efficiently.

One such example is the custom user registration API, which validates input fields, checks for existing usernames or emails, and creates new WordPress users through the `wp_create_user` function. This API is exposed under a custom namespace and allows the frontend to support user onboarding while maintaining full compatibility with WordPress’s authentication and user-management systems. Beyond these custom API implementations, the project also integrates OpenAI’s cloud API to generate intelligent responses for the chatbot component. The OpenAI API is used purely for conversational processing and does not interact with the WordPress database directly. Together, the WordPress MySQL dataset, built-in REST API, custom REST endpoints, and OpenAI integration form the complete data and API ecosystem that powers the project's dynamic functionality and AI-driven features.

7.4 Appendix D: Hardware, Software, Cloud, Architecture, etc.

7.4.1 Hardware Environment

The development and execution of this project were supported by a contemporary personal computing environment running Windows 11. A machine equipped with a multi-core processor and a minimum of 8 GB of RAM provided sufficient computational capability for concurrently running a local WordPress instance, a Node.js backend server, and a Next.js development environment. The system also required adequate storage to maintain project dependencies, such as the `node_modules` directory, SQL databases, and various project assets. This hardware configuration ensured that code compilation, API testing, and local development workflows could be executed efficiently without performance bottlenecks.

7.4.2 Software Stack and Development Environment

The software design follows a decoupled, headless architecture that separates content management, backend logic, and presentation layers. The frontend is developed using Next.js 16 with React 19 and TypeScript 5, leveraging the App Router for optimized server-side rendering, component modularity, and improved routing capabilities. Supporting technologies such as TailwindCSS, PostCSS, and a strict TypeScript configuration contribute to consistent styling, reliable build processes, and code quality enforcement. Development utilities including ESLint and modern TypeScript tooling further enhance maintainability and ensure adherence to software engineering best practices. On the backend, Node.js and Express form the primary application framework responsible for handling API requests, environment variable management, and data persistence via a local SQLite database.

7.4.3 Cloud and AI Infrastructure

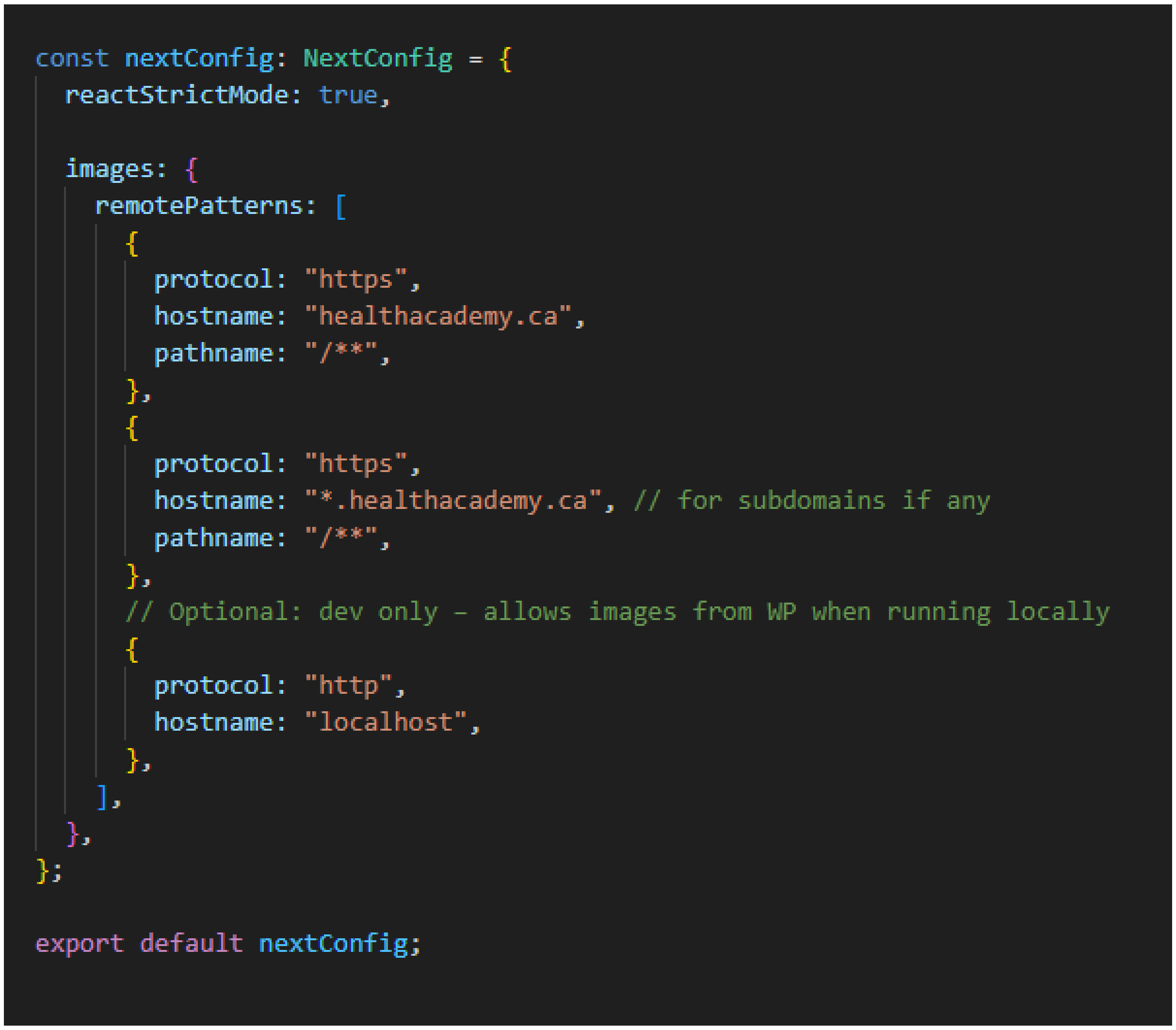
The cloud and artificial intelligence capabilities of this system are powered entirely by OpenAI's API ecosystem, which provides the computational foundation for natural-language understanding, text generation, and semantic processing. Instead of relying on edge-based execution, the project communicates directly with OpenAI's cloud models through secure API requests made from the Node.js backend. Large language models such as OpenAI's GPT-series are utilized to generate conversational responses, interpret user queries, and support knowledge-based interactions. When semantic retrieval or contextual continuity is required, embeddings generated through OpenAI's embedding models enable the system to transform text into high-dimensional vector representations suitable for similarity matching and contextual search. All API keys, model identifiers, and runtime configurations are managed through environment variables within the Node.js microservice, ensuring secure handling of credentials and consistent integration across system components. This cloud-based architecture enables scalable, high-performance AI operations without the need for maintaining local machine-learning infrastructure, while preserving flexibility for future model upgrades or expanded dataset integration.

7.5 Appendix E: Code Explanation

7.5.1 Project Configuration and Global Setup

The structure of the application is defined through core configuration files that specify dependencies, TypeScript behavior, image handling rules, and global styling conventions. The `package.json` file declares Next.js, React, TypeScript, TailwindCSS, GraphQL utilities, and supporting libraries required by the project. It ensures that both development tools and runtime dependencies remain consistent across environments. The `tsconfig.json` file enforces strict type checking, JSX support, and alias imports, allowing the project to maintain clean, scalable code. The `next.config.ts` file defines how external media assets should be handled by the Next.js image optimization pipeline. Since the application fetches images directly from the WordPress backend, explicitly declaring trusted hosts is mandatory for security and optimization. The global stylesheet integrates Tailwind CSS v4 and sets CSS custom properties for fonts and colors, forming a consistent baseline for design across all pages and components.

Example from `next.config.ts`:



```
const nextConfig: NextConfig = {
  reactStrictMode: true,

  images: {
    remotePatterns: [
      {
        protocol: "https",
        hostname: "healthacademy.ca",
        pathname: "/**",
      },
      {
        protocol: "https",
        hostname: "*.healthacademy.ca", // for subdomains if any
        pathname: "/**",
      },
      // Optional: dev only – allows images from WP when running locally
      {
        protocol: "http",
        hostname: "localhost",
      },
    ],
  },
};

export default nextConfig;
```

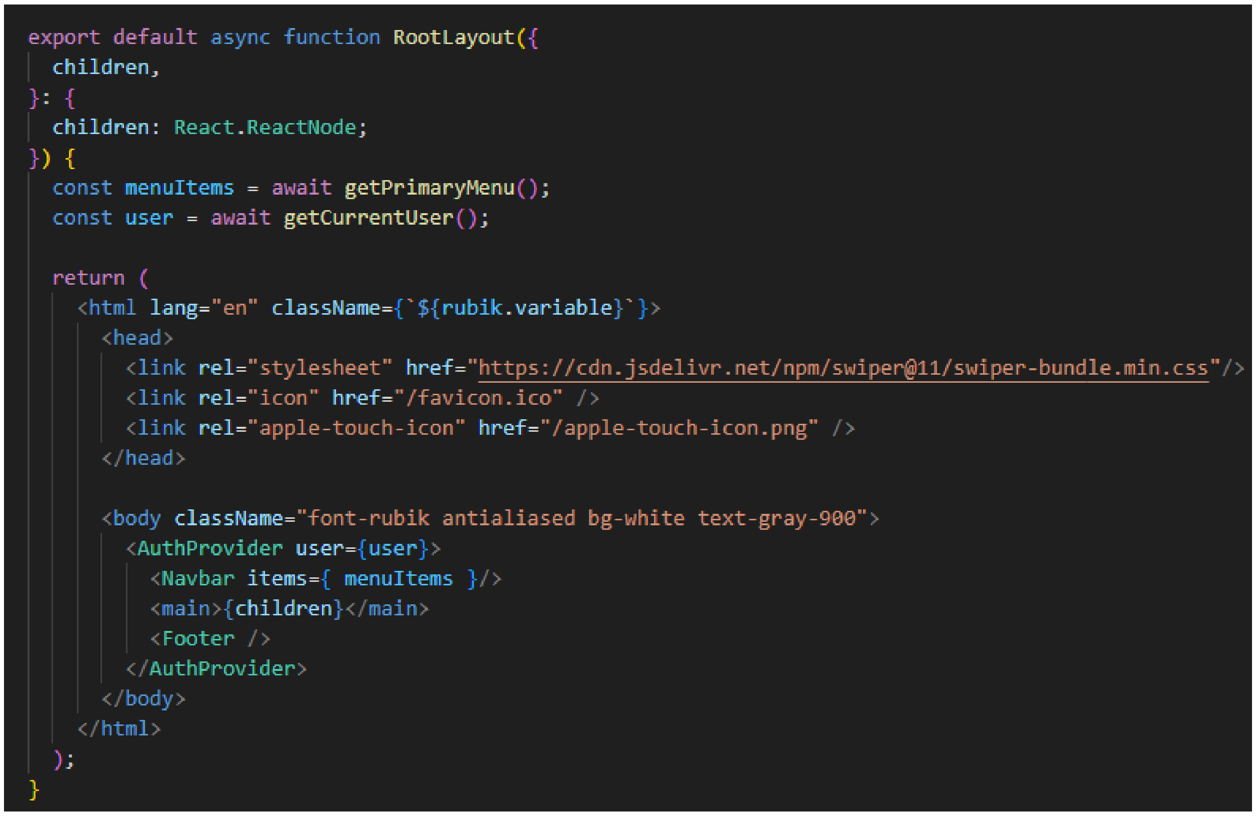
Figure 4(next.config.ts)

This configuration defines two trusted sources for media: the production WordPress site and the local development site. By specifying `remotePatterns`, Next.js allows images from these domains to pass through its built-in image optimization system, which improves loading performance while preventing unauthorized domains from being used. This ensures that all WordPress-hosted images used in hero banners, course cards, and blog content can render safely and responsively.

7.5.2 Application Layout and Shared Components

The overarching layout for the project resides in `app/layout.tsx`, where the Rubik font is loaded, global styles are applied, and WordPress menu and user information are retrieved on the server before rendering. Because this layout is wrapped around every page, it serves as the structural foundation of the entire application. The layout fetches the WordPress menu and authenticated user data using server components, a feature enabled by Next.js 16. This data is then injected into an authentication provider that exposes it to all client components. The use of a shared Navbar and Footer ensures that navigation remains consistent throughout the application. By fetching user data at the layout level, the system avoids redundant network requests and establishes user identity early in the render pipeline.

Example from app/`layout.tsx`:



```
export default async function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  const menuItems = await getPrimaryMenu();
  const user = await getCurrentUser();

  return (
    <html lang="en" className={`${rubik.variable}`}>
      <head>
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/swiper@11/swiper-bundle.min.css"/>
        <link rel="icon" href="/favicon.ico" />
        <link rel="apple-touch-icon" href="/apple-touch-icon.png" />
      </head>

      <body className="font-rubik antialiased bg-white text-gray-900">
        <AuthProvider user={user}>
          <Navbar items={ menuItems }/>
          <main>{children}</main>
          <Footer />
        </AuthProvider>
      </body>
    </html>
  );
}
```

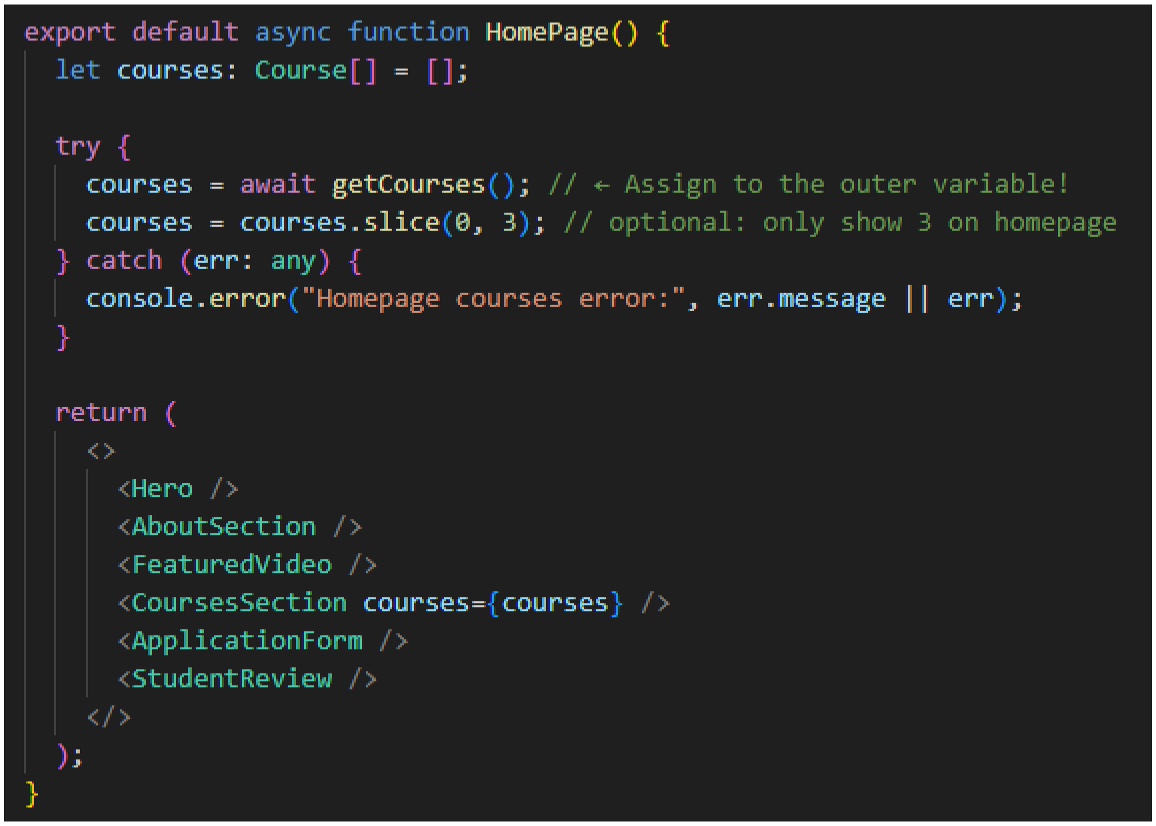
Figure 5(layout.tsx)

This example demonstrates how server-side logic is integrated inside a React component in the App Router. The `getPrimaryMenu` function retrieves the WordPress menu structure through a GraphQL request, while `getCurrentUser` checks for a JWT token stored in HTTP-only cookies and verifies it against the WordPress REST API. Wrapping the layout with an `AuthProvider` allows client components such as the Navbar to determine whether the user is logged in and personalize the UI accordingly. This prevents unauthorized access to protected routes and ensures reliable state propagation.

7.5.3 Page Structure and Content Rendering

The pages in the app directory assemble content from modular components that correspond to sections of the website. The home page loads its visual sections such as the hero banner, about summary, featured video, and student testimonials. It also retrieves LearnDash course data from the backend using the data layer, selects a subset of three courses to feature on the homepage, and passes them into a dedicated component. Using server components allows Next.js to fetch data during the initial page load, improving both SEO performance and user experience.

Example from app/`page.tsx`:



```tsx
export default async function HomePage() {
  let courses: Course[] = [];

  try {
    courses = await getCourses(); // ← Assign to the outer variable!
    courses = courses.slice(0, 3); // optional: only show 3 on homepage
  } catch (err: any) {
    console.error("Homepage courses error:", err.message || err);
  }

  return (
    <>
      <Hero />
      <AboutSection />
      <FeaturedVideo />
      <CoursesSection courses={courses} />
      <ApplicationForm />
      <StudentReview />
    </>
  );
}
```

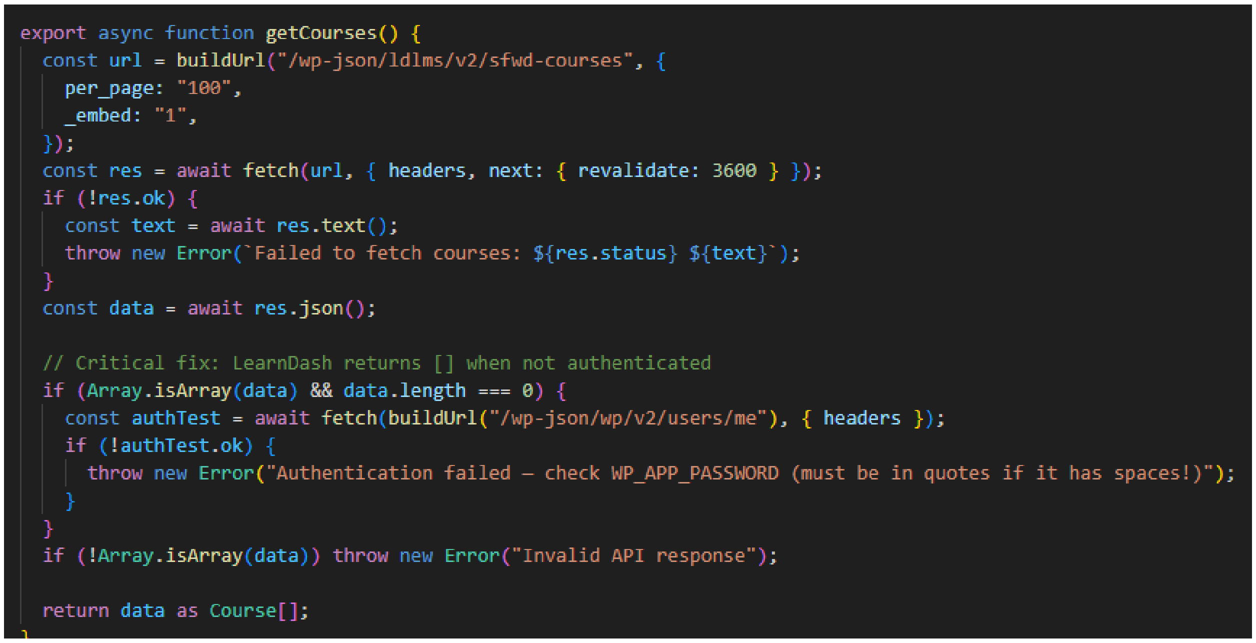
Figure 6(page.tsx)

This code uses asynchronous server-side rendering to fetch course data from LearnDash before the page renders. The slicing operation ensures that only a curated selection of courses is displayed, preventing the homepage from becoming visually overwhelming. The use of individual components for each section makes the code easier to maintain, as stylistic or structural changes can be performed in isolation without affecting the surrounding page.

7.5.4 Data Layer and WordPress / LearnDash Integration

The data layer encapsulates all communication with the WordPress backend. It handles REST requests, authentication, and response normalization, ensuring that the frontend receives predictable and typed data structures. API requests to LearnDash are protected using WordPress App Passwords, which serve as a secure authentication mechanism. The LearnDash API exposes course, lesson, and topic data, allowing the frontend to dynamically generate course pages and dashboards without relying on manual data entry.

Example from `lib/learnDash.ts`:



```
export async function getCourses() {
  const url = buildUrl("/wp-json/ldlms/v2/sfwd-courses", {
    per_page: "100",
    _embed: "1",
  });
  const res = await fetch(url, { headers, next: { revalidate: 3600 } });
  if (!res.ok) {
    const text = await res.text();
    throw new Error(`Failed to fetch courses: ${res.status} ${text}`);
  }
  const data = await res.json();

  // Critical fix: LearnDash returns [] when not authenticated
  if (Array.isArray(data) && data.length === 0) {
    const authTest = await fetch(buildUrl("/wp-json/wp/v2/users/me"), { headers });
    if (!authTest.ok) {
      throw new Error("Authentication failed — check WP_APP_PASSWORD (must be in quotes if it has spaces!)");
    }
  }
  if (!Array.isArray(data)) throw new Error("Invalid API response");

  return data as Course[];
```

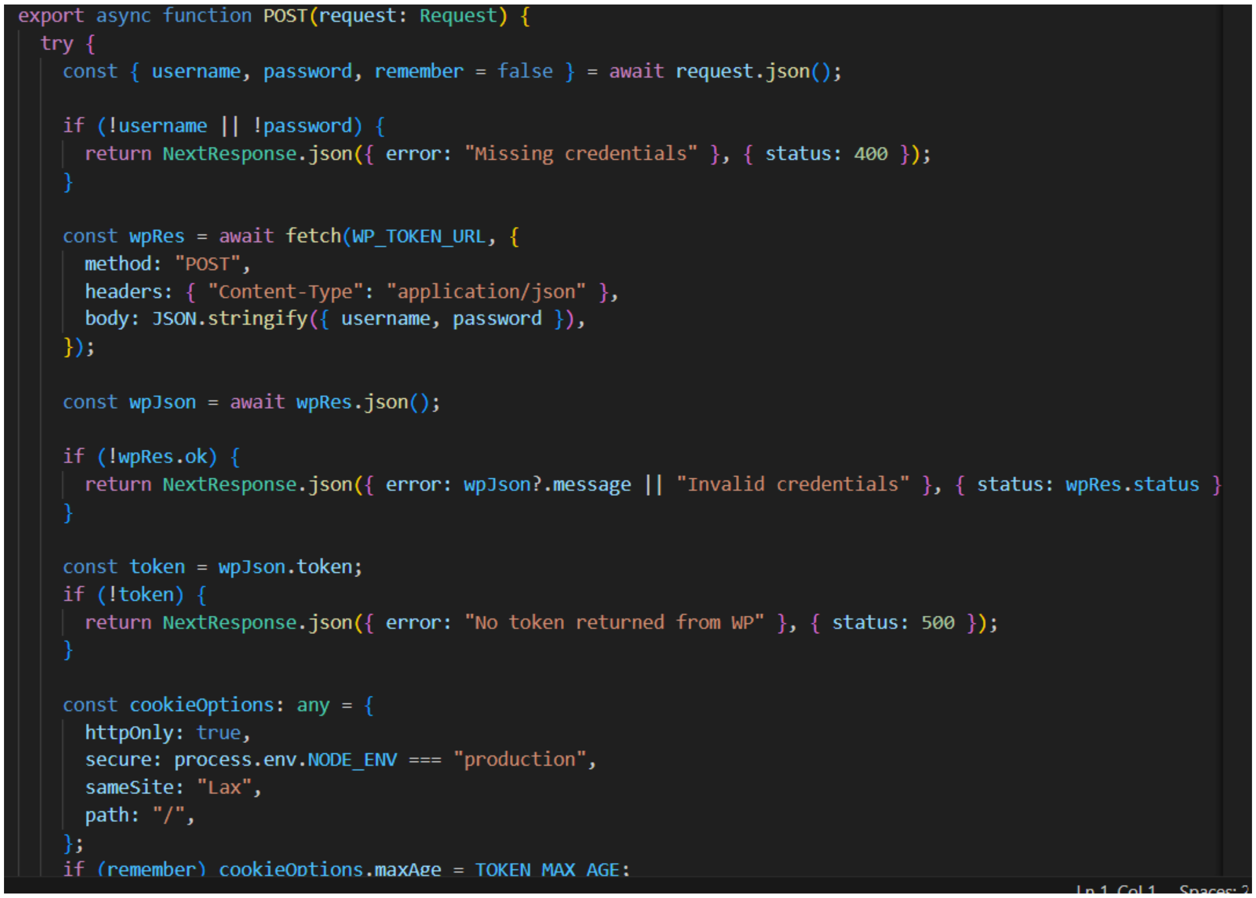
Figure 7(learnDash.ts)

This function constructs an authenticated request to the LearnDash API using environment variables for credentials. The Base64 conversion ensures that the username and app-password pair are securely encoded. Once the JSON response is parsed, it provides an array of course objects containing metadata such as titles, excerpts, progress details, and media IDs. By centralizing this logic, the project ensures consistent data handling and reduces redundancy across different components.

7.5.5 Authentication API Routes and Middleware

Authentication is implemented using a dedicated set of API routes that interact directly with WordPress. The login route forwards user credentials to the WordPress JWT Authentication plugin, which validates the credentials and returns a signed JWT token. The Next.js API route then persists this token inside an HTTP-only cookie, protecting it from client-side JavaScript access and preventing XSS attacks. This cookie is subsequently used by server components and middleware to authenticate users when they attempt to access protected pages.

Example from `app/api/login/route.ts`:



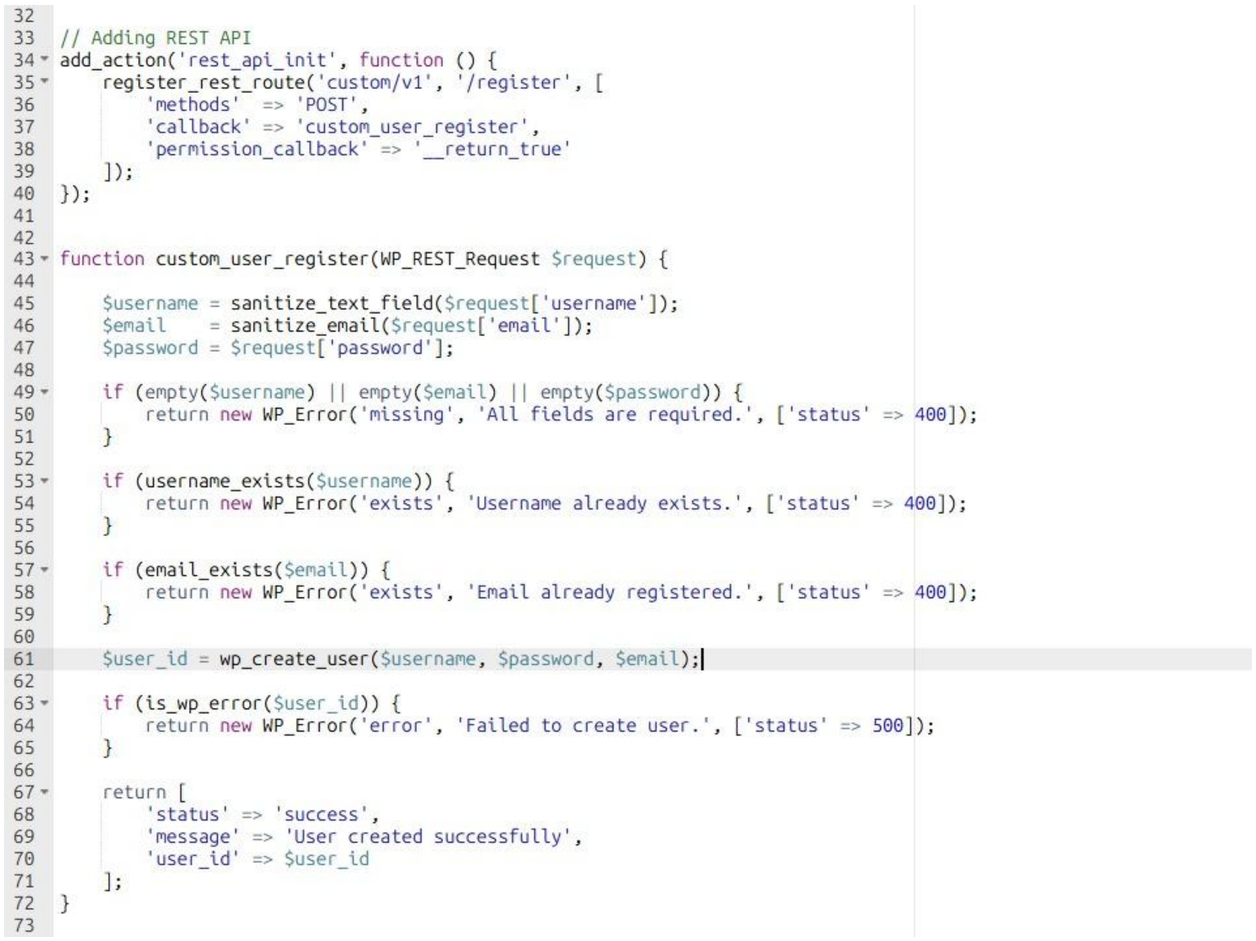
```
export async function POST(request: Request) {
  try {
    const { username, password, remember = false } = await request.json();

    if (!username || !password) {
      return NextResponse.json({ error: "Missing credentials" }, { status: 400 });
    }

    const wpRes = await fetch(WP_TOKEN_URL, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ username, password }),
    });

    const wpJson = await wpRes.json();

    if (!wpRes.ok) {
      return NextResponse.json({ error: wpJson?.message || "Invalid credentials" }, { status: wpRes.status }
    }

    const token = wpJson.token;
    if (!token) {
      return NextResponse.json({ error: "No token returned from WP" }, { status: 500 });
    }

    const cookieOptions: any = {
      httpOnly: true,
      secure: process.env.NODE_ENV === "production",
      sameSite: "Lax",
      path: "/",
    };
    if (remember) cookieOptions.maxAge = TOKEN_MAX_AGE;
```

Figure 8(route.ts)

This approach ensures that the entire authentication process is handled server-side. The client never has access to the raw token, which strengthens security. The use of `cookies().set` allows Next.js to manage authentication consistently across both server and client components.

The signup route interacts with a custom WordPress REST endpoint that registers new users.

Example from your custom WordPress REST API:



```php

// Adding REST API
add_action('rest_api_init', function () {
    register_rest_route('custom/v1', '/register', [
        'methods'  => 'POST',
        'callback' => 'custom_user_register',
        'permission_callback' => '__return_true'
    ]);
});


function custom_user_register(WP_REST_Request $request) {

    $username = sanitize_text_field($request['username']);
    $email    = sanitize_email($request['email']);
    $password = $request['password'];

    if (empty($username) || empty($email) || empty($password)) {
        return new WP_Error('missing', 'All fields are required.', ['status' => 400]);
    }

    if (username_exists($username)) {
        return new WP_Error('exists', 'Username already exists.', ['status' => 400]);
    }

    if (email_exists($email)) {
        return new WP_Error('exists', 'Email already registered.', ['status' => 400]);
    }

    $user_id = wp_create_user($username, $password, $email);

    if (is_wp_error($user_id)) {
        return new WP_Error('error', 'Failed to create user.', ['status' => 500]);
    }

    return [
        'status' => 'success',
        'message' => 'User created successfully',
        'user_id' => $user_id
    ];
}

```

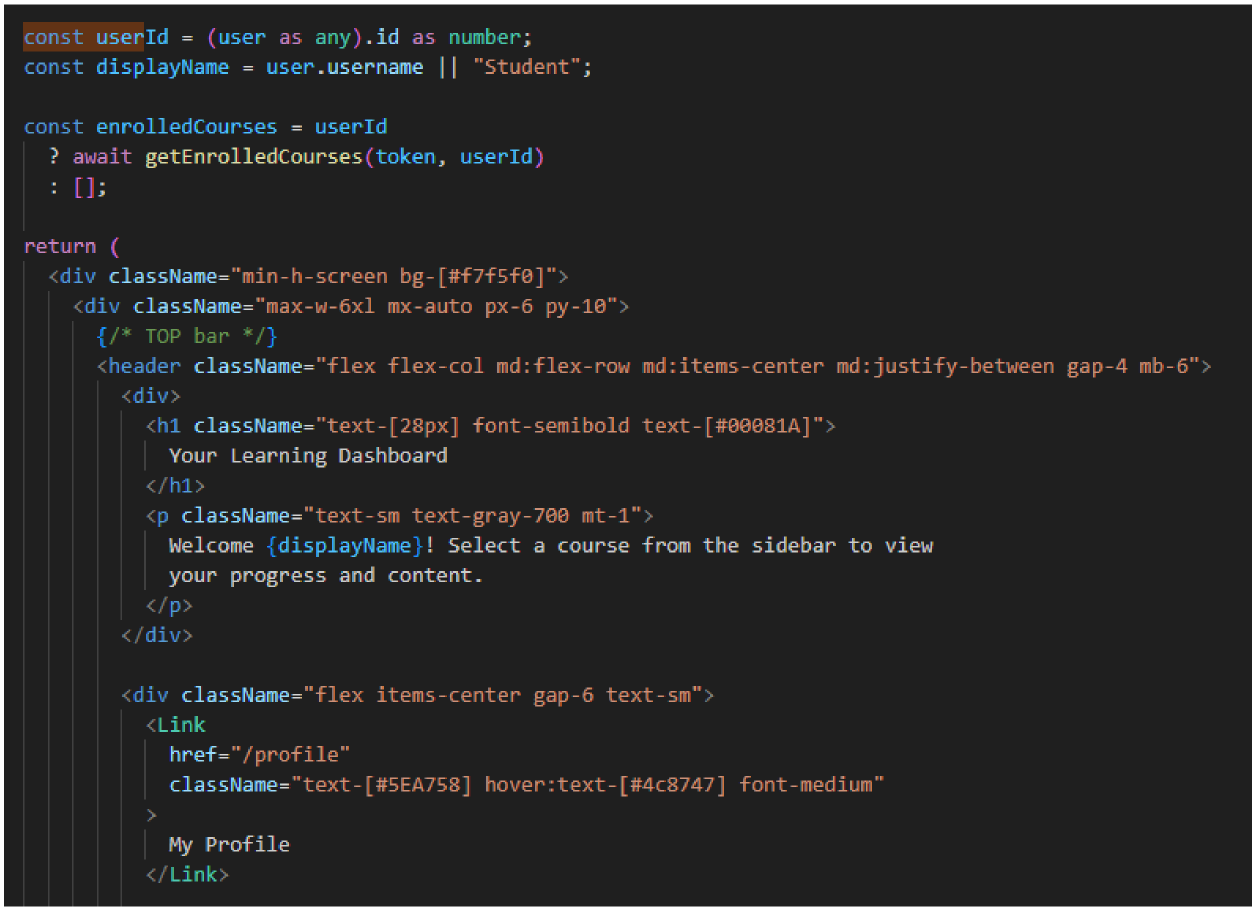
Figure 9(Custom API)

This logic ensures that only valid data is accepted. Fields are sanitized, existing accounts are checked, and new accounts are stored securely in the MySQL database. The response is structured in JSON so the Next.js frontend can easily parse status messages.

7.5.6 Dashboard, LearnDash Progress, and User Data Handling

The dashboard retrieves both user identity and course enrollment data. Access to the dashboard is restricted through both middleware and server-side checks. Once authenticated, the application makes a series of LearnDash API calls to load the user’s enrolled courses and progress metrics. This creates a personalized dashboard that updates dynamically based on user activity.

Example from `app/dashboard/page.tsx`:



```
const userId = (user as any).id as number;
const displayName = user.username || "Student";

const enrolledCourses = userId
  ? await getEnrolledCourses(token, userId)
  : [];

return (
  <div className="min-h-screen bg-[#f7f5f0]">
    <div className="max-w-6xl mx-auto px-6 py-10">
      {/* TOP bar */}
      <header className="flex flex-col md:flex-row md:items-center md:justify-between gap-4 mb-6">
        <div>
          <h1 className="text-[28px] font-semibold text-[#00081A]">
            Your Learning Dashboard
          </h1>
          <p className="text-sm text-gray-700 mt-1">
            Welcome {displayName}! Select a course from the sidebar to view
            your progress and content.
          </p>
        </div>

        <div className="flex items-center gap-6 text-sm">
          <Link
            href="/profile"
            className="text-[#5EA758] hover:text-[#4c8747] font-medium"
          >
            My Profile
          </Link>
```

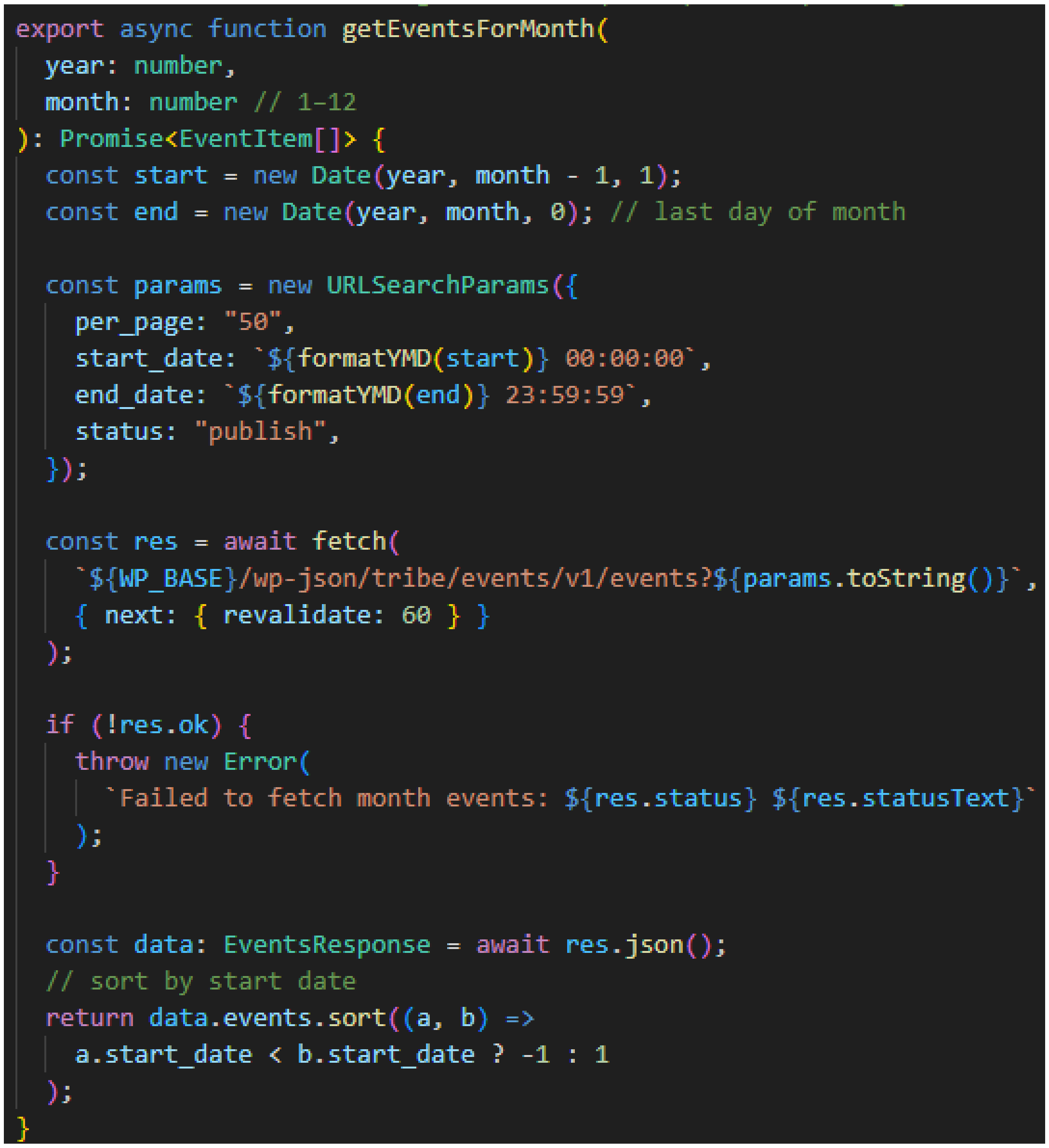
Figure 10(page.tsx)

This server component guarantees that only authenticated users can access their dashboard. The use of `redirect` prevents unauthorized rendering, and fetching LearnDash data at the server level ensures that the dashboard loads quickly and accurately.

7.5.7 Events System and Calendar Interface

The events system integrates with The Events Calendar plugin, allowing event data to be retrieved and displayed inside a custom React calendar component. By using the plugin's REST API, the system avoids manual data entry and stays synchronized with WordPress.

Example from `lib/events.ts`:



```
export async function getEventsForMonth(
  year: number,
  month: number // 1-12
): Promise<EventItem[]> {
  const start = new Date(year, month - 1, 1);
  const end = new Date(year, month, 0); // last day of month

  const params = new URLSearchParams({
    per_page: "50",
    start_date: `${formatYMD(start)} 00:00:00`,
    end_date: `${formatYMD(end)} 23:59:59`,
    status: "publish",
  });

  const res = await fetch(
    `${WP_BASE}/wp-json/tribe/events/v1/events?${params.toString()}`,
    { next: { revalidate: 60 } }
  );

  if (!res.ok) {
    throw new Error(
      `Failed to fetch month events: ${res.status} ${res.statusText}`
    );
  }

  const data: EventsResponse = await res.json();
  // sort by start date
  return data.events.sort((a, b) =>
    a.start_date < b.start_date ? -1 : 1
  );
}
```

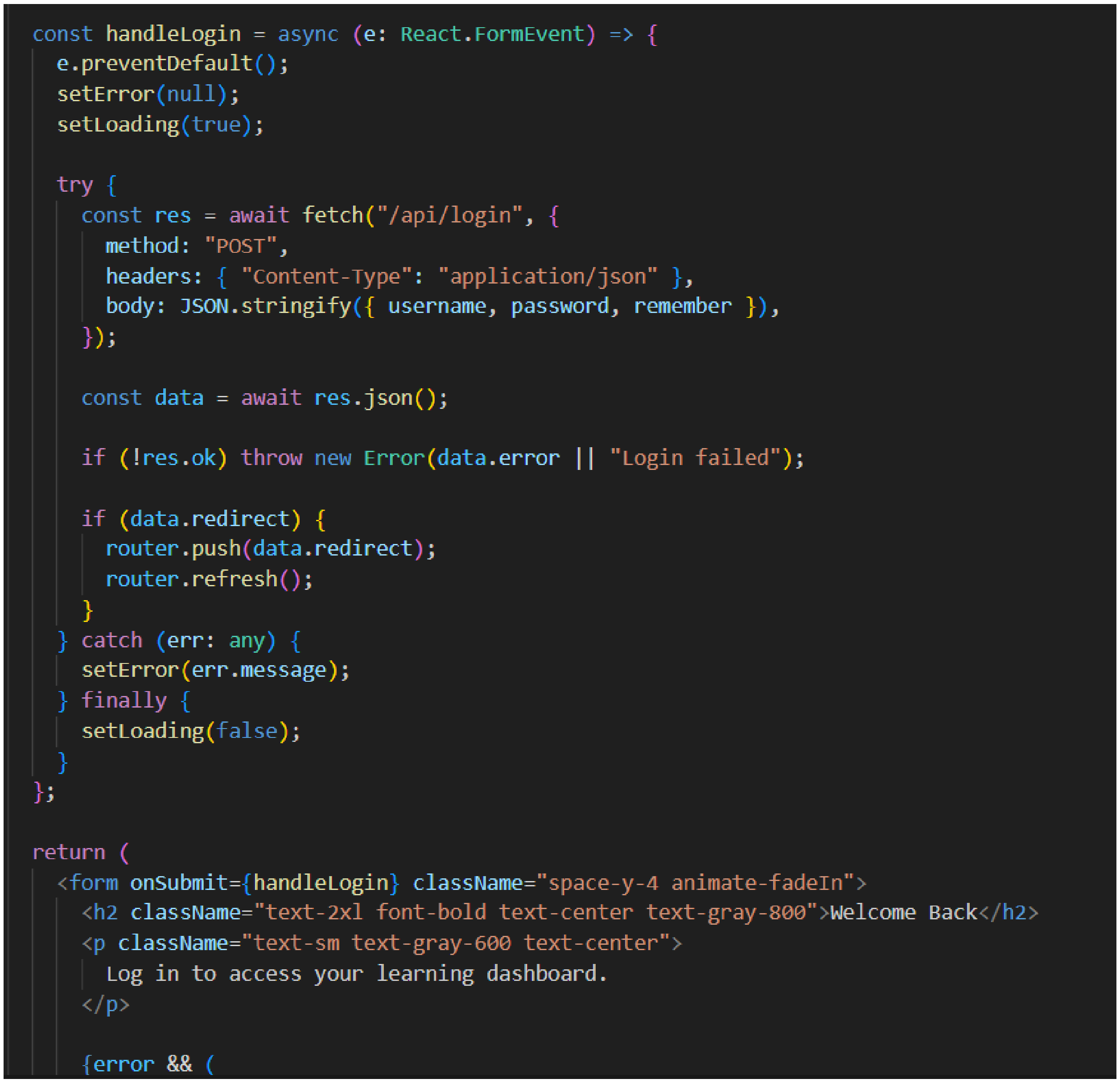
Figure 11(events.ts)

This function retrieves all scheduled events for the given month, parses the JSON response, and sends structured event objects to the frontend. The calendar interface then uses these objects to render a daily grid view, list view, or single-day focus view based on user interaction.

7.5.8 Login and Signup UI Components

The login and signup forms provide structured user input handling, error validation, and communication with backend APIs.

Example from `LoginForm.tsx`:



```
const handleLogin = async (e: React.FormEvent) => {
  e.preventDefault();
  setError(null);
  setLoading(true);

  try {
    const res = await fetch("/api/login", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ username, password, remember }),
    });

    const data = await res.json();

    if (!res.ok) throw new Error(data.error || "Login failed");

    if (data.redirect) {
      router.push(data.redirect);
      router.refresh();
    }
  } catch (err: any) {
    setError(err.message);
  } finally {
    setLoading(false);
  }
};

return (
  <form onSubmit={handleLogin} className="space-y-4 animate-fadeIn">
    <h2 className="text-2xl font-bold text-center text-gray-800">Welcome Back</h2>
    <p className="text-sm text-gray-600 text-center">
      Log in to access your learning dashboard.
    </p>

    {error && (
```

Figure 12(LoginForm.tsx)

The login form submits user credentials by sending a POST request to the internal /api/login route. When the user submits the form, the handleLogin function prevents the default browser behavior, resets previous errors, and activates a loading state. The request is constructed using fetch() with a JSON payload containing the username, password, and any additional fields such as "remember me."

After receiving a response, the code parses the JSON body and checks the HTTP status. If the request fails or the server returns an error message, the function throws an exception that is caught and displayed to the user. If the response includes a redirect value—indicating that authentication succeeded—the component programmatically navigates to the specified route using router.push() and then forces a refresh with router.refresh() to ensure updated authentication state is reflected across server components.

The entire process is wrapped in a try–catch–finally block. This structure ensures that any backend or network errors are surfaced through setError, and the loading indicator is properly reset regardless of success or failure. The result is a reliable and secure login workflow that handles redirects, error reporting, and authentication state updates in a predictable and user-friendly manner.

7.5.9 Overall Architectural Integration

All components operate together as part of a unified and modernized architecture. WordPress provides content, authentication, and user management through its REST API and MySQL database. LearnDash supplies course structures and progress data through authenticated REST endpoints. Next.js manages routing, rendering, and interactive client components, combining server-side rendering with rich user interfaces. The chatbot is powered by OpenAI and integrated through a custom WordPress REST namespace, ensuring that conversational logic remains modular and isolated. Each layer remains independent yet interconnected, forming a high-performance and maintainable system that enhances the WordPress experience through a modern frontend built with Next.js.