Ziang Zhao

Build A Live News Application With Next.js 13

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

2 February 2023

Abstract

Author:	Ziang Zhao
Title:	Build A Live News Application With Next.js 13
Number of Pages:	42 pages + 0 appendices
Date:	2 February 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Mobile Solutions
Supervisors:	Ilkka Kylmäniemi

The objective of this thesis is to leverage the advanced capabilities of Next.js 13 and expertise in front-end development to develop a web application that enables users to access and search for comprehensive information and details pertaining to the latest news globally.

This web application follows an online tutorial titled “Build a live news app with Next.js 13” and employs split-stack development as an architectural paradigm, wherein the front-end and back-end development components are dissociated into two distinct stacks that function autonomously and interact through APIs. The front-end of the application is constructed utilizing the Next.js 13 framework, in conjunction with React, Typescript and Tailwind CSS, while the back-end is established upon a MediaStack News REST API and can be queried through the GraphQL interface implemented using StepZen.

This study showcases the implementation of Next.js 13 and other advanced front-end technologies to develop a compelling user interface for browsing live news. The resulting application affords users the ability to peruse the most recent news updates from across the world, categorize said news according to type, and conduct targeted searches. Additionally, the inclusion of a dark mode feature further enhances the user experience, and the deployment of the

application on Vercel ensures a comprehensive and user-friendly solution for browsing news content.

In conclusion, this study underscores the multifaceted potential and practical applications of Next.js 13 for software development. As such, it constitutes a valuable addition to the body of knowledge in the field, offering insights into the utilization of these technologies to create functional and engaging applications.

Keywords: Next.js 13, React, Typescript, Tailwind CSS, StepZen, GraphQL

Contents

List of Abbreviations

1 Introduction..... 1

2 Theory Background..... 2

2.1 Node.js..... 2

2.2 React.js..... 3

2.2.1 React Components..... 3

2.2.2 JSX..... 5

2.2.3 Virtual DOM..... 6

2.2.4 React Hooks..... 7

2.3 Next.js..... 7

2.3.1 CSR..... 8

2.3.2 SSR..... 9

2.3.3 SSG..... 10

2.3.4 ISG..... 11

2.3.5 Page Routing..... 12

2.3.6 API Routing..... 13

2.4 Next.js 13..... 13

2.4.1 Efficient Routing in Next.js 13..... 13

2.4.2 Server-Side Components..... 14

2.4.3 Asynchronous Components And Fetching Data..... 14

2.4.4 Stream Loading..... 15

2.4.5 Turbopack..... 16

2.4.6 Other Upgrades..... 16

2.5 GraphQL..... 17

2.5.1 StepZen..... 19

2.6 TypeScript..... 19

2.7 Tailwind CSS..... 20

3 Practical Implementation..... 20

3.1 Backend Implementation..... 21

3.2 Frontend Implementation..... 24

3.2.1 Improved Code Management With New App Directory Structure..... 24

3.2.2 Dynamic News Retrieval With StepZen-Generated GraphQL API..... 28

3.2.3 Implementing Dark Mode With Tailwind CSS And Next.js..... 30

4 Conclusion..... 33

References..... 35

List of Abbreviations

API: Application Programming Interface

DOM: Document Object Model

JSON: JavaScript Object Notation, a data format in human readable-text

HTML: Hypertext Markup Language, a code used to build the user interface of the web

HOP: Humble Object Principle

HOC: Higher Order Component

UI: User Interface

JSX: JavaScript XML, a method to construct rendered components

ES6: ECMAScript 6

XML: Extensible Markup Language

SEO: Search Engine Optimization

NPM: Node Package manager

CSR: Client-Side Rendering

SSR: Server-Side Rendering

SSG: Static Site Generation

ISG: Incremental Static Generation

1 Introduction

In recent years, Next.js has gained great popularity among front-end developers. Next.js is a React-based framework that offers a comprehensive set of tools and configurations for building web applications. As a framework, it streamlines the process of developing a React application by handling the necessary setup and providing enhanced structure, features, and optimization options. It is built on top a popular library for building user interfaces called React, React's popularity is partly due to its flexibility, as it doesn't dictate how the other parts of an application should be built. This has led to the growth of a thriving ecosystem of third-party tools and solutions (Vercel Inc 2023). With React, you can create your user interface and gradually incorporate Next.js features to address common application needs like routing, data retrieval, and integrations, all while enhancing the developer and user experience. Additionally, Next.js version 13, released in January 2023, introduces new features such as new app directory and Turbopack, further increasing its utility for web developers.

The purpose of this thesis is to utilize Next.js 13's advanced features and front-end development skills to create a web-based platform based on an online tutorial called “Build a live news app with Next.js 13” that enables users to easily search and access extensive information about the latest global news.

The initial part of this thesis introduces the basic theoretical knowledge. It comprehensively examines the concepts of Node.js, React.js, and Next.js, and provides a complete explanation of the core features and related principles of Next.js. In addition, this thesis delves into the latest trends of Next.js 13, providing an in-depth study of its new features and usage. Finally, the practical implementation section of this thesis presents a comprehensive manual for constructing front-end web applications utilizing Next.js 13.

2 Theory Background

2.1 Node.js

Node.js is a powerful framework that allows developers to run JavaScript code outside of a web browser (Node.js Foundation 2014). Before Node.js, JavaScript was completely a client-side technology to manipulate the DOM and implement animations inside the browser. Moreover, Node.js has advanced significantly with ES6, making advantage of new arrow functions, handling files and databases, internet communication, and sending responses in a uniform format (Node.js Foundation, n.d.). Further benefits that Node.js offers include event-driven, asynchronous programming, and non-blocking I/O, which other backend technologies may not have or may be lacking in (Node.js Foundation, n.d.). As a result, Node.js helps developers arrange their code more effectively and increase reusability.

The event loop architecture used by Node.js processes and responds to each request provided from the client to the server one at a time, whereas many requests can be handled concurrently using separate threads. Node.js continues activities after receiving a response rather than blocking threads or wasting CPU time while waiting for it (Node.js Foundation, n.d.).

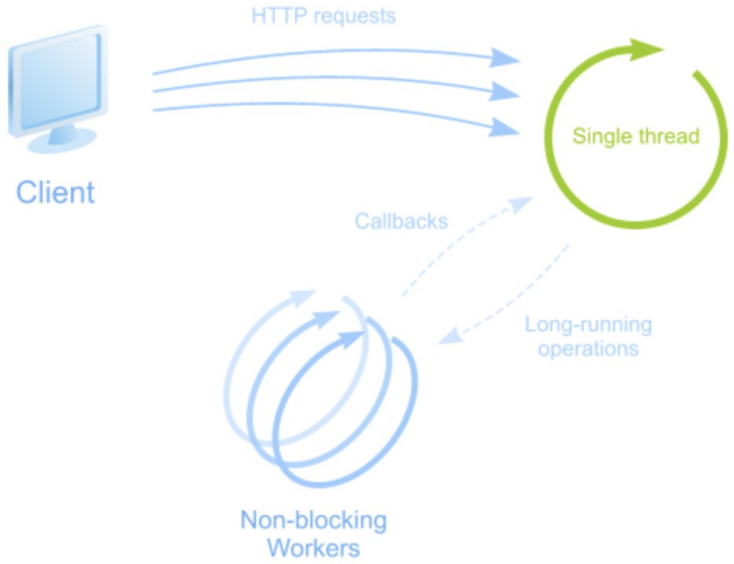

Figure 1. How does Node.js handle multiple requests.

As a result, Node.js' design enables it to handle a lot of requests while yet maintaining excellent speed levels. Figure 1 shows the architecture of Node.js, giving an overview of its parts and structure.

2.2 React.js

Facebook developed the powerful JavaScript library React, which is now utilized by many businesses and developers (W3schools.com, n.d.). It enables developers to design massive, data-altering apps without the need for page updates, making it an important component of the Model View Controller (MVC) architectural pattern. React.js is well-known for abstracting the Document Object Model (DOM) to improve development efficiency (W3schools.com, n.d.). Furthermore, it is mostly server-side rendered with Node.js and includes mobile app compatibility with React.js Native.

React.js uses a one-way data flow that is less difficult than conventional data binding and simplifies boilerplate code. The virtual DOM is the essential idea behind this framework, and it makes excellent use of it. Both client-side and server-side rendering of the virtual DOM are possible, and both instances can talk to one another. The virtual DOM constructs a subtree of nodes based on state changes and does minimum DOM operations to keep the component current instead of doing explicit updates.

2.2.1 React Components

In React, components are JavaScript classes or functions that are used to render different UI elements. Each component describes how a certain area of markup will appear at any given time (W3schools.com, n.d.). They have a collection of "props" or attributes, that enable the transfer of data from parent to child components. A component can also keep track of its own internal state, which gives it the ability to manage its own data and react to changes.

A JavaScript function called a function component creates a section of the user interface. As seen in Figure 2, it can accept input in the form of props, which are then utilized to build the user interface.

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Usage:
<Greeting name="John" />
```

Figure 2. Example of a function component in React.

A class component is a type of JavaScript class that inherits from the React.Component class. Along with props, it can maintain its own internal state and use lifecycle methods to perform various actions, Figure 3 is an illustrative example.

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  handleClick = () => {
    this.setState({ count: this.state.count + 1 });
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.handleClick}>Click me</button>
      </div>
    );
  }
}

// Usage:
<Counter />
```

Figure 3. Example of a class component in React.

A HOC, or a higher-order component, is a function that accepts a component as input and produces a new component with additional capabilities. These HOCs are frequently utilized to enhance code reuse and incorporate supplementary features like logging, authentication, or data retrieval to existing components. An example will be shown in Figure 4.

```
function withData(WrappedComponent) {
  return class extends React.Component {
    state = { data: [] };

    componentDidMount() {
      // fetch data and update state
    }

    render() {
      return <WrappedComponent data={this.state.data} {...this.props} />;
    }
  }
}

// Usage:
const MyComponentWithData = withData(MyComponent);
```

Figure 4. Example of a higher-order component in React.

The preceding examples barely scratch the surface of the universe of React components. In reality, React components can be significantly more intricate and integrated to create engaging, dynamic user experiences.

2.2.2 JSX

Similar to template syntax or an XML-like extension to ECMAScript syntax, JSX is a JavaScript syntactic extension that enables developers to write HTML-like code in their JavaScript files (Reactjs.org, n.d.). It can be converted by Babel into standard JavaScript code that web browsers can run and has all the features of JavaScript. This increases the effectiveness and experience of development and produces more succinct code with a clearer code structure.

In JSX, developers can use HTML-like tags to create elements and use JavaScript expressions inside curly braces to dynamically generate content. This syntax sugar allows developers to write JS code as if it were HTML, reducing learning costs and improving development efficiency. For example, a simple JSX component might look like this in Figure 5.

```jsx
const Greeting = ({ name }) => {
  return (
    <div>
      <h1>Hello, {name}!</h1>
      <p>Welcome to my website.</p>
    </div>
  );
};
```

Figure 5. Example of a JSX component in React.

This component takes a name prop and uses it to dynamically generate a greeting message. The JSX code is compiled into regular JavaScript code that creates the corresponding DOM elements when the component is rendered. While it is not strictly required to use JSX with React, it is a highly recommended approach as it makes the code more concise and easier to read and understand.

2.2.3 Virtual DOM

React uses the Virtual DOM to improve the speed and effectiveness of changing the user interface of a web application (Reactjs.org, n.d.). When the state of a component changes, React calls the render function to re-render the entire component's UI. However, the large number of DOM operations can lead to significant performance issues. Therefore, the virtual DOM becomes the solution to the problem at this point. The virtual DOM is mapped to the component's DOM structure, which is a lightweight approximation of the actual DOM built in memory (Reactjs.org, n.d.).

Moreover, React applies a diff process to the Virtual DOM. When the component has to be re-rendered, it first uses the diff technique to identify the DOM node that needs to be changed. Then, it updates these modifications to the real DOM node in the browser when the component has to be re-rendered (Reactjs.org, n.d.). Performance is improved as a result of not having to render the complete DOM tree again. In addition, the Virtual DOM is substantially

quicker than the native DOM since it is a data structure created only in JavaScript. In conclusion, the Virtual DOM is a crucial component of the React architecture that enables developers to create declarative code that is simpler to understand and maintain while also allowing for quick and efficient UI changes.

2.2.4 React Hooks

After the introduction of Hooks in React version 16.8, functional components have become even more powerful. This is because React hooks enable the decoupling of React business logic from UI in a way that class components cannot. When using hooks, it is good to distinguish two different behaviors, one is setting the value (setState), and another is handling the side effects of the behavior (useEffect) which is a very good decoupling (Reactjs.org, n.d.). In this case, our UI layer can be almost completely turned into a HOP (Humble Object Principle), and the hook implements the interaction between the HOP and the business logic. As a result, React hooks become an effective tool for business logic and UI dependency inversion. Therefore, using functional components as a development approach will reduce the overall maintenance effort of the project and improve development efficiency.

2.3 Next.js

Next.js is a React application framework designed for production environments, which means it is stable and functional with numerous real-world use cases (Vercel Inc 2023). This framework enables developers to create React applications rapidly without having to spend much time and effort on configuring various development tools.

A broad range of configuration options are available to applications created using the Next.js framework, including server-side rendering (SSR), compile-time rendering (SSG), support for Typescript, automated packaging, routing, and loading. Because of this, Next.js is a great option for developers who want to shorten their development process without sacrificing the

dependability and speed of their apps. It provides a dependable and effective way to create React apps.

2.3.1 CSR

Client-Side Rendering, sometimes known as CSR, is a method for rendering web pages in which the server responds to requests with an HTML document including JavaScript code. The JavaScript code then executes in the browser and renders the page content dynamically. This strategy can give a more dynamic and interactive user experience, as page modifications can be implemented without requiring a full-page refresh. Additionally, being a framework for server-side rendering, Next.js enables the implementation of CSR through the useEffect hook and client-side routing.

```
import { useState, useEffect } from 'react'
function Home() {
  const [data, setData] = useState(null)
  const [isLoading, setLoading] = useState(false)
  useEffect(() => {
    setLoading(true)
    fetch('/api/get-data')
      .then((res) => res.json())
      .then((data) => {
        setData(data)
        setLoading(false)
      })
  }, [])
  if (isLoading) return <p>Loading...</p>
  if (!data) return <p>No data</p>
  return (
    <div>
      // Use data
    </div>
  )
}
```

Figure 6. Fetching data with useEffect() hook.

Use the useEffect hook to retrieve data from a remote API or database when the component mounts in order to implement CSR in Next.js. Besides, client-side routing can also be used to update page content without reloading the full page, as shown in Figure 6.

2.3.2 SSR

With server-side rendering (SSR), when a user visits a web page, the browser sends a request to the server about that page. Next, the server retrieves the necessary data from the database and sends it along with the page content to the browser. Finally, the browser then displays it to the user.

For each link that the user clicks, the browser sends this request, which means that the server handles the request every time (Nextjs.org, n.d.). This can potentially reduce the website's performance. However, server-side rendering is very suitable for pages that use dynamic data. Whenever a user requests it, the page is rebuilt using getServerSideProps, as shown in Figure 7.

```
export default function Home({ data }) {
  return (
    <main>
      // Use data
    </main>
  );
}
export async function getServerSideProps() {
  // Fetch data from external api
 const res = await fetch('https://.../data')
  const data = await res.json()
  // Will be passed to the page component as props
  return { props: { data } }
}
```

Figure 7. An example of how to use getServerSideProps function.

getServerSideProps runs only on the server. First, when a user directly accesses a page, it runs on request and the page uses the properties returned by it for pre-rendering. Second, when a user accesses a page through a Next link, the browser sends a request to the server running it.

2.3.3 SSG

With Static Site Generation (SSG), data is only fetched once during the build time for the page. Static generated pages are very fast and performant because all pages are pre-built (Nextjs.org, n.d.). SSG is therefore well-suited for pages that use static content (such as sales pages or blogs). This approach is useful for pages that require a lot of data, as it can reduce the page's time to interactivity. Additionally, SSG can also help with search engine optimization (SEO), as search engines can easily crawl and index the page content. In Next.js, users must export the getStaticProps function from the pages they want to statically render, as shown in Figure 8.

```
export default function Home({ data }) {
  return (
    <main>
      // Use data
    </main>
  );
}
export async function getStaticProps() {
  // Fetch data from external API at build time
  const res = await fetch('https://.../data')
  const data = await res.json()
  // Will be passed to the page component as props
 return { props: { data } }
}
```

Figure 8. An example of how to use getStaticProps function.

Moreover, the user can also query the database in getStaticProps, as shown in Figure 9 below. In conclusion, SSG is a strong feature that may greatly enhance the functionality and user interface of Next.js apps.

2.3.4 ISG

Sometimes, users may want to use SSG but also need to update the content periodically. In such cases, Incremental Static Generation (ISG) can be very helpful. ISG allows users to create or update static pages at specified intervals after building the static pages. This way, users do not need to rebuild the entire site, only the pages that require it. ISG preserves the benefits of SSG and adds the benefit of providing the latest content to users. ISG is particularly suitable for pages on a site that use data that is constantly changing. For example, users can use ISG to render blog posts so that the blog stays up to date when editing or adding new posts. To use ISG, add the revalidate attribute to the getStaticProps function on the page, as shown in Figure 9.

```
export async function getStaticProps() {
  const res = await fetch('https://.../data')
  const data = await res.json()
  return {
    props: {
      data,
    },
    revalidate: 60
  }
}
```

Figure 9. An example of using getStaticProps function.

In the example above, when the request arrives after 60 seconds, Next.js will try to rebuild the page. The next request will generate a response with the updated page.

2.3.5 Page Routing

After initializing the React application with Next.js, it automatically renders each component in the pages directory into a page and automatically sets up a route to a new about.js in the pages directory, as shown in Figure 10.

```
// pages/about.js
export default function About() {
  return <div>About: </div>;
}
```

```
pages
.
├── _app.js
├── about.js
├── api
│   └── hello.js
└── index.js
```

Figure 10. An about component under pages folder and the file structure.

The about page can be accessed after the program has been launched at http://localhost:3000/about. Additionally, Next.js also enables dynamic routing of pages. The pages directory can contain a file called “posts/[id].js”, as illustrated in Figure 11, and Next.js will automatically route requests like “/post/1” and “/post/2” to this component.

```
// pages/posts/[id].js
import { useRouter } from "next/router";

export default function Post() {
  const router = useRouter();
  const { id } = router.query;
  return (
    <div>
      Post
      <div>id: {id}</div>
    </div>
  );
}
```

Figure 11. Dynamic routing in Next.js.

At this point, requests such as http://localhost:3000/posts/1 and http://localhost:3000/posts/2 will be routed to the component, and you can see that the routing parameters can be obtained through the router.

2.3.6 API Routing

Next.js provides a solution for building APIs, similar to how page routing is used above, files added under the "pages/api" directory are mapped to the "/api/*" API (Nextjs.org, n.d.). When building the project, the files under "pages/api" will not increase the size of client-side packages, but only the size of server-side packages.

```
// pages/api/user.js  → /user

export default function handler(req, res) {
  res.status(200).json({ data: 'hello API' })
}
```

Figure 12. An example of API routing in Next.js.

The code in Figure 12 returns a json response with a status code of 200.

2.4 Next.js 13

Next.js 13's debut signifies a comprehensive initiative to combine React's dual identities as a user interface library and architectural framework. As a result, this section will focus on the intriguing novel attributes of Next.js 13.

2.4.1 Efficient Routing in Next.js 13

Next.js offers file-based routing, a powerful feature that simplifies route settings by allowing the specification of routes using the project's directory structure instead of using complex routing systems like react-router. By adding an entry point to the directory page, a new path can be created.

In Next.js 13, the file routing mechanism has been updated with the introduction of a new directory that offers various new features and enhancements, which are optional. Consequently, the directory structure has undergone some modifications. Specifically, each path in the route is now linked to a specific directory that includes a page.js file acting as the entry point for the content. Moreover, the new directory structure permits the inclusion of additional files in each path directory. For example, there is the layout.js that represents a path and its sub-paths system, the loading.js that uses React to create an instant loading system, and the error.js, a component displayed when the primary component fails to load.

Furthermore, it is now possible to co-locate source files inside path directories since each path is now its directory.

2.4.2 Server-Side Components

The latest release of Next.js introduces an exciting development in the form of extended support for React server-side components. The utilization of server-side components allows for the execution and rendering of React

components on the server-side, which results in faster delivery, smaller JavaScript packages, and less overhead client-side rendering.

Additionally, depending on the type of data required for generating routes, server-side components are automatically cached either at build time or runtime to enhance performance. The combination of server-side and client-side components offers the option of using server-side components for fast-loading, non-interactive sections of the application, while client-side components can be employed for interactions, browser APIs, and other functionalities.

2.4.3 Asynchronous Components And Fetching Data

Next.js 13 presents a new development in the form of async components that offer a novel approach to data collection for server-rendered components (Vercel.com, n.d.). With the utilization of async components, systems can be rendered using Promises using async & await.

```
async function fetchData() {
  const response = await fetch('https://api.example.com/data');
  const data = await response.json();
  return data;
}

export default async function HomePage() {
  const data = await fetchData();
  return (
    <div>
      <h1>{data.title}</h1>
      <p>{data.description}</p>
    </div>
  );
}
```

Figure 13. An example of fetching data in Next.js 13.

When requesting data from an external service or API that returns a Promise, the component is declared as async and the response is awaited, as seen in Figure 13. This approach ensures that data is retrieved from an external service or API before rendering the component, resulting in efficient and reliable server-side rendering.

2.4.4 Stream Loading

In the previous version, users had to wait for the entire page to be generated before it could be transmitted. However, with the new release of Next.js, the server now transmits smaller pieces of the UI as they are generated, avoiding interruptions caused by larger pieces (Vercel.com, n.d.). This feature is currently only supported by the app directory, and there are no indications of further support (Vercel.com, n.d.).

This feature is particularly beneficial to users with weaker internet connections, as faster site loading times improve their user experience. Although users with stronger internet connections or faster Wi-Fi may not see as much of a difference, it is worth noting that there are more users with weaker internet connections than one might think.

2.4.5 Turbopack

Next.js version 13 features a notable modification with the introduction of Turbopack, a new JavaScript bundler that replaces Webpack, one of the most widely used JavaScript build tools. Although Webpack is known for its configurability and potency, it can sometimes be slow and complicated. However, Turbopack, developed by the creators of Webpack and built with Rust, claims to be 700 times faster than its predecessor Webpack and 10 times faster than the more contemporary alternative, Vite (Vercel.com, n.d.).

2.4.6 Other Upgrades

The latest version of Next.js has introduced various upgrades, such as the new Image component, @next/font, and next/link. The new Image component reduces client-side JavaScript, improves accessibility, and provides simpler styling and configuration options. To upgrade to this new component, the previous next/legacy/image and next/future/image imports have been renamed to next/image, and a codemod is available for quick migration.

In addition, the new @next/font enables the use of Google Fonts or custom fonts without submitting any queries to the browser. The font files and CSS are downloaded during build time along with other static assets.

Next.js has also introduced next/link, a novel font system that enhances efficiency and privacy by offering automatic font optimization and the option to integrate custom fonts without the use of external network requests. These upgrades have made Next.js a more efficient and privacy-friendly technology for developers.

2.5 GraphQL

GraphQL is a query language and runtime for APIs, initially proposed by Facebook (Graphql.org, 2012). It enables developers to describe the data available in their API in a comprehensive and clear way. This means that clients can easily understand what data is available and obtain precisely the data they need, without redundancy (Graphql.org, 2012).

With traditional REST API endpoints, client applications request server resources and receive responses containing all the data matching the request. For example, a successful response from a REST API endpoint returns 35 fields, the client application will receive 35 fields. In contrast, with GraphQL, different client applications within an organization can easily query only the

required data, which surpasses other REST methods and brings about a real improvement in application performance.

Here's an example of a GraphQL query that retrieves the name and email address of all users in a hypothetical database, as shown in Figure 14.

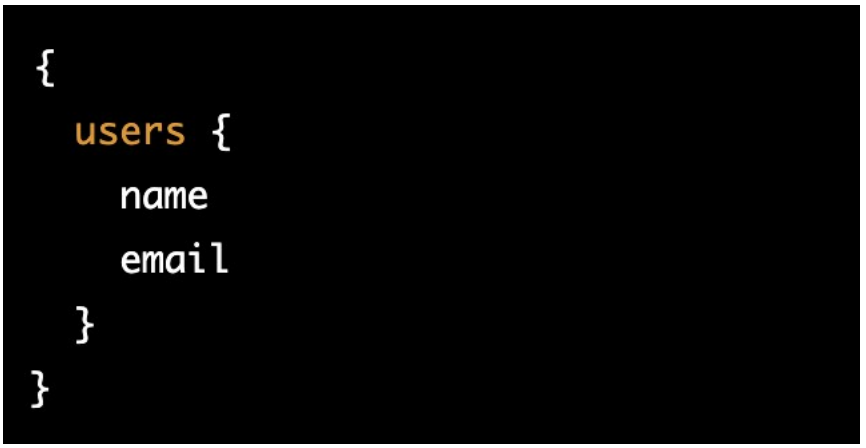

Figure 14. An example GraphQL query.

The query above simply asks for the user's name and email with no other information. Therefore, the resulting response will be shown in Figure 15 below.

Figure 15. An example response data from a GraphQL query.

The server sends a JSON response to the client containing the requested data. Specifically, the response includes an array of objects, each of which has a name and an email field. In comparison to traditional REST APIs, which require loading more URLs, the GraphQL API is designed to be fast and efficient. This is achieved by obtaining all the necessary data in a single request query, which allows it to perform well even on slow mobile network connections.

In conclusion, GraphQL is a preferred syntax because it allows clients to precisely indicate the data they require, enables straightforward data aggregation from various sources, and uses a type system to describe data, rather than endpoints.

2.5.1 StepZen

StepZen is a GraphQL server that has a distinct structure, designed to aid developers in quickly constructing APIs. By utilizing declarative configurations, developers can write less code (Stepzen.com, n.d.). The APIs operate within an in-memory GraphQL engine that is based on Golang and is deployed in Kubernetes, making them very responsive to the needs of the application (Stepzen.com, n.d.). With StepZen, developers can write a small amount of declarative code and still achieve impressive results. Whether the backends are REST, databases, or GraphQL, the team can establish a flexible GraphQL layer in just a few days, rather than taking weeks or months. The APIs are succinct and declarative, and they can operate on StepZen's cloud or in a private cloud with built-in optimizations for performance, cost, and reliability.

2.6 TypeScript

TypeScript, a programming language developed by Microsoft and made available to the public in October 2012, has been used to expand the capabilities of JavaScript. Since it was first made available to the public, it has rapidly risen to become one of the programming languages that is most frequently used. It is essential, for the purpose of having a comprehensive understanding of TypeScript, to investigate its concept and its goal.

Moreover, TypeScript is regarded as a strict superset of JavaScript, it solves one of the most significant problems with JavaScript by introducing a powerful type system (W3schools.com, n.d.). Both TypeScript and JavaScript may be used interchangeably, which implies that TypeScript code can be compiled into standard JavaScript. This makes it possible for both forms of code to function together in a project without any complications. Even though typing in TypeScript is not required, its close relationship to JavaScript makes it simple for developers to pick up and adjust to using it. The major goal of TypeScript is to improve scalability in projects, especially those that involve a large number of contributors. In circumstances like this, having consistent and bug-free code is

essential, and TypeScript does exceptionally well when it comes to attaining this goal.

2.7 Tailwind CSS

Tailwind CSS is a CSS framework that allows developers to quickly style websites with ease. Unlike other CSS frameworks, Tailwind CSS is a utility-first framework, meaning it provides building blocks that developers can use to build custom designs quickly without imposing any opinionated styles. This low-level CSS framework is highly customizable, and developers have complete control over the design of their website. One of the advantages of using Tailwind CSS is that it allows developers to build unique user interfaces by putting together tiny components, without being restricted by any pre-designed templates or specifications. To use Tailwind CSS, you only need to provide a "raw" CSS file, which will be processed by a configuration file to produce the desired output. This configuration file enables web designers to alter the look and feel of their website. Due to its adaptability and simplicity, Tailwind CSS has become a preferred tool among developers who want to create unique user experiences rapidly.

3 Practical Implementation

In the practical application portion of this thesis, it is intended to combine the advanced capabilities of Next.js 13 and front-end programming skills to create a web-based platform that enables users to rapidly search and access a vast quantity of information on the newest global news. The front-end of the application was developed using the Next.js 13 framework in conjunction with React, Typescript, and Tailwind CSS, while the back-end was developed using the MediaStack News API and a GraphQL interface built using StepZen. The application will be deployed on a Vercel server, and an automatic Vercel deployment will be set up by submitting the code to a GitHub repository.

3.1 Backend Implementation

The back-end implementation of the project is based on the serverless idea; the so-called serverless backend is a platform that eliminates or reduces the need for developers to perform complex and time-consuming back-end server operations. It liberates developers to focus on front-end functionality and user experience. Thus, this project will leverage StepZen technology to construct GraphQL API based on the REST API supplied by the MediaStack platform in an effective and rapid manner.

To utilize the MediaStack REST API, you first sign up for a free account on their website and then browse to the dashboard's fast start section to obtain an API key, as shown in Figure 16.

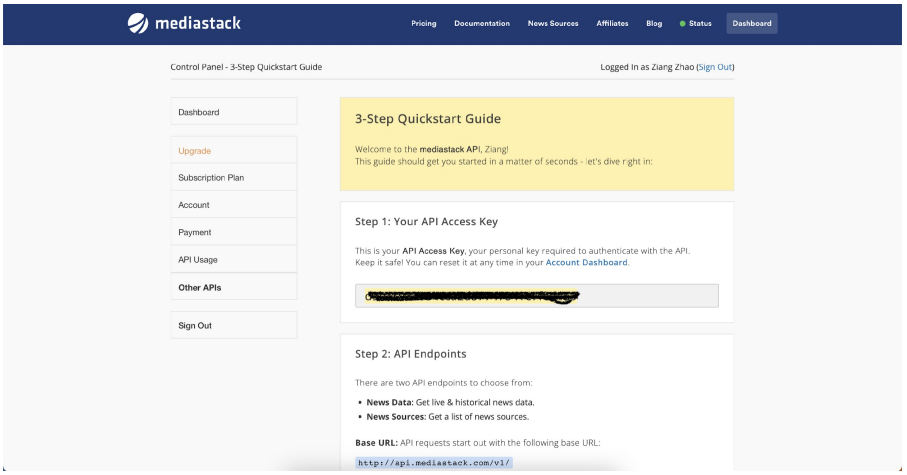

Figure 16. Get the user’s private API access key.

After obtaining your private API access key, you may begin testing the API using curl queries. As illustrated in Figure 17, you may submit the following request to retrieve the most recent news from the United States.

```
"http://api.mediastack.com/v1/news?access_key=YOUR_ACCESS_KEY&countries=us"
```

Figure 17. Example endpoint of fetching latest news from the United States.

Note that to make the request successfully, you must replace the access_key query with your actual API key.

Now that you have confirmed that the MediaStack REST API is working correctly, it's time to leverage StepZen to create a GraphQL API with ease. To begin, you will need to install the StepZen CLI after successfully logging in. Once you have logged in, you can proceed to enter your administrative key when prompted in the terminal.

Figure 18. Example workflow of using StepZen CLI.

After successfully setting up the StepZen CLI, you will receive a confirmation message similar to the one displayed in Figure 18. Next, you can create a new endpoint by utilizing your chosen backend through the command "stepzen import curl" and integrating your REST API. Once this is completed, you can initiate your work by executing "stepzen start".

Figure 19. Import curl data source into your GraphQL schema.

Figure 20. Deploy the new GraphQL endpoint to StepZen server.

StepZen will automatically develop and deploy the GraphQL API on its server following the execution of the instructions described before, as seen in Figures 19 and 20.

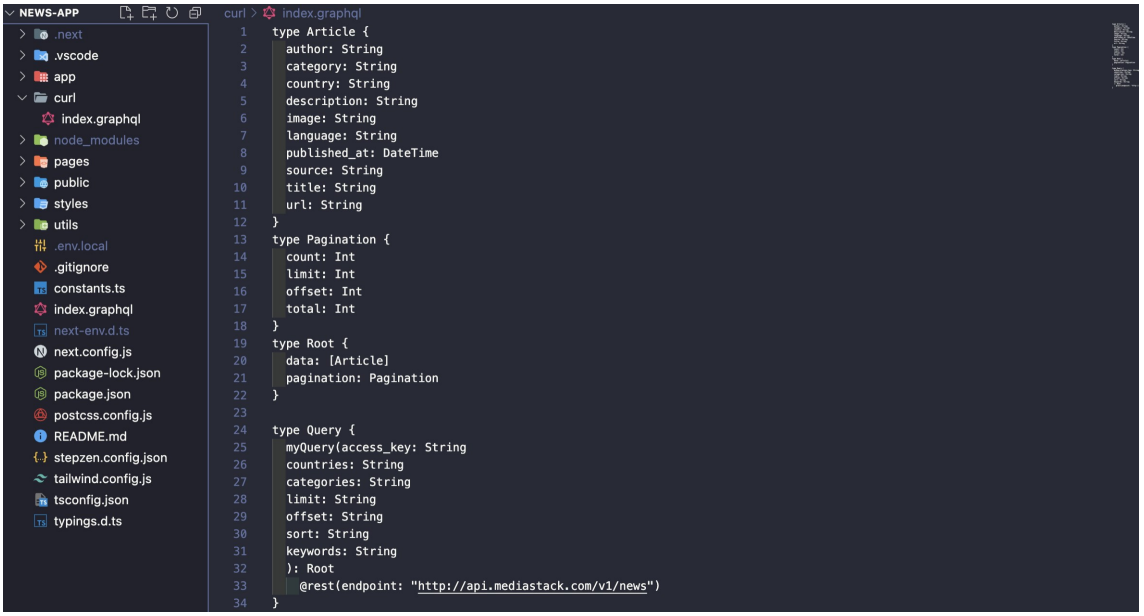

Figure 21. An index.graphql file that auto generated by StepZen.

In the meanwhile, as represented in Figure 21, StepZen is capable of automatically generating all the necessary type definitions and other requirements for your project based on the REST API response. This is accomplished in a smooth and quick manner, and the file is instantly stored in a folder named "curl" within your project.

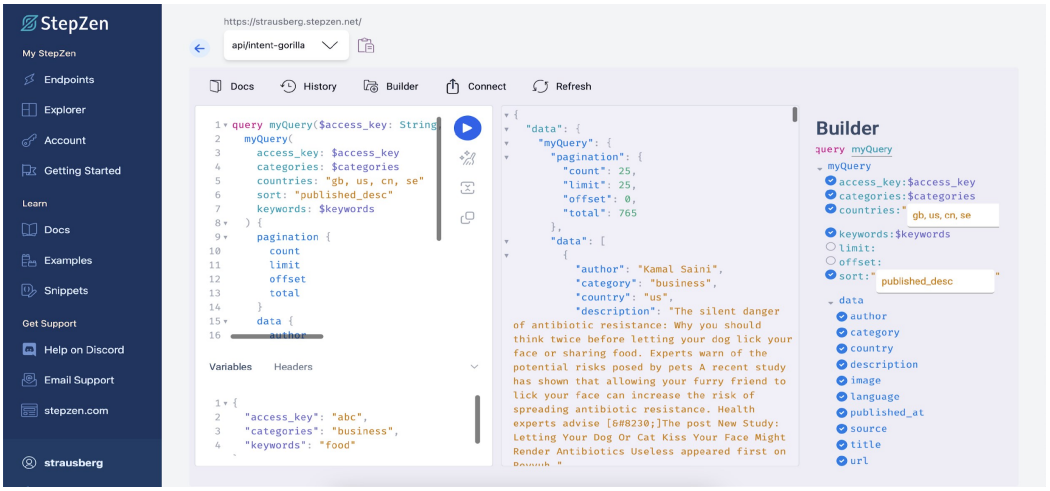

Figure 22. A GraghQL interface builder in StepZen dashboard.

As seen in Figure 22, it is also possible to adapt the searches to your specifications by leveraging variables or a specified value with the StepZen endpoint builder. Moreover, you may choose the response data that best suits your frontend requirements. Interfacing with StepZen is all that is required, as StepZen will automatically retrieve the data on their end.

3.2 Frontend Implementation

The front-end of this web application will utilize the Next.js13 framework to process the information we present to the user and StepZen to connect with the backend GraphQL API. This example project will demonstrate an online news system that allows users to search and query for relevant news articles based on their preferences.

3.2.1 Improved Code Management With New App Directory Structure

Next.js 13 comes with an updated app directory structure that simplifies code management and improves code maintainability for developers. The new application directory separates code into distinct logical modules, including pages and header design. Development can be substantially facilitated by dividing up the front-end into smaller components. This method makes it simpler to monitor, manage, and reuse code, while also lowering the total amount of code in the project. By creating reusable components, developers are able to make changes to particular files that are subsequently mirrored throughout the whole project. This method is especially effective for refactoring since it enables developers to concentrate on editing a single file rather than updating several files throughout the project.

Figure 23. New decomposition style in Next.js 13.

Figure 23 depicts how the front-end utilizes component deconstruction. In addition, the app directory's routes are determined by the files and folders included within it. Inside a folder, the page file defines the user interface for a single route. Thus, an equivalent folder structure to "app/article/page.tsx" will be responsible for displaying the article route.

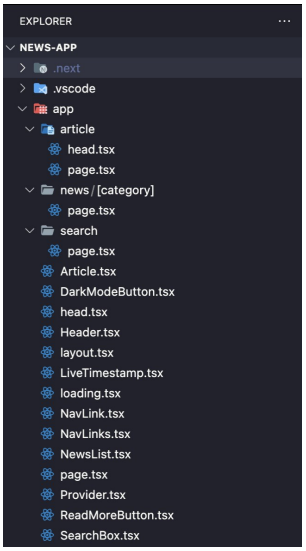

Figure 24. Routing structure in the front-end side.

Figure 24 describes the application's routing architecture. This application has four distinct routes: the root, article, news, and search routes. In addition, the "loading.tsx" file is optional and can be created in any subfolder of the application directory. It automatically places the page in React suspense limits, and it will display immediately when the page initially loads and when transitioning between related routes. Furthermore, the "layout.tsx" file is a customisable component that creates a common user interface across numerous pages. As a result, it lets you to create a unified layout for your website or application, including a header, footer, navigation menu, and any other typical elements.

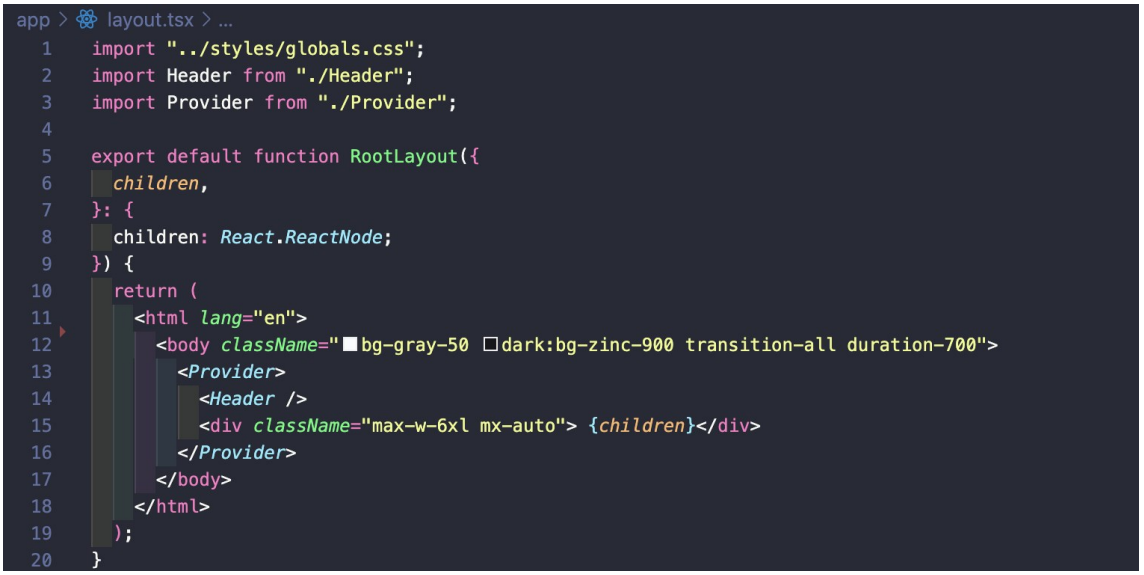

Figure 25. An example of layout.tsx in this application.

As seen in Figure 25, it is important to note that the root directory of the application must have a file that defines the essential layout. This layout applies to all the app's routes. In addition, the root layout must have both the <html> and <body> tags because Next.js 13 does not automatically include them.

Figure 26. An example of head.tsx in this application.

Additionally, a "head.tsx" file within any directory specifies the contents of the head> tag for a particular route. It offers a collection of tags that include essential page information, including as the title, favicon, and description. As seen in Figure 26, the component returns a title tag with the phrase "Home Page", a meta tag that defines the viewport width and initial scale, a description tag that states "Generated by construct next app", and a link tag that references a favicon file named "favicon.ico". This component may often be used on any page to modify the content of the head> element.

Figure 27. An example of client-side component in Next.js 13.

Every component built in the app directory is a React server component by default, which improves efficiency by lowering bundle size. Consequently, the key benefit of utilizing server components is that they do not add to the overall bundle size of your application, resulting in reduced download sizes for users and quicker page loads.

Use the "use client" directive at the beginning of the code, as shown in Figure 27, if we wish to switch to a client component instead. When interactive components or dynamic material that need client-side rendering are included. In this scenario, the "use client" directive may be used at the beginning of the file to signal that the component should be constructed as a client-side component.

3.2.2 Dynamic News Retrieval With StepZen-Generated GraphQL API

The fetch Web API in React and Next.js has been improved to enable automated request deduplication and a flexible approach to obtaining, caching, and revalidating component-level data. Users may now experience the benefits of Static Site Generation (SSG), Server-Side Rendering (SSR), and Incremental Static Regeneration (ISR) under a single API thanks to these changes.

```
import { gql } from 'graphql-request'
import sortNewsByImage from './sortNewsByImage'

const fetchNews = async (
    category?: Category | string,
    keywords?: string,
    isDymamic?: boolean,
) => {
    // GraphQL query
    const query = gql`
    query MyQuery(
        $access_key: String!
        $categories: String!
        $keywords: String
        ) {
        myQuery(
            access_key: $access_key
            categories: $categories
            countries: "gb, us, cn, se"
            keywords: $keywords
            sort: "published_desc"
            ) {
        data {
            author
            category
            country
            description
            image
            language
            published_at
            source
            title
            url
        }
        pagination {
            count
            limit
            offset
            total
        }
        }
    }
`
    //  Fetch function with Next.js 13 caching
    const res = await fetch("https://strausberg.stepzen.net/api/intent-gorilla/__graphql", {
        method: "POST",
     //   cache: isDymamic ? "no-cache" : "default",
        next: isDymamic ? { revalidate: 0 } : { revalidate: 30 },
        headers: {
            "Content-Type": "application/json",
            Authorization: `Apikey ${process.env.STEPZEN_API_KEY}`,
        },
        body: JSON.stringify({
            query,
            variables: {
                access_key: process.env.MEDIASTACK_API_KEY,
                categories: category,
                keywords: keywords,
            },
        })
    })
    const newsResponse = await res.json()

    const data = sortNewsByImage(newsResponse.data.myQuery)

    return data

}

export default fetchNews
```

Figure 28. A custom function for fetching news.

Figure 28's code exports an asynchronous method named fetchNews that receives news items from the StepZen-generated GraphQL API endpoint. The function accepts category, keywords, and isDynamic as optional inputs. If the isDynamic variable is used, the data is instantly re-validated; otherwise, a new

request is issued to the server when the cached value is out of date, which significantly decreases server-side traffic and increases efficiency. This function enables dynamic retrieval and sorting of news articles from GraphQL API endpoints in the Next.js 13 application. Afterwards, the data is retrieved from the endpoint and converted to JSON format. The method sortNewsByImage then sorts and returns the response data.

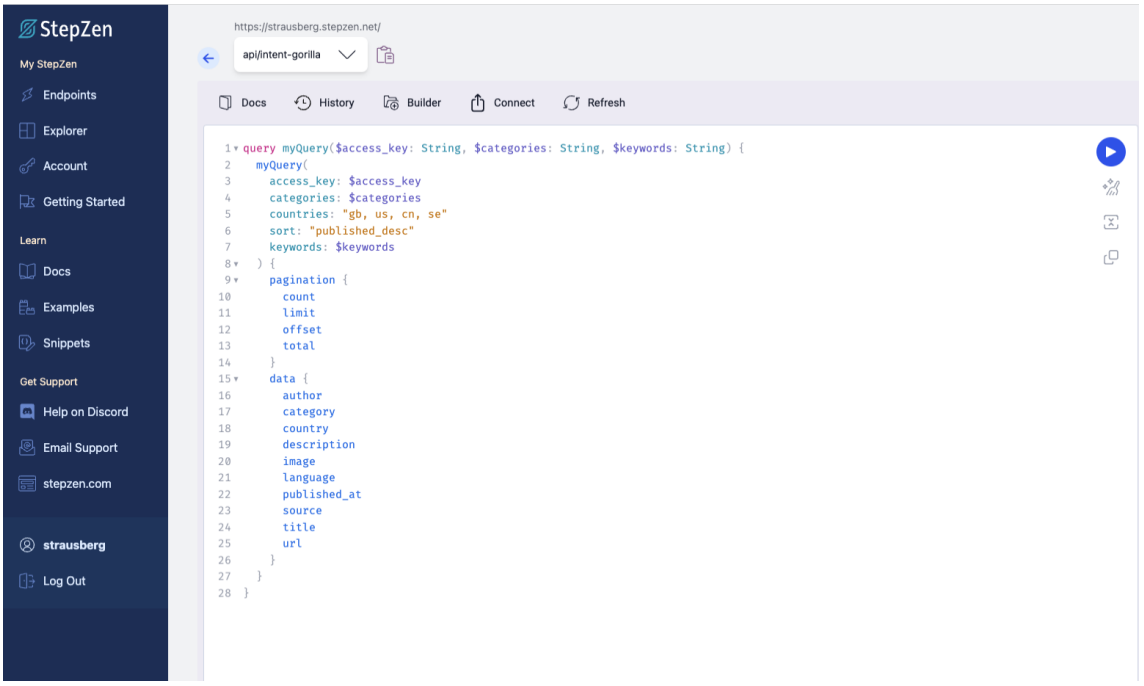

Figure 29. GrapQL query generated by StepZen.

As demonstrated in Figure 29, the StepZen dashboard allows immediate access to queries that are automatically produced. This method conducts the GraphQL query against the API endpoint and returns a collection of variables that includes access keys, categories, and keywords.

3.2.3 Implementing Dark Mode With Tailwind CSS And Next.js

The front-end of the application makes use of Tailwind CSS, which significantly increases both the quality of the development experience and the efficiency with which development is carried out. Installing Tailwind CSS and all of its dependencies is the first step in integrating it into your project. Secondly, you

will need to generate a configuration file, and finally, you will need to import the CSS styles into your Next.js project.

Tailwind CSS is a utility-first CSS framework that enables developers to decorate HTML components fast and simply by applying predefined CSS classes. According to the code example provided in Figure 30. First, the “className” is a React prop used to apply Tailwind CSS classes to an HTML element in the provided code. Next, h-8 sets the element's height to 8 units, the “h” prefix abbreviates the height attribute and w-8 sets the element's width to 8 units, the width attribute is represented by the “w” prefix. Moreover, the cursor-pointer sets the element's cursor property to pointer, which converts the cursor to a pointing hand when the user hovers over it, suggesting that it is clickable. Last but not least, the text-yellow-400 modifies the color attribute of the element's text content to a yellow hue with a brightness value of 400.

Together, the four classes h-8, w-8, cursor-pointer, and text-yellow-400 are used to design the HTML element with a yellow background, a height and width of 8 units, and a cursor that turns into a pointing hand when the mouse is hovered over.

```
"use client";
import { useTheme } from "next-themes";  399 (gzipped: 272)
import { useState, useEffect } from "react";  4.2k (gzipped: 1.8k)
import { MoonIcon, SunIcon } from "@heroicons/react/24/solid";  2.3k (gzipped: 964)

function DarkModeButton() {
  const [mounted, setMounted] = useState(false);

  const { systemTheme, theme, setTheme } = useTheme();

  useEffect(() => setMounted(true), []);

  if (!mounted) return null;

  const currentTheme = theme === "system" ? systemTheme : theme;

  return (
    <div>
      {currentTheme === "dark" ? (
        <SunIcon
          className="h-8 w-8 cursor-pointer text-yellow-400"
          onClick={() => setTheme("light")}
        />
      ) : (
        <MoonIcon
          className="h-8 w-8 cursor-pointer text-gray-900"
          onClick={() => setTheme("dark")}
        />
      )}
    </div>
  );
}

export default DarkModeButton;
```

Figure 30. Toggle dark theme code.

Furthermore, the code in Figure 30 also defines a DarkModeButton React component that toggles the website's color theme between light and dark mode.

The component utilizes the useTheme hook from the next-themes library to retrieve the current theme and set it to "light" or "dark" depending on the user's option. While the theme cannot be decided on the server, some of the data returned by the useTheme function are undefined until they are loaded on the client. Hence, the useState hook is utilized to keep track of whether the component is mounted. Upon component mounting, the useEffect hook is used to set the mounted state to true. The component renders a SunIcon or a MoonIcon from the @heroicons/react library conditionally, depending on whether the current theme is "dark" or "light." The setTheme method is invoked with the new theme when the icon is clicked by the user ("light" or "dark").

```
"use client";

import { ThemeProvider } from "next-themes";  4k (gzipped: 1.7k)

function Provider({ children }: { children: React.ReactNode }) {
  return (
    <ThemeProvider enableSystem={true} attribute="class">
      {children}
    </ThemeProvider>
  );
}

export default Provider;
```

Figure 31. Provider component for wrapping the entire component.

Moreover, to change the theme globally, the Provider component, which acts as a wrapper for the ThemeProvider component from next-themes, is required, as shown in Figure 31. The Provider component receives a single React.ReactNode-typed prop called children. This property is used to render the child components that ThemeProvider will wrap. The enableSystem property of ThemeProvider is set to true, allowing the application to utilize the system's chosen color scheme. The property prop is also set to "class," indicating that the ThemeProvider will add a data-theme attribute to the HTML body element reflecting the current color scheme.

```
tailwind.config.js > <unknown>
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./pages/**/*.{js,ts,jsx,tsx}",
    "./components/**/*.{js,ts,jsx,tsx}",
    "./app/**/*.{js,ts,jsx,tsx}",
  ],
  darkMode: "class",
  theme: {
    extend: {},
  },
  plugins: [require("@tailwindcss/line-clamp")],  1.3k (gzipped: 613)
};
```

Figure 32. Add class strategy in Tailwind config file.

Besides, use the class strategy rather than the media approach to allow manually toggling dark mode rather than relying on the choice of the operating system, as shown in Figure 32.

```
<body className="bg-gray-50 dark:bg-zinc-900 transition-all duration-700">
```

Figure 33. Apply dark theme in the application body.

Finally, the application of dark classes, which used to depend on the prefers-color-scheme attribute, has been modified. These classes will now be applied whenever the dark class appears earlier in the HTML tree, regardless of the prefers-color-scheme attribute, an example of applying dark in the HTML body can be found in Figure 33.

4 Conclusion

The purpose of this thesis was to create an internet application utilizing Next.js 13 that presents global news in real time. To achieve this objective and satisfy the criteria, a web application was built with Next.js 13, React, Typescript, and Tailwind CSS as the frontend technology stack and a StepZen-generated GraphQL API as the backend technology. The project was successfully deployed as a production project to Vercel after the testing and development phases. Thus, this thesis focuses on the different potential and practical applications of Next.js 13 in software development.

This thesis implements a web application for searching and displaying live news, checking news according to different categories, and switching between light and dark themes. The author illustrates the frontend development process of the web application by illustrating the impact of Next.js 13, React, Typescript, and Tailwind CSS. As a result of development, the components of the project are reliable and can be simply maintained and expanded for future expansion. In addition, the author includes instructions for constructing an out-of-the-box GraphQL API using StepZen, which may significantly increase the development productivity of front-end developers and decrease back-end development costs and time. In the future, the author will continue to maintain and expand additional features.

In conclusion, the project met its criteria and initial objective. Also, this thesis clearly highlights the numerous advantages of using Next.js 13 in the development process. Finally, this thesis gives application guidance for front-end developers who seek to construct online apps with the latest version of Next.js 13.

References

Sonny Sangha. (2023). Build a live news app with Next.js 13. [online] Available at: https://www.youtube.com/watch?v=QcEY72FX9go

Graphql.org. (2012). *GraphQL: A query language for APIs.* [online] graphql.org. Available at: https://graphql.org/.

Nextjs.org. (n.d.). *API Routes: Introduction | Next.js*. [online] nextjs.org. Available at: https://nextjs.org/docs/api-routes/introduction.

Nextjs.org. (n.d.). *Basic Features: Pages | Next.js*. [online] nextjs.org. Available at: https://nextjs.org/docs/basic-features/pages#static-generation.

Node.js Foundation (2014). *About | Node.js*. [online] Node.js. Available at: https://nodejs.org/en/about/.

Node.js Foundation (n.d.). *Introduction to Node.js*. [online] *Node.js*. Introduction to Node.js. Available at: https://nodejs.dev/en/learn/.

Reactjs.org. (2018). *Introducing Hooks – React*. [online] Available at: https://reactjs.org/docs/hooks-intro.html.

Reactjs.org. (n.d.). *Introducing JSX – React*. [online] Available at: https://reactjs.org/docs/introducing-jsx.html#gatsby-focus-wrapper.

Reactjs.org. (n.d.). *Virtual DOM and Internals – React*. [online] Available at: https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom.

Stepzen.com. (n.d.). *StepZen: Deliver GraphQL faster and scale seamlessly with GraphQL-as-a-Service.* [online] Available at: https://stepzen.com/why-stepzen [Accessed 7 Mar. 2023].

Vercel, Inc. (2023). *Learn | Next.js*. [online] nextjs.org. Available at: https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs.

Vercel.com. (n.d.). *Introducing Turbopack: Rust-based successor to Webpack – Vercel*. [online] nextjs.org. Available at: https://vercel.com/blog/turbopack [Accessed 5 Mar. 2023].

W3schools.com. (n.d.). *React Components*. [online] www.w3schools.com. Available at: https://www.w3schools.com/react/react_components.asp.

W3schools.com. (n.d.). *TypeScript Introduction*. [online] www.w3schools.com. Available at: https://www.w3schools.com/typescript/typescript_intro.php.