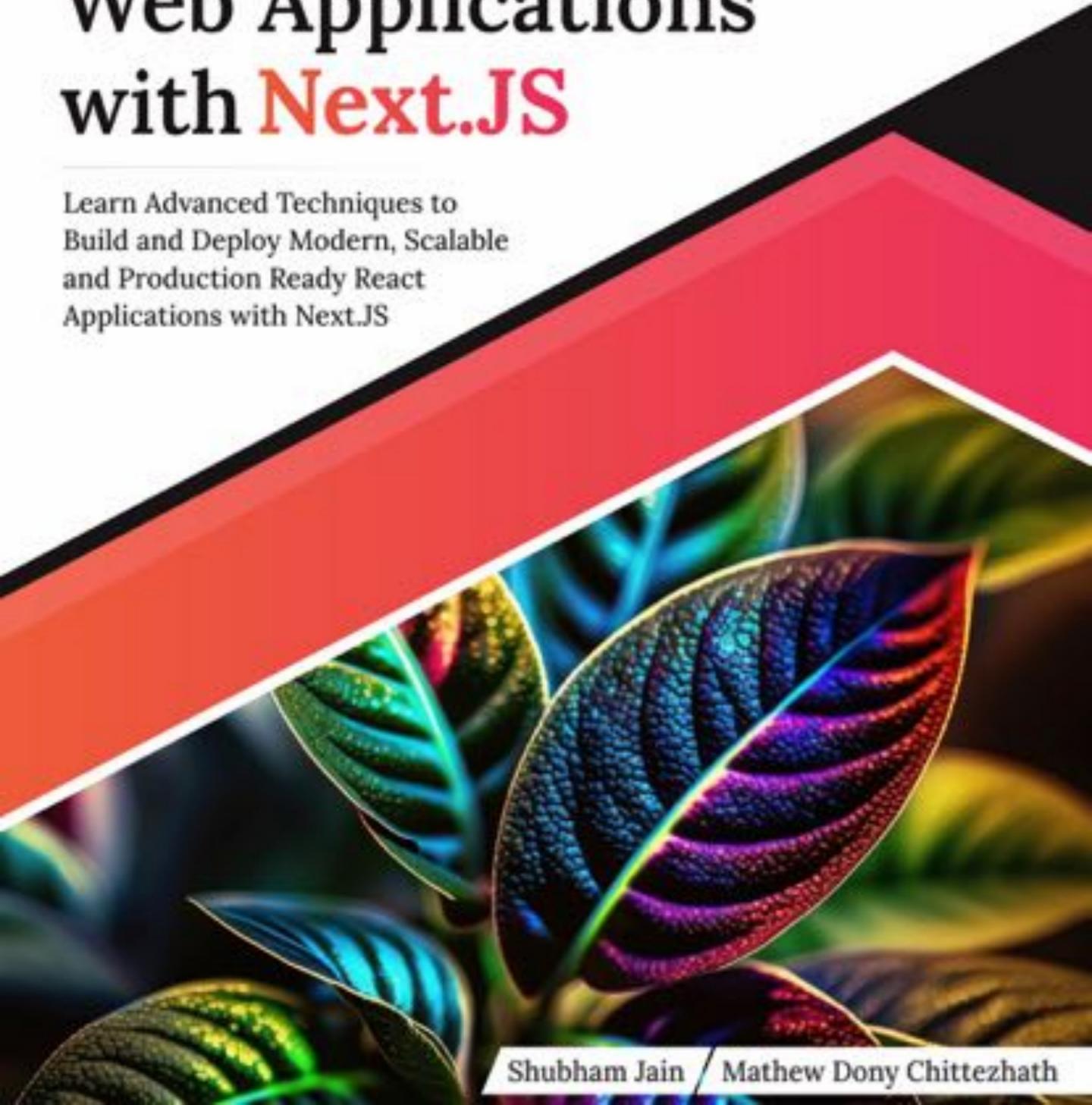




Modern Web Applications with **Next.JS**

Learn Advanced Techniques to
Build and Deploy Modern, Scalable
and Production Ready React
Applications with Next.JS



Shubham Jain

Mathew Dony Chittezhath

Aplicaciones web modernas con Next.JS

Aprenda técnicas avanzadas para crear e implementar aplicaciones React modernas, escalables y listas para producción con Next.JS

Shubham Jain

Mateo Dony Chittezhath



www.orangeava.com

Derechos de autor © 2023, Orange Education Pvt Ltd, AVA™

Todos los derechos reservados. Ninguna parte de este libro podrá reproducirse, almacenarse en un sistema de recuperación de datos ni transmitirse en ninguna forma ni por ningún medio sin la autorización previa por escrito del editor, excepto en el caso de citas breves incluidas en artículos críticos o reseñas.

En la preparación de este libro se ha hecho todo lo posible para garantizar la precisión de la información presentada. Sin embargo, la información contenida en este libro se vende sin garantía, ni expresa ni implícita. Ni el autor ni Orange Education Pvt Ltd ni sus distribuidores serán responsables de ningún daño causado o presuntamente causado, directa o indirectamente, por este libro.

Orange Education Pvt Ltd se ha esforzado por proporcionar información sobre las marcas comerciales de todas las empresas y productos mencionados en este libro mediante el uso adecuado del capital. Sin embargo, Orange Education Pvt Ltd no puede garantizar la exactitud de esta información. El uso de nombres descriptivos generales, nombres registrados, marcas comerciales, marcas de servicio, etc. en esta publicación no implica, incluso en ausencia de una declaración específica, que dichos nombres estén exentos de las leyes y regulaciones de protección pertinentes y, por lo tanto, sean de libre uso general.

Primera publicación: noviembre de 2023

Publicado por: Orange Education Pvt Ltd, AVA™

Dirección: 9, Daryaganj, Delhi, 110002

ISBN: 978-93-88590-97-6

www.orangeava.com

Dedicado a

A aquellos que han dado forma a mi trayectoria: a mi familia, por su amor incondicional; a mis amigos, por su apoyo inquebrantable; a mis mentores, por su sabiduría orientadora; y a la comunidad de ingeniería de software.

Mateo Dony Chittezhath

A los soñadores y constructores, quienes dan vida a las líneas de código y convierten las ideas en realidad digital. Este libro está dedicado a las mentes apasionadas y al espíritu incansable que impulsan la innovación en el mundo del desarrollo web. A quienes ven los desafíos como oportunidades y disfrutan de la elegante armonía entre la tecnología y la creatividad.

En particular, te dedico este libro a ti, querido lector. Tu curiosidad y tu compromiso con la maestría inspiran la esencia misma de estas páginas. Que el conocimiento que contiene te impulse a superar los límites de lo posible con Next.js. Brindemos por las infinitas posibilidades que se abren cuando programamos con propósito, aprendemos con entusiasmo y creamos con el corazón.

Mis amados padres:

Sri Pushpendra Jain

Anjana Jain

y

Mi esposa Riddhi Vanawat

Shubham Jain

Acerca de los autores

Shubham Jain: Shubham Jain, ingeniero de software full-stack con amplia experiencia, se especializa en desarrollo e implementación web integral. Posee una maestría en Tecnologías de la Información, lo que demuestra su dedicación al aprendizaje y a la experiencia. Con una profunda pasión por la tecnología y una dedicación a la creación de software centrado en el usuario, se mantiene a la vanguardia del panorama tecnológico en constante evolución, ofreciendo constantemente experiencias de usuario excepcionales.

Mathew Dony Chittezhath: ingeniero de software full-stack con más de 5 años de experiencia, Mathew Dony es un experto en React, Next.js, Typescript y Node.js.

Habiendo completado una Maestría en Tecnología de la Información de la Universidad de Swinburne, Australia, tiene una gran pasión por la tecnología y le gusta mantenerse actualizado con los últimos desarrollos en el mundo de la tecnología y disfruta desarrollando software que pueda ofrecer una experiencia alegre a todos los humanos que los utilizan.

Acerca de los revisores técnicos

Gaurav Patel trabaja como ingeniero de software senior (más de 8 años) y participa en el desarrollo y mantenimiento de proyectos de varios tamaños para mejorar las capacidades y la eficiencia, además de utilizar mis habilidades en una industria técnica con margen para aprender y crecer.

Trabajé con servicios web y los implementé en proyectos y se implementó la interfaz de usuario.

El marco de JavaScript JQuery se utiliza ampliamente para crear aplicaciones web basadas en Ajax.

Sus puntos fuertes son la buena comunicación y la capacidad de aprender rápidamente.

Especialidades: Javascript, Angular JS/2/4/5/6, React JS, Redux, MySQL, Apache, Jquery, Servicios Web, HTML5, CSS3, Node JS, Express JS, Java Core, Docker, Kubernetes, GitLab, Spring Boot, Java, .NET, SQL, Base de datos, Diseño, Agile, API, Pruebas, AWS, CI/CD, NoSQL, Python.

Supreet Sethi es un profesional experimentado con más de 10 años de experiencia en JavaScript y React. Como líder tecnológico dedicado, Supreet se centra en el desarrollo web, creando aplicaciones SaaS avanzadas con la metodología Agile Scrum. Sus habilidades técnicas abarcan React.js, React Native, Redux y Next.js, demostrando no solo su capacidad de ejecución, sino también una perspectiva experta en el sector. Más allá de la tecnología, el liderazgo de Supreet destaca al guiar con éxito equipos y proyectos. Sus excelentes habilidades interpersonales, pensamiento crítico y enfoque orientado a objetivos crean un entorno donde la innovación prospera. Además, Supreet destaca como profesor de JavaScript, simplificando conceptos complejos y aportando perspectivas prácticas de situaciones reales. Ya sea liderando un equipo, diseñando estructuras complejas o formando a la próxima generación de desarrolladores, Supreet Sethi encarna un enfoque integral hacia la excelencia en habilidades técnicas, liderazgo y formación.

Expresiones de gratitud

Shubham Jain: Estoy profundamente agradecido a las diversas personas y organizaciones que han desempeñado un papel fundamental en la creación de este libro. Su apoyo y aliento incondicionales han sido fundamentales para que este proyecto se haga realidad.

Ante todo, quiero expresar mi más sincero agradecimiento a mis padres por su inquebrantable apoyo y aliento durante todo el proceso de escritura. Su fe en mí ha sido una fuente constante de motivación, y no habría podido completar este libro sin su fe inquebrantable.

Quisiera expresar mi sincero agradecimiento a las instituciones educativas y empresas que brindaron un apoyo invaluable durante mi aprendizaje del web scraping y el dominio de las herramientas asociadas. Sus contribuciones fueron fundamentales para el desarrollo de este libro.

Estoy particularmente agradecido por el apoyo incondicional que recibí de varias personas cuya guía y asistencia marcaron una diferencia significativa. Agradezco especialmente al Sr. Vibhu Bansal por su meticulosa revisión técnica y su valiosa...

ideas que mejoraron enormemente la calidad de este libro.

También quisiera agradecer al increíble equipo que me apoyó durante este proyecto. Su inquebrantable apoyo, paciencia y comprensión, especialmente al concederme el tiempo para completar la primera parte del libro y permitir su publicación en varias secciones, fueron esenciales. Dada la vasta y dinámica naturaleza del procesamiento de imágenes como campo de investigación, fue fundamental explorar diversas áreas problemáticas de forma exhaustiva sin abrumar al lector con un trabajo excesivamente voluminoso.

Una vez más, gracias a todos los que han contribuido a este libro de diversas maneras. Su apoyo ha sido invaluable, y agradezco profundamente la confianza y el aliento que me han brindado a lo largo de este camino.

Mathew Dony Chittezhath: Ante todo, quiero expresar mi más sincero agradecimiento a las mentes innovadoras detrás de React, el equipo de Facebook, quienes no solo transformaron el panorama del desarrollo front-end, sino que también fomentaron una comunidad dinámica y solidaria. Su visión y compromiso incansable han impulsado a innumerables desarrolladores, incluyéndome a mí, a crear aplicaciones intuitivas y escalables.

Un agradecimiento especial a la vasta comunidad de React: desarrolladores, mentores, educadores y entusiastas. Sus contribuciones de código abierto, sus útiles tutoriales y sus valiosos comentarios han sido la base de este libro. Es el colectivo...

sabiduría y experiencia de esta comunidad que ha hecho que comprender y dominar React sea alcanzable y agradable.

A mis amigos y familiares, gracias por su inquebrantable apoyo y paciencia durante este camino. Escribir un libro no es tarea fácil, y su comprensión, ánimo y café de vez en cuando fueron el combustible que necesité para llevar a cabo este proyecto.

Por último, a los lectores: gracias por confiarne su tiempo y esfuerzo para aprender React. Espero que este libro les proporcione los conocimientos y las habilidades necesarias para aprovechar al máximo el potencial de React en sus proyectos. Recuerden siempre que el aprendizaje continuo y la colaboración son clave para el crecimiento en este campo en constante evolución.

Prefacio

Bienvenido a "Aplicaciones Web Modernas con Next..JS". En el mundo del desarrollo web, en constante evolución, mantenerse a la vanguardia tecnológica es fundamental. Next.js, un potente framework basado en React, ha ganado una inmensa popularidad para crear aplicaciones web robustas y eficientes.

Este libro está diseñado para guiarte en un recorrido completo por Next.js y JavaScript, brindándote los conocimientos y las habilidades necesarias para desarrollar aplicaciones web modernas y de alto rendimiento. Tanto si eres un desarrollador experimentado que busca ampliar sus habilidades como si te estás iniciando en el mundo del desarrollo web, este libro te proporcionará las herramientas para destacar en tus proyectos de aplicaciones web.

Capítulo 1: Introducción a las aplicaciones web con Next.js y JavaScript

Lo que aprenderá: Una descripción general de las aplicaciones web, Next.js y JavaScript en el contexto del desarrollo web.

Capítulo 2: Recordar Reaccionar

Lo que aprenderá: Una revisión de los conceptos y principios clave de React para prepararse para el desarrollo de Next.js.

Capítulo 3: Fundamentos de Next.js

Lo que aprenderá: Principios básicos y fundamentos de Next.js, incluido el enrutamiento y la representación del lado del servidor.

Capítulo 4: Nueva versión de Next.js - Conceptos básicos

Lo que aprenderá: Explorar la última versión de Next.js y sus conceptos centrales para crear aplicaciones web modernas.

Capítulo 5: Optimización de aplicaciones Next.js

Lo que aprenderá: Técnicas y estrategias para optimizar las aplicaciones Next.js para el rendimiento y la eficiencia.

Capítulo 6: Entendiendo el enrutamiento en Next.js

Lo que aprenderá: comprensión profunda del enrutamiento en Next.js y cómo navegar entre páginas.

Capítulo 7: Gestión de estados en Next.js

Lo que aprenderá: Implementar soluciones de gestión de estados en aplicaciones Next.js para el manejo de datos.

Capítulo 8: Implementación de API Restful y GraphQL

Lo que aprenderá: Implementar API RESTful y GraphQL en aplicaciones Next.js para la recuperación y manipulación de datos.

Capítulo 9: Uso de diferentes tipos de bases de datos

Lo que aprenderás: Trabajar con varios tipos de bases de datos e integrarlas en aplicaciones Next.js.

Capítulo 10: Renderizado del lado del cliente y del lado del servidor en Next.js

Lo que aprenderá: comprender la renderización del lado del cliente y del servidor y sus aplicaciones en Next.js.

Capítulo 11: Protección de aplicaciones con Next Auth

Lo que aprenderá: Implementar mecanismos de autenticación y autorización en aplicaciones Next.js usando Next Auth.

Capítulo 12: Desarrollo de una aplicación CRUD con Next.js

Lo que aprenderás: Construir una aplicación CRUD (Crear, Leer, Actualizar, Eliminar) desde cero usando Next.js.

Capítulo 13: Arquitectura de implementación

Lo que aprenderá: Explorar arquitecturas de implementación y estrategias para implementar aplicaciones Next.js en entornos de producción.

Nos entusiasma emprender este viaje de aprendizaje contigo. Cada capítulo está cuidadosamente diseñado para brindarte una comprensión profunda de Next.js y sus aplicaciones prácticas. Al finalizar este libro, tendrás los conocimientos y la confianza para crear aplicaciones web modernas que satisfagan las demandas del panorama digital actual. ¡Comencemos!

Descarga de los paquetes de códigos y las imágenes en color

Por favor siga el enlace para descargar el
Paquetes de códigos del libro:

<https://github.com/OrangeAVA/Aplicaciones web modernas con Next.JS>

Los paquetes de códigos y las imágenes del libro también están alojados en
<https://rebrand.ly/546338>

En caso de que haya una actualización del código, se actualizará en el repositorio de
GitHub existente.

Errata

En Orange Education Pvt Ltd, nos enorgullecemos enormemente de nuestro trabajo y seguimos las mejores prácticas para garantizar la precisión de nuestro contenido y ofrecer una experiencia de lectura placentera a nuestros suscriptores. Nuestros lectores son nuestros espejos, y utilizamos sus aportaciones para reflexionar y mejorar los errores humanos, si los hubiera, que pudieran haberse producido durante los procesos de publicación. Para mantener la calidad y ayudarnos a contactar a los lectores que puedan tener dificultades debido a errores imprevistos, escríbanos a:

erratas@orangeava.com

Su apoyo, sugerencias y comentarios son muy apreciados.

SABÍAS

¿Sabías que Orange Education Pvt Ltd ofrece versiones electrónicas de todos los libros publicados, con archivos PDF y ePub disponibles? Puedes actualizar a la versión electrónica en www.orangeava.com y, como cliente de la versión impresa, tienes derecho a un descuento en la copia electrónica. Contáctanos en: info@orangeava.com para más información.

En www.orangeava.com, también puede leer una colección de artículos técnicos gratuitos, suscribirse a una variedad de boletines gratuitos y recibir descuentos y ofertas exclusivas en libros y libros electrónicos de AVA™.

PIRATERÍA

Si encuentra copias ilegales de nuestras obras en internet, en cualquier formato, le agradeceríamos que nos proporcionara la dirección o el nombre del sitio web. Por favor, contáctenos en info@orangeava.com con el enlace al material.

¿ESTÁS INTERESADO EN ESCRIBIR CON NOSOTROS?

Si hay un tema en el que usted es experto y está interesado en escribir o contribuir a un libro, escríbanos a business@orangeava.com.

com. Nos dedicamos a ayudar a desarrolladores y profesionales de la tecnología a comprender mejor los avances tecnológicos y las innovaciones globales, y a construir una comunidad que cree que el conocimiento se adquiere mejor compartiendo y aprendiendo con otros. Contáctanos para conocer las demandas de nuestro público y cómo puedes formar parte de esta reforma educativa. También agradecemos las ideas de expertos en tecnología y les ayudamos a crear contenido de aprendizaje y desarrollo para sus áreas de especialización.

RESEÑAS

Por favor, deja una reseña. Una vez que hayas leído y usado este libro, ¿por qué no dejas una reseña en el sitio web donde lo compraste? Los lectores potenciales podrán ver y usar tu opinión imparcial para tomar decisiones de compra. En Orange Education nos encantaría saber qué opinas de nuestros productos, y nuestros autores pueden aprender de tus comentarios. ¡Gracias!

Para obtener más información sobre Orange Education, visite www.orangeava.com.

Tabla de contenido

1. Introducción a las aplicaciones web con Next.js y JavaScript	1
Introducción	1
Estructura	1
Aplicaciones web y sus componentes básicos	2
Definición de Next.js	6
Casos de uso de Next.js	7
Razones para usar Next.js para el desarrollo de aplicaciones web	8
Conceptos básicos de JavaScript para Next.js	10
Variables y tipos de datos	10
Declaraciones de flujo de control	12
Funciones	13
Clases	14
Módulos	15
Promesas	16
Funciones de flecha	16
Desestructuración	17
Operador de propagación	17
Asíncrono/Espera	18
Literales de plantilla	18
Programación Orientada a Objetos (POO)	19
Comprensión de la representación del lado del servidor (SSR) y la representación del lado del cliente (CSR)	20
Renderizado del lado del servidor	20
Representación del lado del cliente	21
Renderizado híbrido	22
Configuración de un entorno de desarrollo para Next.js	24
Creación de una aplicación Next.js sencilla	27
Conclusión	29
2. Recordar Reaccionar.....	30
Introducción	30

Estructura	30
Introducción a React	31
El DOM virtual y sus beneficios	31
Arquitectura basada en componentes en React	32
Comprensión de los componentes funcionales y de clase	33
Métodos del ciclo de vida de los componentes de React y su uso	36
Sintaxis JSX y sus diferencias con el HTML tradicional	41
Reglas y mejores prácticas de sintaxis JSX	41
Incrustar expresiones JavaScript en JSX	42
Manejo de renderizado condicional y bucles en JSX	43
Diferencias entre JSX y HTML	44
Manejo de eventos en React y paso de datos entre componentes	45
Conceptos básicos del manejo de eventos de React	46
Eventos sintéticos y agrupación de eventos	46
Vinculación de controladores de eventos a componentes	47
Pasando datos a través de props	48
Uso de funciones de devolución de llamada para la comunicación entre padres e hijos Elevación del estado y gestión del estado	49
compartido Estado y propiedades de React	51
Introducción a React Hooks y su uso	55
Otros ganchos integrados y sus casos de uso	60
Programación asíncrona en JavaScript y su aplicación en React	61
Conclusión	65
Preguntas de opción múltiple	65
Respuestas	68
3. Fundamentos de Next.js	69
Introducción	69
Estructura	69
Presentación del framework Next.js y sus ventajas	69
Ventajas de usar Next.js	70
Comparación con otros marcos	71
Casos de uso reales de Next.js	72

Instalación y creación de un nuevo proyecto Next.js	73
Requisitos previos para la instalación de Next.js	73
Instalación de Next.js	74
Creación de su primer proyecto Next.js	74
Entendimiento de la configuración inicial	75
Entendimiento de la estructura de carpetas de un proyecto Next.js	76
Descripción general de la estructura de carpetas	76
Exploración del directorio 'pages'	76
Exploración del directorio public	77
Exploración del directorio de estilos	77
Otros archivos	78
Entendimiento del rol de las páginas en Next.js	78
Introducción a las páginas	79
Creación y renderizado de una página básica en Next.js	81
Implementación de estilos CSS en Next.js usando módulos CSS	85
Conclusión	88
Preguntas de opción múltiple	88
Respuestas	90
4. Next.js 13	91
Introducción	91
Estructura	91
Configuración de una aplicación Next.js 13	92
Enrutador de aplicaciones	94
Componentes de cliente y servidor	95
Enrutamiento	98
Renderizado	99
Obtención de datos	99
Conclusión	100
Preguntas de opción múltiple	101
Respuestas	103
5. Optimización de aplicaciones Next.js	104
Introducción	104
Estructura	104

Importancia y beneficios de optimizar las aplicaciones Next.js	105
Aregar metadatos a páginas usando el componente head	105
Implementar el servicio de archivos estáticos en Next.js	108
Comprender el uso del componente de imagen de Next.js para la optimización de imágenes	111
Comprender la arquitectura de Next.js y cómo funciona	117
Configurar Next.js para un rendimiento óptimo	120
Implementar el almacenamiento en caché del lado del servidor	121
División de código e importaciones dinámicas	123
Almacenamiento en caché y mejora de la obtención de datos	124
Analizar y reducir el tamaño del paquete	126
Estrategias de implementación y mejores prácticas	127
Monitoreo del rendimiento	129
Conclusión	131
Preguntas de opción múltiple	131
Respuestas	132
6. Entendiendo el enrutamiento en Next.js	133
Introducción	133
Estructura	133
Comprender el rol del enrutador Next.js	133
Comprender el componente Link de Next.js y su uso	136
Navegar entre páginas en Next.js usando el enrutador	140
Trabajar con rutas dinámicas en Next.js	140
Conclusión: Descubriendo la belleza del enrutamiento de Next.js	146
Preguntas de opción múltiple	147
Respuestas	148
7. Gestión de estados en Next.js	149
Introducción	149
Estructura	150
Introducción a la gestión de estados en Next.js y su importancia	150
Diferentes opciones de gestión de estados disponibles en Next.js	151
Pros y contras de las opciones de gestión de estados	154
Implementación de la gestión de estados con React state y el uso de hooks	155

Mejores prácticas para gestionar el estado en aplicaciones Next.js	158
Implementación de la gestión de estados usando Redux en la aplicación Next.js	160
Flujo	160
Combinando Redux y Flux	161
Redux Thunk	168
Implementación de la gestión de estados usando el contexto de React en una aplicación simple de Next.js	171
Estudios de casos y ejemplos	178
Conclusión	180
Preguntas de opción múltiple	181
Respuestas	182
8. Implementación de API Restful y GraphQL	183
Introducción	183
Estructura	183
Introducción a las API y su importancia en el desarrollo web moderno	184
Protocolos y arquitecturas API	185
API RESTful versus API GraphQL	186
Configuración de una API RESTful en Next.js	187
Configuración de una API GraphQL en Next.js	191
Integración de los puntos finales de la API con el lado del cliente en Next.js	199
Manejo de errores y excepciones en llamadas API	211
Mejores prácticas para la seguridad y autenticación de API en aplicaciones Next.js	214
Conclusión	216
Preguntas de opción múltiple	217
Respuesta:	218
9. Uso de diferentes tipos de bases de datos.....	219
Introducción	219
Estructura	220
Descripción rápida del sistema de gestión de bases de datos	220
Sistemas de gestión de bases de datos relacionales	224

Sistemas de administración de bases de datos NoSQL	225
Configuración de una conexión de base de datos en Next.js	227
Operaciones CRUD con la base de datos seleccionada	232
Manejo de errores de base de datos y técnicas de depuración	237
Mejores prácticas de seguridad de bases de datos en Next.js	242
Modelado de datos y diseño de esquemas	244
Escalado de la base de datos para rendimiento y alta disponibilidad	253
Conclusión	254
Preguntas de opción múltiple	255
Respuestas	257
10. Comprensión de la representación en aplicaciones Next.js	258
Introducción	258
Estructura	259
Comprensión de la representación en Next.js	259
Beneficios y desventajas de CSR y SSR	263
Enfoque de Next.js para CSR y SSR	264
SSR con Next.js	266
Representación del lado del cliente a CSR con Next.js	269
Representación dinámica del lado del cliente	272
Mejores prácticas para usar CSR y SSR en aplicaciones Next.js	279
Conclusión	281
Preguntas de opción múltiple	281
Respuestas	282
11. Protección de aplicaciones con Next Auth	283
Estructura	284
Introducción a la autenticación y la seguridad	284
Descripción general de Next Auth	287
Configuración de Next Auth en una aplicación Next.js	291
Implementación de diferentes proveedores de autenticación	297
Protección de páginas y rutas API con autenticación	302
Mejores prácticas para autenticación y seguridad en aplicaciones	
Next.js	306
Conclusión	308

Preguntas de opción múltiple	308
Respuestas	309
12. Desarrollo de una aplicación CRUD con Next.js.....	310
Introducción	310
Estructura	311
Configuración de su entorno de desarrollo	311
Visualización de elementos de tareas pendientes (Leer)	316
Configuración de la base de datos con Supabase	324
Adición de nuevos elementos de tareas pendientes (Crear)	331
Edición de elementos de tareas pendientes (Actualizar)	336
Eliminación de elementos de tareas pendientes (Eliminar)	
340 Implementación de la aplicación	344
Conclusión	345
13. Exploración de la arquitectura de implementación en aplicaciones Next.js	346
Estructura	347
Comprensión del proceso de implementación en Next.js	348
Configuración de variables de entorno para la implementación	351
Implementación de aplicaciones Next.js en diferentes plataformas de alojamiento	356
Optimización del proceso de implementación para implementaciones más rápidas y eficientes	363
Configuración de la aplicación Next.js para producción	365
Configuración de canalizaciones (CI/CD) para aplicaciones Next.js	367
Supervisión y depuración de aplicaciones implementadas	370
Conclusión	373
Preguntas de opción múltiple	374
Respuestas	375
Índice	376-387

Capítulo 1

Introducción a las aplicaciones web con Next.js y JavaScript

Introducción

¡Bienvenido al mundo de las aplicaciones web! En este capítulo, exploraremos cómo crear aplicaciones web robustas y de alto rendimiento con Next.js y React. Tanto si eres un desarrollador web experimentado como si estás empezando, este libro te ofrecerá una introducción completa al apasionante mundo del desarrollo web. Next.js es un potente framework para crear aplicaciones React renderizadas del lado del servidor. Al combinar la potencia de React con la simplicidad y facilidad de uso de Next.js, podemos crear aplicaciones web rápidas y escalables. ¡Comencemos y creemos juntos increíbles aplicaciones web!

Estructura

En este capítulo se tratarán los siguientes temas:

- ¿Qué son las aplicaciones web?
- ¿Qué es Next.js y por qué está ganando popularidad?
- Características y beneficios de usar Next.js para crear sitios web dinámicos aplicaciones
- Una revisión de los fundamentos de JavaScript, incluidos los tipos de datos, las estructuras de control, las funciones y los objetos.
- Conceptos avanzados de JavaScript, como programación asíncrona, promesas y características de ES6

- Cómo usar Next.js para crear React renderizado del lado del servidor y del lado del cliente aplicaciones
- Cómo crear una aplicación Next.js sencilla en tu computadora

Aplicaciones web y sus componentes básicos

El desarrollo web es el proceso de crear sitios web y aplicaciones web.

Una aplicación web es un programa que se ejecuta en un servidor web y al que se accede a través de un navegador. Los tres pilares fundamentales del desarrollo web son HTML, CSS y JavaScript.

Diferencia entre sitios web y aplicaciones web

Los términos sitio web y aplicación web se suelen usar indistintamente, pero presentan algunas diferencias distintivas. Un sitio web es un conjunto de páginas web estáticas que proporcionan información o contenido a los visitantes. Los sitios web suelen estar diseñados para ser navegados por los visitantes, que son consumidores pasivos del contenido. Algunos ejemplos de sitios web son blogs, sitios de noticias y páginas de inicio de empresas. Por otro lado, una aplicación web es un programa de software al que se accede a través de un navegador web y que proporciona funcionalidad interactiva a los usuarios. Las aplicaciones web son más complejas que los sitios web y requieren la entrada e interacción del usuario para funcionar. Algunos ejemplos de aplicaciones web son las plataformas de redes sociales, los mercados en línea y las herramientas de productividad. La principal diferencia entre sitios web y aplicaciones web es el nivel de interactividad y funcionalidad que ofrecen. Mientras que los sitios web se centran principalmente en proporcionar información, las aplicaciones web permiten a los usuarios realizar tareas complejas e interactuar con otros usuarios. Otra diferencia es el nivel de personalización disponible en las aplicaciones web.

Los sitios web generalmente ofrecen una experiencia estandarizada para todos los visitantes, mientras que las aplicaciones web pueden adaptar su funcionalidad y contenido a cada usuario según sus preferencias y comportamiento. En resumen, si bien tanto los sitios web como las aplicaciones web se acceden a través de navegadores web y están alojados en internet, las aplicaciones web ofrecen una experiencia más interactiva y personalizable que los sitios web.

HTML (lenguaje de marcado de hipertexto)

HTML es el lenguaje de marcado estándar para crear páginas web y aplicaciones. Proporciona la estructura y el contenido de las páginas web y aplicaciones mediante la definición de elementos como encabezados, párrafos, imágenes e hipervínculos. Utiliza un lenguaje basado en etiquetas, donde cada etiqueta representa un elemento específico de la página.

Aquí hay un ejemplo de cómo un elemento está definido

```
<!DOCTYPE html>
<html>
<cabeza>
    <title>Mi aplicación web</title>
</cabeza>
<cuerpo>
    ¡Bienvenido a mi aplicación web!
    Mi nombre es Mathew y ¡vamos a aprender Next.js!
</cuerpo>
</html>
```

El código anterior da el siguiente resultado cuando se visualiza en un navegador web:

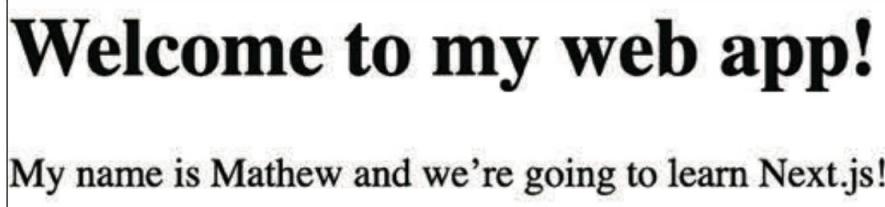


Figura 1.1: Salida del sitio web

En este ejemplo, hemos definido una página básica que incluye un título, un encabezado y un párrafo. La etiqueta `<!DOCTYPE html>` especifica el tipo de documento y la etiqueta `<html>` es el elemento raíz de la página HTML. La etiqueta `<head>` contiene información sobre la página, como el `<title>`, y la etiqueta `<body>` contiene el contenido de la página.

HTML ofrece una amplia gama de elementos que pueden usarse para crear páginas web y aplicaciones. Estos elementos incluyen encabezados (`<h1>` a `<h6>`), párrafos (`<p>`), listas (`` y ``), enlaces (`<a>`), imágenes (``), tablas (`<table>`, `<tr>`, `<td>`), formularios (`<form>`, `<input>`, `<textarea>`, `<button>`) y muchos más.

Hojas de estilo en cascada (CSS)

CSS se utiliza para describir la presentación de páginas web, incluidos colores, fuentes,

y diseño. Se utiliza para dar estilo a los elementos definidos en la página. CSS puede usarse para aplicar estilos a elementos específicos o para aplicarlos globalmente a toda la página web.

Aquí hay un ejemplo de cómo se define una regla CSS

```
h1 {  
    tamaño de fuente: 24px;  
    color: rojo;  
}
```

En el ejemplo anterior, definimos una regla CSS que aplica estilos a todos los elementos h de la página. Establecimos el tamaño de fuente en 24 píxeles y el color en rojo.

Si vinculamos el código CSS anterior a nuestra página HTML, obtenemos el siguiente resultado:

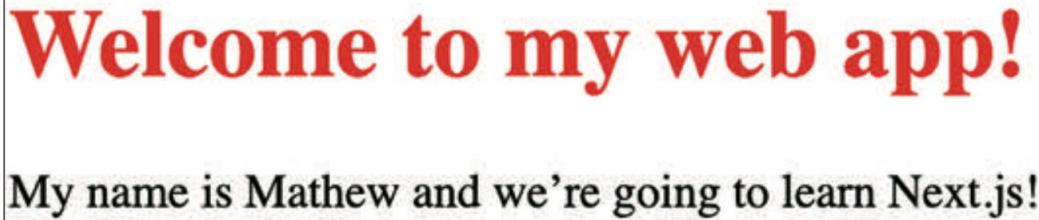


Figura 1.2: Página HTML actualizada

CSS proporciona una amplia gama de propiedades que pueden utilizarse para dar estilo a elementos HTML. Estas propiedades incluyen propiedades de fuente (tamaño de fuente, familia de fuente, peso de fuente, estilo de fuente), propiedades de color (color, color de fondo), propiedades de diseño (margen, relleno, borde, ancho, alto) y muchas más.

CSS también proporciona una amplia gama de selectores que se pueden utilizar para aplicar estilos a elementos específicos de la página. Estos selectores incluyen selectores de etiquetas (h1, p, ul, li), selectores de clase (.my-class), selectores de ID (#my-id), selectores de atributos ([attribute=value]) y muchos más.

JavaScript

JavaScript es un lenguaje de programación de alto nivel que se utiliza para crear páginas web interactivas y dinámicas. Se utiliza para añadir funcionalidad a la web.

páginas, como manejo de eventos, validación de formularios y solicitudes y respuestas de API. JavaScript es ejecutado por el navegador web y se utiliza para interactuar con el HTML y CSS en la página. El código avaScript se puede agregar a los archivos usando el elemento script , que se puede colocar en la sección de encabezado o cuerpo del archivo HTML. Alternativamente, el código avaScript se puede incluir en un archivo separado y vincular al archivo utilizando el atributo src .

Ahora vamos a ampliar nuestro ejemplo original y agregar algo de magia de JavaScript para hacer que nuestra aplicación web sea interactiva:

```
<!DOCTYPE html>
<html>
<cabeza>
    <title>Mi aplicación web</title>
</cabeza>
<cuerpo>
    ¡Bienvenido a mi aplicación web!
    Mi nombre es Mathew y ¡vamos a aprender Next.js!
    ¡Haz clic en mí!
    <guión>
        var button = document.getElementById("miBoton");
        botón.addEventListener("clic", función() {
            alert("¡Guau! ¡Has hecho clic en este botón!");
        });
    </script>
</cuerpo>
</html>
```

En el ejemplo anterior, añadimos un nuevo botón con un detector de eventos onclick que ejecutará una alerta del navegador mediante JavaScript. Al hacer clic en el botón, se generará la siguiente notificación de alerta del navegador.

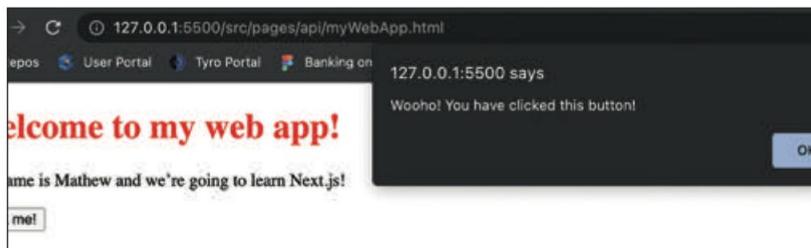


Figura 1.3: Alerta del navegador

Al combinar estos tres bloques fundamentales, podemos crear aplicaciones web potentes e interactivas que brinden una excelente experiencia de usuario.

Definición de Next.js

Next.js es un popular framework de código abierto para crear aplicaciones React renderizadas del lado del servidor (SSR) y generadas por sitios estáticos (SSG). Desarrollado por Vercel, Next.js ofrece un enfoque intuitivo y fácil de usar para el desarrollo de aplicaciones web, permitiéndoles centrarse en crear experiencias de usuario excepcionales sin preocuparse por la infraestructura subyacente.

Con Next.js, los desarrolladores pueden crear aplicaciones web dinámicas y de alto rendimiento que pueden ejecutarse en cualquier servidor o plataforma, gracias a sus capacidades de renderizado universal. Next.js ofrece una variedad de características y beneficios, incluidos

- Representación del lado del servidor (SSR) y generación de sitios estáticos (SSG)

Next.js ofrece potentes funciones de renderizado del lado del servidor que permiten a los desarrolladores crear aplicaciones web dinámicas e interactivas que se cargan rápidamente y ofrecen una excelente experiencia de usuario. Next.js también permite la generación de sitios web estáticos, lo que permite a los desarrolladores generar archivos estáticos durante la compilación que se pueden entregar de forma rápida y eficiente.

- Herramientas integradas y optimización automática

Next.js incluye una gama de herramientas y funciones integradas que facilitan la optimización del rendimiento y la accesibilidad de tu aplicación. Con funciones como la optimización de imágenes y la división automática de código, Next.js ayuda a garantizar que tu aplicación sea rápida, eficiente y accesible para todos los usuarios.

- División y optimización automática de código

Next.js divide automáticamente su código en fragmentos más pequeños y carga solo el código necesario para cada página, lo que mejora el tiempo de carga inicial y reduce el tamaño del paquete de JavaScript.

- Enfoque híbrido

Con Next.js, puedes crear aplicaciones híbridas que combinan la renderización del lado del servidor y la renderización del lado del cliente, lo que te permite aprovechar los beneficios de ambos enfoques.

En general, Next.js es un marco potente y flexible que permite a los desarrolladores crear aplicaciones web escalables y de alta calidad de forma rápida y eficiente.

- Fácil configuración e implementación

Next.js es fácil de configurar e implementar, lo que permite a los desarrolladores comenzar rápidamente y centrarse en el desarrollo de sus aplicaciones. Gracias a su compatibilidad integrada con el alojamiento en Vercel, Next.js ofrece una experiencia de implementación fluida que facilita la implementación y el escalado de su aplicación.

- Apoyo comunitario y ecosistema

Next.js cuenta con una amplia y activa comunidad de desarrolladores y colaboradores que brindan soporte, comparten conocimientos y contribuyen al desarrollo del framework. Además, Next.js cuenta con un sólido ecosistema de plugins, herramientas y recursos que pueden ayudar a los desarrolladores a crear aplicaciones web mejores y más eficientes.

Casos de uso de Next.js

Next.js es adecuado para una amplia gama de casos de uso de desarrollo de aplicaciones web, incluidos:

- Aplicaciones de comercio electrónico
- Sitios web y blogs basados en contenido
- Sitios de redes sociales
- Herramientas y paneles de control basados en la web
- Aplicaciones web progresivas (PWA)
- Aplicaciones móviles que utilizan React Native

Con su flexibilidad, escalabilidad y facilidad de uso, Next.js es un marco versátil que se puede utilizar para crear una amplia gama de aplicaciones web.

En resumen, Next.js es un framework potente, flexible y fácil de usar para crear aplicaciones web de alta calidad con React. Gracias a su compatibilidad integrada con renderizado del lado del servidor y generación de sitios estáticos, división automática de código y...

Gracias a su optimización, implementación y escalabilidad sencillas, Next.js es una opción ideal para desarrolladores que buscan crear aplicaciones web dinámicas, de alto rendimiento y escalables.

Razones para utilizar Next.js para el desarrollo de aplicaciones web

Next.js es un framework popular para crear aplicaciones web con React. Estas son algunas de las razones clave por las que podrías considerar usar Next.js para el desarrollo de tus aplicaciones web:

- Representación del lado del servidor integrada

Una de las principales ventajas de usar Next.js es su compatibilidad integrada con SSR. SSR permite que tu aplicación se renderice en el servidor antes de enviarse al cliente, lo que puede mejorar el tiempo de carga inicial y mejorar el SEO. Con Next.js, no necesitas configurar un servidor independiente ni preocuparte por la gestión del proceso de renderizado del lado del servidor, ya que el framework lo gestiona todo automáticamente.

- División y optimización automática de código

Next.js incluye funciones automáticas de división y optimización de código, que ayudan a reducir el tiempo de carga inicial y a mejorar el rendimiento general de la aplicación. La división de código permite dividir el código en fragmentos más pequeños, que se cargan según demanda, en lugar de todos a la vez. Esto ayuda a reducir la cantidad de JavaScript que el navegador debe descargar y analizar, lo que a su vez acelera el tiempo de carga inicial. Next.js también admite la optimización de imágenes y otras optimizaciones de rendimiento de forma predeterminada.

- Generación sencilla de sitios estáticos

Next.js también ofrece compatibilidad integrada con SSG, lo cual resulta útil para crear sitios web estáticos, blogs y otras aplicaciones basadas en contenido. SSG permite generar páginas HTML durante la compilación, que se pueden entregar directamente al cliente, en lugar de generarse dinámicamente en el servidor. Esto puede mejorar el rendimiento y reducir la carga de la aplicación en el servidor.

- Fácil de configurar e implementar

Next.js es fácil de configurar e implementar gracias a su compatibilidad integrada con el alojamiento en Vercel. Vercel ofrece una experiencia de implementación fluida que facilita la implementación y el escalado de su aplicación, sin tener que preocuparse por la gestión de servidores o infraestructura.

- Comunidad grande y activa

Next.js cuenta con una amplia y activa comunidad de desarrolladores y colaboradores que brindan soporte, comparten conocimientos y contribuyen al desarrollo del framework. Esto ha dado lugar a la creación de numerosos plugins, herramientas y recursos útiles que ayudan a los desarrolladores a crear aplicaciones web mejores y más eficientes.

En resumen, Next.js es un framework potente y versátil para crear aplicaciones web con React. Gracias a su compatibilidad integrada con renderizado del lado del servidor, división y optimización automáticas de código, generación sencilla de sitios estáticos, implementación sencilla y una comunidad amplia y activa, Next.js es la opción ideal para crear aplicaciones web de alta calidad, de alto rendimiento y escalables.

- Experiencia de desarrollador mejorada

Next.js ofrece una excelente experiencia para desarrolladores gracias a sus API y funciones intuitivas y fáciles de usar. El framework incluye compatibilidad integrada con muchas tareas comunes de desarrollo web, como el enrutamiento, la obtención de datos y el estilo, lo que puede ayudar a agilizar el proceso de desarrollo y reducir el tiempo y el esfuerzo necesarios para crear y mantener la aplicación.

- Compatibilidad con TypeScript

Next.js también ofrece compatibilidad integrada con TypeScript, un superconjunto de JavaScript con tipado estático que puede ayudar a mejorar la calidad del código y detectar errores en las primeras etapas del desarrollo. La compatibilidad con TypeScript es especialmente útil para aplicaciones o equipos de gran tamaño, donde el código base puede volverse complejo y difícil de gestionar.

- Opciones flexibles de obtención de datos

Next.js ofrece opciones flexibles de obtención de datos, lo que simplifica el proceso de obtención y gestión de datos en tu aplicación. El framework admite la obtención de datos tanto del lado del servidor como del lado del cliente, así como la regeneración estática incremental, que te permite actualizar tus páginas estáticas con nuevos datos sin tener que reconstruir la página completa.

- Extensible y personalizable

Next.js es altamente extensible y personalizable, lo que le permite agregar sus propios complementos, middleware y opciones de configuración para adaptar el marco a sus necesidades específicas. Esto puede ayudar a mejorar la flexibilidad y escalabilidad de su aplicación y permitirle agregar nuevas características y funcionalidades a medida que su aplicación crece y evoluciona.

- Soporte integrado para React

Finalmente, Next.js ofrece compatibilidad integrada con React, una popular biblioteca de JavaScript para crear interfaces de usuario. React es conocida por su simplicidad, rendimiento y flexibilidad, y se usa ampliamente en la comunidad de desarrollo web. Next.js se integra perfectamente con React, lo que permite crear aplicaciones web potentes y dinámicas con un conjunto de herramientas conocido y popular.

En resumen, Next.js ofrece una gama de beneficios y características que lo convierten en la opción ideal para crear aplicaciones web de alta calidad, de alto rendimiento y escalables. Con su experiencia de desarrollador mejorada, compatibilidad con TypeScript, opciones flexibles de obtención de datos, extensibilidad y compatibilidad integrada con React, Next.js es un framework potente y versátil que puede ayudarte a crear aplicaciones web mejores y más eficientes.

Conceptos básicos de JavaScript para Next.js

Antes de adentrarnos en el desarrollo de Next.js, es importante comprender bien los fundamentos de JavaScript. En esta sección, abordaremos algunos conceptos fundamentales de JavaScript esenciales para el desarrollo de aplicaciones Next.js.

Variables y tipos de datos

Las variables se utilizan para almacenar valores de datos en JavaScript. Hay tres maneras de declarar una variable en JavaScript:

- var
- dejar
- constante

var es la forma antigua de declarar variables en JavaScript y tiene algunas peculiaridades que pueden causar problemas. let y const se introdujeron en JavaScript ES6 y son la forma preferida de declarar variables en JavaScript moderno.

He aquí un ejemplo:

```
// Declarar una variable usando var  
var x = 10;
```

```
// Declarar una variable usando let  
sea y = 20;
```

Introducción a las aplicaciones web con Next.js y JavaScript

```
// Declarar una variable usando const
```

```
constante z = 30;
```

```
// Intentar reasignar un valor a una variable constante dará como resultado un  
error
```

```
// z = 40; // Esto generará un error
```

JavaScript tiene varios tipos de datos integrados, incluidos:

- número (para valores numéricos) •
- cadena (para valores de texto)
- booleano (para valores verdaderos/
falsos) • nulo (para un valor
nulo) • indefinido para un valor indefinido
- objeto (para estructuras de datos complejas)

He aquí un ejemplo:

```
// Declarar una variable numérica
```

```
sea a = 10;
```

```
// Declarar una variable de cadena
```

```
sea b = “¡Hola, mundo!”;
```

```
// Declarar una variable booleana
```

```
sea c = verdadero;
```

```
// Declarar una variable nula
```

```
sea d = nulo;
```

```
// Declarar una variable indefinida
```

```
sea e;
```

```
// Declarar una variable de objeto  
sea f = { nombre: "Mathew", edad: 28 };
```

Declaraciones de flujo de control

Las declaraciones de flujo de control se utilizan para controlar el flujo de ejecución en avaScript.

Las declaraciones de flujo de control más comunes son las siguientes

- instrucción if...else
- bucle for
- bucle while
- declaración de cambio

A continuación se muestran ejemplos de cómo se utiliza cada una de estas afirmaciones:

```
// Declaración if...else
```

```
sea edad = 20;
```

```
si (edad >= 18) {  
    console.log("Eres un adulto.");  
} demás {  
    console.log("Aún no eres un adulto.");  
}
```

```
// Salida: Eres un adulto
```

```
// Bucle for  
para (sea i = 0; i < 5; i++) {  
    consola.log(i);  
}
```

```
// Salida: 0 1 2 3 4
```

```
// Bucle While
```

Introducción a las aplicaciones web con Next.js y JavaScript

```
sea i = 0;
```

```
mientras (i < 5) {  
    consola.log(i);  
    yo++;  
}
```

```
// Producción:  
0 1 2 3 4
```

```
// Sentencia Switch  
sea día = "lunes";
```

```
cambiar (día) {  
    caso "Lunes":  
        console.log("Hoy es lunes.");  
        romper;  
    caso "martes":  
        console.log("Hoy es martes.");  
        romper;  
    por defecto:  
        console.log("Hoy es otro día.");  
        romper;  
}
```

```
// Salida: Hoy es lunes
```

Funciones

Las funciones se utilizan para agrupar un conjunto de sentencias y realizar una tarea específica. En JavaScript, las funciones se pueden declarar mediante la palabra clave `function` o la notación de flecha (`=>`).

He aquí un ejemplo:

```
// Declarar una función usando la palabra clave function
función add(x, y) {
    devuelve x + y;
}

// Declarar una función usando la notación de función de flecha
constante resta = (x, y) => {
    devuelve x - y;
};

// Llamar a las funciones
console.log(add(5, 10)); // Salida: 15
console.log(subtract(20, 5)); // Salida: 15
```

Clases

Las clases se utilizan para crear objetos en JavaScript. En ES6, se introdujeron para facilitar la creación de objetos e implementar la herencia.

He aquí un ejemplo:

```
clase Persona {
    constructor(nombre, edad) {
        este.nombre = nombre;
        esto.edad = edad;
    }

    diHola() {
        console.log(`Hola, mi nombre es ${this.name} y soy ${this.age}`)
    }
}
```

Introducción a las aplicaciones web con Next.js y JavaScript

```
años.');
}

}

// Crea un nuevo objeto Persona
const matthew = new Persona("Mathew", 28);

// Llama al método sayHello
matthew.sayHello(); // Salida: "Hola, mi nombre es Mathew y tengo 28 años".
```

Módulos

Los módulos se utilizan para organizar el código en fragmentos reutilizables. En Next.js, usamos módulos para organizar nuestro código y facilitar su compartición entre las diferentes partes de nuestra aplicación.

He aquí un ejemplo:

```
// Exportar una función desde un archivo math.js
función de exportación add(x, y) {
    devuelve x + y;
}

// Importar la función a otro módulo
importar { agregar } desde "./math.js";

// Llamar a la función
console.log(add(5, 10)); // Salida: 15
```

Promesas

Las promesas se utilizan para gestionar operaciones asíncronas en JavaScript. Son una forma de gestionar las funciones de devolución de llamada de forma más legible y predecible.

A continuación se muestra un ejemplo de cómo crear una Promesa en JavaScript:

```
//Crear una nueva promesa
const promesa = nueva Promesa((resolver, rechazar) => {
    // Simular una operación asíncrona
    establecerTiempo de espera(() => {
        //Resolver la promesa
        resolve("¡Datos recuperados exitosamente!");
    }, 2000);
});

// Llama a la promesa
promesa.then((resultado) => {
    console.log(result); // Salida: "¡Datos recuperados exitosamente!"
});
```

Funciones de flecha

Las funciones de flecha son una forma abreviada de escribir expresiones de función en JavaScript. Sirven para simplificar la sintaxis de las funciones y hacerlas más legibles.

He aquí un ejemplo:

```
// Expresión de función tradicional
constante add = función(x, y) {
    devuelve x + y;
}

// Expresión de función de flecha
```

Introducción a las aplicaciones web con Next.js y JavaScript

```
constante add = (x, y) => x + y;
```

Desestructuración

La desestructuración consiste en extraer valores de objetos o matrices y asignarlos a variables. Es una forma abreviada de escribir asignaciones y permite que el código sea más conciso.

He aquí un ejemplo:

```
// Desestructuración de una matriz
const [primero, segundo, tercero] = [1, 2, 3];
console.log(first); // Salida: 1
```

```
// Desestructuración de un objeto
constante persona = {
    nombre: "Mathew",
    edad: 28
};
const { nombre, edad } = persona;
console.log(nombre); // Salida: "Mathew"
```

Operador de propagación

El operador de propagación permite expandir una matriz u objeto en elementos individuales. Permite combinar varias matrices u objetos en una sola matriz u objeto.

He aquí un ejemplo:

```
// Uso del operador de propagación para combinar matrices
constante arr1 = [1, 2, 3];
constante arr2 = [4, 5, 6];
```

```
const combinada = [...arr1, ...arr2];
console.log(combined); // Salida: [1, 2, 3, 4, 5, 6]

// Uso del operador de propagación para copiar un objeto
constante persona = {
    nombre: "Mathew",
    edad: 28
};
const copia = { ...persona };
console.log(copia); // Salida: { nombre: "Mathew", edad: 28 }
```

Asíncrono/Espera

Syncawait es una nueva función de avaScript que simplifica el trabajo con promesas. Permite escribir código asíncrono con la misma apariencia que el código síncrono, lo que facilita su lectura y depuración.

A continuación se muestra un ejemplo de cómo usar async/await para manejar una promesa en JavaScript:

```
// Uso de async/await para manejar una promesa
función asíncrona getData() {
    constante respuesta = await fetch('https://api.example.com/data');
    const data = await respuesta.json();
    devolver datos;
}

// Llamar a la función asíncrona
const datos = await getData();
console.log(datos);
```

Literales de plantilla

Los literales de plantilla son una forma de escribir cadenas que incluyen variables y expresiones.

Introducción a las aplicaciones web con Next.js y JavaScript

Utilizan comillas invertidas (`) en lugar de comillas y permiten interpolar variables directamente en la cadena.

He aquí un ejemplo:

```
// Uso de literales de plantilla para interpolar una variable
const nombre = "Mathew";
const mensaje = `Hola, ${nombre}!`;
console.log(mensaje); // Salida: "¡Hola, Mathew!"
```

Programación Orientada a Objetos (POO)

La Programación Orientada a Objetos es un paradigma de programación que utiliza objetos para representar entidades del mundo real. Es una forma de organizar el código en componentes modulares y reutilizables.

He aquí un ejemplo:

```
// Usando OOP para crear una clase
clase Animal {
    constructor(nombre, especie) {
        este.nombre = nombre;
        esta.especie = especie;
    }

    hablar() {
        console.log(`${this.name} dice hola!`);
    }
}

// Creando una instancia de la clase Animal
const perro = nuevo Animal("Dottie", "Perro");
```

```
// Llamando al método speak  
perro.hablar(); // Salida: “¡Dottie dice hola!”
```

Al comprender estos conceptos adicionales de JavaScript, podrá escribir código más potente y flexible en sus aplicaciones web Next.js. En los siguientes capítulos, exploraremos cómo aplicar estos conceptos específicamente a la creación de aplicaciones web integrales con Next.js.

Comprender la renderización del lado del servidor (SSR) y renderizado del lado del cliente (CSR)

Al crear aplicaciones web con Next.js, es importante comprender la diferencia entre SSR y CSR. Ambos enfoques tienen sus ventajas y desventajas, y es fundamental elegir el más adecuado para cada caso de uso.

Renderizado del lado del servidor

SSR es la forma tradicional de renderizar páginas web. Con SSR, el servidor envía una página HTML completamente renderizada al cliente en respuesta a una solicitud. Esto significa que el cliente solo necesita descargar y mostrar la página, sin necesidad de renderizar ni obtener datos adicionales.

SSR puede mejorar el tiempo de carga inicial de sus páginas web, ya que el cliente recibe una página completa de inmediato. También puede mejorar el SEO de sus páginas web, ya que los motores de búsqueda pueden rastrear e indexar con mayor facilidad las páginas HTML completamente renderizadas.

A continuación se muestra un ejemplo del uso de SSR con Next.js:

```
// Una página simple de Next.js que usa SSR  
función Página de inicio({ datos }) {  
    devolver (  
        <div>  
            ¡Hola, {data.name}!  
            <p>{datos.descripcion}</p>  
        </div>  
    );
```

Introducción a las aplicaciones web con Next.js y JavaScript

```
}
```

```
exportar función asíncrona getServerSideProps() {  
    // Obtener datos de una API externa  
    constante respuesta = await fetch("https://api.example.com/data");  
    const data = await respuesta.json();  
  
    // Pasar los datos al componente de la página como propiedades  
    devolver { propiedades: { datos } };  
}
```

```
exportar página de inicio predeterminada;
```

En este ejemplo, la función `getServerSideProps` es una función especial en Next.js que le permite obtener datos y pasarlo a su componente de página como propiedades. Cuando el cliente solicita esta página, el servidor ejecutará esta función para obtener los datos y renderizar la página, luego enviará la página HTML completamente renderizada al cliente.

Representación del lado del cliente

CSR es un enfoque más moderno para renderizar páginas web. Con CSR, el cliente descarga una página HTML mínima, obtiene los datos y renderiza la página en el lado del cliente mediante JavaScript.

La CSR puede mejorar la experiencia del usuario de su aplicación web, ya que el cliente puede obtener y renderizar datos dinámicamente, sin necesidad de actualizar la página. También puede reducir la carga del servidor, ya que este solo necesita enviar la página HTML inicial y el cliente puede gestionar la renderización y la obtención de datos posteriores.

A continuación se muestra un ejemplo del uso de CSR con Next.js:

```
// Una página simple de Next.js que usa CSR  
función Página de inicio() {  
    const [datos, setData] = useState(nulo);
```

```
usarEfecto(() => {
    // Obtener datos de una API externa
    función asíncrona fetchData() {
        constante respuesta = await fetch('https://api.example.com/data');
        const data = await respuesta.json();
        setData(datos);
    }
    obtener datos();
}, []);

si (!datos) {
    regresar <div>Cargando...</div>;
}

devolver (
    <div>
        ¡Hola, {data.name}!
        <p>{datos.descripcion}</p>
    </div>
);
}

exportar página de inicio predeterminada;
```

En este ejemplo, usamos ganchos de React para obtener datos y renderizar la página en el lado del cliente. Cuando el cliente solicita esta página, el servidor envía una página HTML mínima y el cliente ejecuta el código JavaScript para obtener los datos y renderizar la página.

Renderizado híbrido

Además de SSR y CSR, Next.js también admite la renderización híbrida, que es una combinación de ambos enfoques. Con la renderización híbrida, puede obtener datos

en el servidor y envía una página parcialmente renderizada al cliente, luego continúa renderizando y obteniendo datos en el lado del cliente.

La renderización híbrida puede ofrecer lo mejor de ambos mundos: el rápido tiempo de carga inicial y los beneficios S de SSR, y la interfaz de usuario dinámica y receptiva de CSR.

Sin embargo, también puede ser más complejo de implementar y puede requerir configuración y optimización adicionales del lado del servidor.

A continuación se muestra un ejemplo del uso de renderizado híbrido con Next.js:

```
// Una página simple de Next.js que utiliza renderizado híbrido
función Página de inicio({ datos }) {
    const [dynamicData, setDynamicData] = useState(null);

    usarEfecto(() => {
        // Obtener datos dinámicos en el lado del cliente
        función asincrona fetchDynamicData() {
            respuesta constante = esperar búsqueda('https://api.example.com/dynamic-da-
ejército de reserva');

            const dynamicData = await respuesta.json();
            setDynamicData(dynamicData);
        }
        obtenerDatosDinámicos(); },
    []);
}

devolver (
    <div>
        ¡Hola, {data.name}!
        <p>{datos.descripcion}</p>
        {dynamicData && <p>{dynamicData.message}</p>}
    </div>
);
```

```
exportar función asíncrona getServerSideProps() {  
    // Obtener datos estáticos de una API externa  
  
    constante staticResponse = await fetch('https://api.example.com/  
datos estáticos');  
  
    const staticData = await staticResponse.json();  
  
    // Pasar los datos estáticos al componente de la página como propiedades  
    devolver { propiedades: { datos: staticData } };  
}  
  
exportar página de inicio predeterminada;
```

En este ejemplo, usamos `getServerSideProps` para obtener datos estáticos en el servidor y enviarlos al cliente como propiedades. También usamos ganchos de React para obtener datos dinámicos en el lado del cliente y actualizar el contenido de la página dinámicamente.

En resumen, al crear aplicaciones web con Next.js, es importante comprender la diferencia entre SSR y CSR, y elegir el enfoque adecuado para cada caso. Next.js también admite el renderizado híbrido, que ofrece las ventajas de ambos enfoques. Al comprender estos conceptos, podrá crear aplicaciones web rápidas, dinámicas y optimizadas para SEO con Next.js.

Configuración de un entorno de desarrollo para Next.js

Configurar un entorno de desarrollo para Next.js es un paso importante para crear aplicaciones web con Next.js. En esta sección, le guiaremos en el proceso de configuración de un entorno de desarrollo para Next.js.

Paso 1: Instalar Node.js y npm

El primer paso para configurar un entorno de desarrollo para Next.js es instalar Node.js y npm. Node.js es un entorno de ejecución de JavaScript que permite ejecutar JavaScript en el servidor, mientras que npm es un gestor de paquetes que permite instalar y administrar las dependencias del proyecto.

Puede descargar e instalar Node.js desde el sitio web oficial en <https://nodejs.org>. Elija la versión adecuada para su sistema operativo y descárguela.

Una vez completada la descarga, siga las instrucciones de instalación para instalar Node.js.

Una vez que haya instalado Node.js, puede verificar la instalación ejecutando los siguientes comandos en su terminal como se muestra en la figura

A screenshot of a terminal window on a dark background. It shows two command-line entries: 'node -v' followed by the output 'v18.16.1', and 'npm -v' followed by the output '9.7.2'. Both commands begin with a green arrow icon and are preceded by a tilde (~) indicating the user's home directory.

Figura 1.4: Verificación de la instalación

Estos comandos deberían mostrar las versiones de Node.js y npm que tenga instaladas en su sistema.

Paso 2: Crea un nuevo proyecto Next.js

Una vez instalado Node.js y npm, puede crear un nuevo proyecto Next.js con la herramienta de línea de comandos `create-next-app`. Esta herramienta configura un nuevo proyecto Next.js con todas las dependencias y archivos de configuración necesarios.

Para crear un nuevo proyecto Next.js, abra su terminal y ejecute el siguiente comando:

```
npx crear-siguiente-aplicación@última
```

Después de ejecutar el comando, aparecerán las siguientes preguntas:

¿Cómo se llama tu proyecto? my-app

¿Te gustaría usar TypeScript? No / Sí

¿Le gustaría usar ESLint? No / Sí

¿Te gustaría usar Tailwind CSS? No / Sí

¿Desea usar el directorio `src`? No / Sí

¿Te gustaría utilizar App Router? (recomendado) No / Sí

¿Desea personalizar el alias de importación predeterminado? No / Sí

¿Qué alias de importación le gustaría configurar? @/*

Este comando le hará algunas preguntas para iniciar el proyecto, lo que creará un nuevo proyecto Next.js en un directorio llamado my-app. En este ejemplo, hemos decidido usar TypeScript en lugar de JavaScript.

Paso 3: Iniciar el servidor de desarrollo

Una vez creado un nuevo proyecto Next.js, puede iniciar el servidor de desarrollo con el comando `npm run dev`. Este comando inicia un servidor de desarrollo local que le permite previsualizar su aplicación Next.js en el navegador.

Para iniciar el servidor de desarrollo, navegue al directorio del proyecto y ejecute el siguiente comando en su terminal:

```
cd mi-aplicación
```

```
npm ejecutar dev
```

```
+ my-app git:(main) npm run dev
> my-app@0.1.0 dev
> next dev

ready - started server on 0.0.0.0:3000, url: http://localhost:3000
Attention: Next.js now collects completely anonymous telemetry regarding usage.
This information is used to shape Next.js' roadmap and prioritize features.
You can learn more, including how to opt-out if you'd not like to participate in this anonymous program, by visiting the following URL
https://nextjs.org/telemetry

event - compiled client and server successfully in 1242 ms (178 modules)
wait - compiling...
event - compiled successfully in 63 ms (137 modules)
```

Figura 1.5: Inicio exitoso del servidor local

Esto iniciará el servidor de desarrollo en `http://localhost`. Puede abrir esta URL en su navegador web para ver su aplicación Next.js.



Figura 1.6: Página de inicio del servidor local de Next.js

Paso 4: Instalar dependencias adicionales

Según los requisitos de su aplicación Next.js, podría necesitar instalar dependencias adicionales. Puede hacerlo con el comando `npm install`.

Por ejemplo, para instalar la biblioteca `axios` para realizar solicitudes HTTP, puede ejecutar el siguiente comando en su terminal:

```
npm instalar axios
```

Esto instalará la biblioteca `axios` y la agregará al archivo `package.json` de su proyecto.

Configurar un entorno de desarrollo para Next.js es un paso crucial para crear aplicaciones web con Next.js. Siguiendo los pasos descritos en esta sección, podrá crear un nuevo proyecto de Next.js, iniciar el servidor de desarrollo e instalar dependencias adicionales. Con un entorno de desarrollo completamente funcional, estará listo para empezar a crear sus propias aplicaciones Next.js.

Creando una aplicación Next.js sencilla

En esta sección, crearemos una aplicación Next.js sencilla desde cero. Comenzaremos configurando un nuevo proyecto Next.js con la herramienta CLI `create-next-app`.

Luego, crearemos una aplicación básica Hola Mundo para familiarizarnos con la estructura de una aplicación Next.js.

- Configuración de un nuevo proyecto Next.js

Consulta la sección anterior sobre cómo crear una aplicación Next.js desde cero.

Una vez completada la instalación, navegue hasta el directorio del proyecto:

```
cd mi-aplicación
```

Ahora que tenemos nuestro proyecto Next.js configurado, echemos un vistazo a la estructura del proyecto.

- Comprender la estructura de un proyecto Next.js

Un proyecto típico de Next.js tiene la siguiente estructura:

```
.próximo/  
módulos_de_nodo/  
fuente/  
páginas/  
estilos/  
público/  
paquete.json
```

El directorio `.next` es generado por Next.js y contiene los archivos compilados de la aplicación. El directorio `node_modules` contiene todas las dependencias instaladas por npm. El directorio `pages` contiene todas las páginas de la aplicación. El directorio `public` contiene todos los recursos estáticos, como imágenes, vídeos y fuentes. El directorio `designs` contiene todas las hojas de estilos CSS utilizadas en la aplicación. El archivo `package.json` contiene los metadatos del proyecto y las dependencias utilizadas.

Ahora que tenemos una comprensión básica de la estructura de un proyecto Next.js, creemos nuestra primera página.

- Creación de una aplicación sencilla de “Hola mundo”

En Next.js, una página es un componente React que se exporta como la exportación predeterminada de un archivo en el directorio de páginas. Para crear una aplicación Hello World simple , reemplazamos el código repetitivo en `index.tsx`:

```
// src/pages/index.tsx  
  
función Inicio() {  
    regresar <h1>¡Hola mundo!</h1>  
}  
  
exportar predeterminado Inicio;
```

En este código, hemos definido un nuevo componente llamado `Home` que devuelve un elemento `h1` simple con el texto "¡Hola mundo!". También hemos exportado este componente como la exportación predeterminada del archivo.

Ahora, si ejecuta el servidor de desarrollo con el siguiente comando:

```
npm ejecutar dev
```

Deberías poder ver la aplicación abriendo tu navegador y accediendo a `http://localhost`.

Deberías ver el texto "¡Hola mundo!".

se muestra en la página.

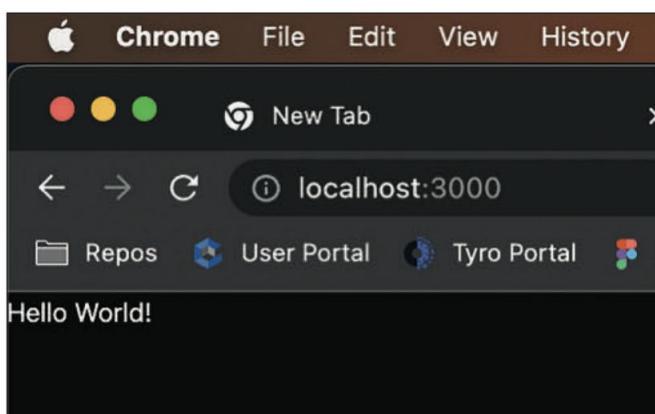


Figura 1.7: Aplicación Next.js ejecutándose en el navegador

¡Por fin has creado con éxito una aplicación Next.js sencilla!

Conclusión

En este capítulo, aprendimos sobre las aplicaciones web, comprendimos el funcionamiento básico de Next.js, sus ventajas y las funcionalidades básicas de HTML, CSS y JavaScript. También exploramos lo fácil que es ejecutar una aplicación web Next.js localmente en nuestro ordenador.

En el próximo capítulo, recordaremos los conceptos básicos de React, que es la biblioteca de JavaScript más popular que se utiliza para crear aplicaciones web interactivas y aplicaciones móviles multiplataforma.

Capítulo 2

Recordar Reaccionar

Introducción

React es una potente biblioteca de JavaScript que se utiliza para crear interfaces de usuario. En este capítulo, repasaremos los conceptos esenciales de React, cruciales para crear aplicaciones web robustas.

Comenzaremos presentando los componentes básicos de las aplicaciones React. Analizaremos cómo crear y usar componentes en React y cómo permiten a los desarrolladores crear elementos de interfaz de usuario reutilizables. También abordaremos JSX, una extensión de sintaxis de JavaScript que permite a los desarrolladores escribir código similar a HTML en sus archivos avaScript.

A continuación, profundizaremos en los conceptos importantes de props y estado en React. Explicaremos cómo las props permiten la transferencia de datos entre componentes, mientras que un estado se utiliza para gestionar los datos que cambian con el tiempo dentro de un componente. Abordaremos las mejores prácticas para trabajar con props y estado, y cómo pueden utilizarse para crear interfaces de usuario complejas e interactivas.

Finalmente, analizaremos los métodos de ciclo de vida en React, que permiten a los desarrolladores gestionar ciertos eventos en el ciclo de vida de un componente. Explicaremos cómo usar estos métodos para controlar el comportamiento de un componente y cómo optimizar el rendimiento y evitar errores en las aplicaciones de React.

Al final de este capítulo, tendrá una comprensión sólida de los conceptos esenciales de React y estará listo para comenzar a crear aplicaciones web sólidas y de alto rendimiento utilizando esta poderosa biblioteca.

Estructura

En este capítulo se tratarán los siguientes temas:

- Introducción al marco React, incluido el DOM virtual y la arquitectura basada en componentes

- Métodos del ciclo de vida de los componentes de React y su uso
- Sintaxis JSX y sus diferencias con el HTML tradicional
- Manejo de eventos en React y paso de datos entre componentes
- Trabajar con el estado y los accesorios de React, incluido el flujo de datos entre componentes
- Introducción a React Hooks y su uso
- Conceptos básicos de programación asíncrona en JavaScript y cómo utilizarlos ellos en React

Presentando React

¿Qué es React y por qué es importante? React es una popular biblioteca de JavaScript desarrollada y mantenida por Meta (anteriormente conocida como Facebook) para crear interfaces de usuario, en particular aplicaciones web. Se creó para abordar la necesidad de una forma más eficiente y escalable de gestionar interfaces de usuario complejas, a la vez que ofrece un rendimiento excelente. React ha ganado una inmensa popularidad a lo largo de los años porque permite a los desarrolladores crear componentes reutilizables y modulares que son fáciles de mantener y escalar.

Una de las principales razones por las que React es tan popular es su enfoque en la simplicidad y la facilidad de mantenimiento. Al dividir la interfaz de usuario en pequeños componentes autónomos, permite a los desarrolladores comprender y modificar el código fácilmente, incluso en aplicaciones grandes y complejas. Además, las optimizaciones de rendimiento de React, como el DOM virtual, garantizan que las aplicaciones creadas con React sean rápidas y responsivas.

el DOM y sus beneficios

El DOM virtual es una de las características clave de React y la principal razón de su excepcional rendimiento. El DOM virtual es una representación ligera en memoria del DOM (Modelo de Objetos del Documento) real que se utiliza para rastrear los cambios en la interfaz de usuario. Cada vez que se produce un cambio, React actualiza el DOM virtual en lugar del DOM real, lo que puede ser una operación lenta y costosa.

React utiliza un proceso llamado reconciliación para comparar el DOM virtual actual con el nuevo generado por un cambio en el estado o las propiedades de un componente.

A continuación, calcula la manera más eficiente de actualizar el DOM real para que coincida con el nuevo DOM virtual. Este proceso minimiza el número de actualizaciones del DOM real, lo que genera mejoras significativas en el rendimiento.

Consideremos el siguiente ejemplo:

```
importar React, { useState } de "react";
```

```
función Contador() {
```

```
    const [count, setCount] = useState(0);
```

```
    devolver (
```

```
        <div>
```

```
            <p>Hiciste clic {count} veces</p>
```

```
            <button onClick={() => setCount(count + 1)}>Haz clic en mí</button>
```

```
        </div>
```

```
    );
```

```
}
```

```
exportar contador predeterminado;
```

En este sencillo componente de contador, el estado se actualiza cada vez que se hace clic en el botón. React utiliza el DOM virtual para determinar las actualizaciones mínimas necesarias para el DOM real, garantizando así un rendimiento óptimo.

Arquitectura basada en componentes en React

La arquitectura basada en componentes de React es un factor clave en su popularidad y facilidad de mantenimiento. Los componentes son fragmentos de código autónomos y reutilizables que representan una parte de la interfaz de usuario. Pueden combinarse y anidarse para crear estructuras de interfaz de usuario complejas.

Hay dos principios principales detrás de la arquitectura basada en componentes de React:

- Principio de Responsabilidad Única: Cada componente debe tener una única responsabilidad o propósito. Esto facilita la comprensión, las pruebas y el mantenimiento del código.

- Composición: Los componentes se pueden combinar para crear estructuras de interfaz de usuario más complejas. Esto permite a los desarrolladores reutilizar componentes y crear código más fácil de mantener.

A continuación se muestra un ejemplo sencillo de una interfaz de usuario basada en componentes:

```
importar React desde “react”;
importar encabezado desde “./Header”;
importar contenido desde “./Content”;
importar pie de página desde “./Footer”;

función App() {
    devolver (
        <div>
            <Encabezado />
            <Contenido />
            <Pie de página />
        </div>
    );
}

exportar aplicación predeterminada;
```

En este ejemplo, el componente App se compone de tres componentes: Encabezado, Contenido y Pie de página. Cada uno de estos componentes tiene su propia responsabilidad, lo que facilita el mantenimiento y la comprensión de la aplicación en general.

Comprensión de los componentes funcionales y de clase

Los componentes de React se pueden escribir como componentes funcionales o componentes de clase. Antes de la introducción de Hooks en React 16.8, los componentes funcionales no tenían estado y solo se usaban para presentar datos, mientras que los componentes de clase sí lo tenían y se usaban para una lógica más compleja. Sin embargo, con la introducción de

Los ganchos y componentes funcionales ahora pueden tener estado y realizar efectos secundarios, lo que los hace más potentes y flexibles.

Los componentes funcionales son más simples y concisos que los componentes de clase, lo que los convierte en la opción preferida de muchos desarrolladores. Son simplemente funciones de JavaScript que toman props como entrada y devuelven JSX para renderizar la interfaz de usuario.

A continuación se muestra un ejemplo de un componente funcional:

```
importar React desde "react";
```

```
función Bienvenido(props) {  
    devolver <h1>¡Hola, {props.name}!</h1>;  
}  
}
```

```
exportar predeterminado Bienvenido;
```

Por el contrario, los componentes de clase son clases de JavaScript que extienden React. Clase de componente . Incluye un método render() que devuelve JSX y puede tener métodos locales de estado y ciclo de vida.

He aquí un ejemplo de un componente de clase:

```
importar React, { Component } de "react";
```

```
clase Bienvenido extiende Componente {  
    prestar() {  
        devolver <h1>¡Hola, {this.props.name}!</h1>;  
    }  
}  
}
```

```
exportar predeterminado Bienvenido;
```

Si bien React aún admite componentes de clase, la introducción de Hooks ha hecho que los componentes funcionales sean más potentes y versátiles. Los Hooks permiten

utilizar el estado y otras características de React sin escribir una clase, lo que genera un código más limpio y legible.

Para ilustrar el uso de Hooks en componentes funcionales, convirtamos el ejemplo de componente de clase anterior en un componente funcional con el estado:

```
importar React, { useState } de “react”;

función Bienvenido() {
    const [nombre, setName] = useState(“Juan”);

    const handleChange = (evento) => {
        setName(evento.objetivo.valor);
    };

    devolver (
        <div>
            ¡Hola, {nombre}!
            <tipo de entrada=“texto” valor={nombre} onChange={handleChange} />
        </div>
    );
}

exportar predeterminado Bienvenido;
```

En este ejemplo, utilizamos el gancho useState para administrar el estado del nombre en el componente funcional. También definimos una función handleChange para manejar el evento de cambio de entrada y actualizar el estado.

En conclusión, la arquitectura basada en componentes de React, el DOM virtual y el énfasis en la reutilización lo convierten en una opción potente y popular para crear interfaces de usuario. Con la introducción de Hooks, los componentes funcionales se han vuelto aún más potentes y versátiles, consolidando aún más la posición de React como una biblioteca líder de avaScript. A medida que continúe explorando React y sus características, descubrirá...

que proporciona una forma eficiente y mantenible de crear aplicaciones web complejas y escalables.

Métodos del ciclo de vida de los componentes de React y su uso

Expliquemos los métodos del ciclo de vida del componente React y sus beneficios.

- Descripción general de los métodos del ciclo de vida

Los métodos de ciclo de vida son métodos especiales en los componentes de clase que permiten ejecutar código en puntos específicos durante el ciclo de vida del componente. Son esenciales para gestionar efectos secundarios, como la obtención de datos, la actualización del DOM y la gestión de eventos. Los métodos de ciclo de vida se pueden agrupar en tres fases principales: montaje, actualización y desmontaje.

Nota: Con la introducción de los Hooks, los componentes funcionales ahora pueden realizar tareas similares mediante useEffect. Esta sección se centra en los métodos de ciclo de vida de los componentes de clase, pero es importante tener en cuenta este enfoque alternativo en los componentes funcionales.

- Métodos de fase de montaje

La fase de montaje ocurre cuando se crea e inserta un componente en el DOM. Hay dos métodos de ciclo de vida asociados a esta fase:

constructor: El método constructor se utiliza para inicializar el estado del componente y vincular los controladores de eventos. Se llama antes de montar el componente.

He aquí un ejemplo:

```
clase MyComponent extiende React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { mensaje: "¡Hola, mundo!" };  
    este.handleClick = este.handleClick.bind(este);  
  }  
}
```

componentDidMount: Este método se llama inmediatamente después de insertar el componente en el DOM. Es el lugar ideal para obtener datos, configurar suscripciones o realizar otros efectos secundarios.

He aquí un ejemplo:

```
clase MyComponent extiende React.Component {
  componenteDidMount() {
    console.log("El componente ha sido montado.");
  }
}
```

- Actualización de los métodos de fase

La fase de actualización ocurre cuando el estado o las propiedades de un componente cambian, lo que provoca una nueva renderización. Hay dos métodos de ciclo de vida asociados con esta fase:

`shouldComponentUpdate`: Este método se llama antes de volver a renderizar, lo que permite determinar si el componente debe actualizarse según los cambios de estado o propiedades. Devuelve "true" por defecto. Si devuelve "false", el componente no se actualizará y no se llamarán los demás métodos del ciclo de vida.

He aquí un ejemplo:

```
clase MyComponent extiende React.Component {
  shouldComponentUpdate(nextProps, nextState) {
    devuelve esto.props.someValue !== nextProps.someValue;
  }
}
```

`componentDidUpdate`: Este método se llama inmediatamente después de actualizar un componente. Es útil para realizar efectos secundarios, como la manipulación del DOM o la obtención de datos, en respuesta a cambios de propiedad o estado.

He aquí un ejemplo:

```
clase MyComponent extiende React.Component {
  componenteDidUpdate(prevProps, prevState) {
    si (este.props.algunValor !== prevProps.algunValor) {
```

```

        console.log("algúnValor ha cambiado.");
    }
}
}

```

- Métodos de fase de desmontaje

La fase de desmontaje ocurre cuando se elimina un componente del DOM.

Hay un método de ciclo de vida asociado con esta fase:

`componentWillUnmount`: Este método se llama inmediatamente antes de desmontar y destruir el componente. Es ideal para realizar tareas de limpieza, como cancelar solicitudes de red, eliminar detectores de eventos o borrar temporizadores.

He aquí un ejemplo:

```

clase MyComponent extiende React.Component {
    componenteSeDesmontará() {
        console.log("El componente se desmontará.");
    }
}

```

- Manejo de errores en los métodos del ciclo de vida

React proporciona métodos de ciclo de vida para el manejo de errores y la visualización de interfaces de usuario de respaldo cuando ocurre un error:

`getDerivedStateFromError` estático: Este método se llama cuando se produce un error en un componente secundario. Permite actualizar el estado para mostrar una interfaz de usuario alternativa.

He aquí un ejemplo:

```

clase MyComponent extiende React.Component {
    estática getDerivedStateFromError(error) {
        retorna { hasError: verdadero };
    }
}

```

componentDidCatch: Este método se llama cuando se detecta un error en un componente secundario. Puede usarse para registrar el error o realizar otras acciones secundarias.

He aquí un ejemplo:

```
clase MyComponent extiende React.Component {  
    componentDidCatch(error, información) {  
        console.log("Error detectado:", error, información);  
    }  
}
```

A continuación se muestra un ejemplo del uso de ambos métodos de manejo de errores en un componente:

```
clase ErrorBoundary extiende React.Component {  
    constructor(props) {  
        super(props);  
        este.estado = { hasError: false };  
    }  
  
    estática getDerivedStateFromError(error) {  
        retorna { hasError: verdadero };  
    }  
  
    componentDidCatch(error, información) {  
        console.log("Error detectado:", error, información);  
    }  
  
    prestar() {  
        si (este.estado.tieneError) {  
            devolver <h1>jAlgo salió mal!</h1>;  
        }  
    }  
}
```

```
        }  
        devuelve esto.props.children;  
    }  
}  
  
// Uso:  
<Límite de error>  
  <MiComponente />  
</ErrorBoundary>
```

En este ejemplo, ErrorBoundary es un componente de orden superior que envuelve a MyComponent. Si se produce un error en MyComponent, el límite de error lo detectará y mostrará una interfaz de usuario alternativa.

Los siguientes puntos explican cuándo utilizar métodos de ciclo de vida específicos

- Utilice el constructor para inicializar el estado y vincular los controladores de eventos. Evite realizando efectos secundarios en el constructor.
- Utilice componentDidMount para obtener datos, configurar suscripciones o realizar otros efectos secundarios que deberían ocurrir cuando el componente se monta por primera vez.
- Utilice shouldComponentUpdate para optimizar el rendimiento al evitar re-renderizaciones innecesarias basadas en cambios específicos de estado o propiedad.
- Utilice componentDidUpdate para los efectos secundarios que deberían ocurrir en respuesta a cambios de propiedad o estado, como actualizar el DOM o obtener nuevos datos.
- Utilice componentWillUnmount para tareas de limpieza, como cancelar la conexión de red, solicitudes o eliminar escuchas de eventos.
- Utilice getDerivedStateFromError y componentDidCatch para manejar errores en componentes secundarios y mostrar interfaces de usuario de respaldo.

Al comprender los diferentes métodos del ciclo de vida y sus casos de uso adecuados, podrá gestionar eficazmente los efectos secundarios, optimizar el rendimiento y gestionar errores en sus componentes de React. A medida que adquiera experiencia con React, se familiarizará con los matices de cada método del ciclo de vida y cómo aprovecharlos al máximo en sus aplicaciones.

Sintaxis JSX y sus diferencias con el HTML tradicional

JSX (JavaScript XML) es una extensión de sintaxis para JavaScript que permite escribir código similar a HTML dentro de código JavaScript. No es obligatorio para usar React, pero es el enfoque recomendado porque hace que el código sea más legible y fácil de entender. JSX es similar a HTML, pero presenta algunas diferencias clave y características adicionales que lo hacen más potente y flexible.

Reglas de sintaxis JSX y mejores prácticas

Aquí hay algunas reglas básicas y mejores prácticas para escribir JSX:

- JSX debe estar encerrado en un solo elemento raíz. Si tiene varios elementos, envuélvalos en un elemento padre como un `<div>` o un fragmento `<></>`:

```
// Correcto
función MiComponente() {
    devolver (
        <>
        Título
        <p>Contenido</p>
    </>
);
}
```

```
// Incorrect
función MiComponente() {
    devolver (
        Título
        <p>Contenido</p>
);
}
```

Las etiquetas JSX pueden cerrarse automáticamente si no tienen etiquetas secundarias.

Asegúrese de incluir una barra de cierre antes del corchete angular, como `` o `<input />`:

```
función MiComponente() {  
    devolver ;  
}
```

Los nombres de atributos en JSX usan camelCase en lugar de kebabCase. Por ejemplo, use `onClick` en lugar de `on-click` y `className` en lugar de `class`:

```
función MiComponente() {  
    devolver (  
        <button onClick={() => console.log("Hizo clic")}>  
            Haz clic en mí  
        </botón>  
    );  
}
```

Incrustar expresiones de JavaScript en JSX

JSX permite incrustar expresiones JavaScript mediante llaves `{}`. Puedes usar esta función para mostrar variables, realizar cálculos e incluso renderizar otros componentes en tu JSX.

He aquí un ejemplo:

```
función Saludo(props) {  
    const nombre = props.nombre;  
    devolver <h1>¡Hola, {nombre}!</h1>;  
}
```

Manejo de renderizado condicional y bucles en JSX

La representación condicional y los bucles son tareas comunes en los componentes de React. Puedes usar expresiones JavaScript en tu JSX para gestionar estas tareas, como se indica a continuación:

- Para la representación condicional, utilice operadores ternarios o AND lógico (`&&`) expresiones:

```
función Saludo(props) {  
    devolver (  
        <div>  
            {props.isLoggedIn ? (  
                ¡Bienvenido de nuevo, {props.name}!  
            ) : (  
                <h1>Por favor, inicie sesión.</h1>  
            )}  
        </div>  
    );  
}
```

- Para realizar un bucle, utilice el método de mapa para iterar sobre matrices y renderizar elementos:

```
función ListItems(props) {  
    const items = props.items;  
  
    devolver (  
        <ul>  
            {items.map((item) => (  
                <li key={item.id}>{item.name}</li>  
            ))}  
        </ul>  
    );  
}
```

```
);  
}
```

Diferencias entre JSX y HTML

Si bien JSX parece similar a HTML, existen algunas diferencias clave entre ambos. dos:

- Nombres de atributos: Como se mencionó anteriormente, JSX usa nombres de atributos con formato camelCase en lugar de kebabCase. Por ejemplo, use className en lugar de class y htmlFor en lugar de for.
- Atributo de estilo: En JSX, el atributo de estilo espera un objeto JavaScript en lugar de una cadena CSS. Cada propiedad CSS debe escribirse en mayúsculas y minúsculas.

```
función MiComponente() {  
    constante divStyle = {  
        color: "azul",  
        color de fondo: "blanco",  
    };  
  
    devolver <div style={divStyle}>¡Hola, mundo!</div>;  
}
```

Etiquetas de cierre: En JSX, todas las etiquetas deben cerrarse, incluso si no tienen etiquetas hijas. Esto difiere de HTML, donde algunas etiquetas, como `` y `<input>`, no requieren etiquetas de cierre. En JSX, estos elementos deben cerrarse automáticamente, con una barra diagonal antes del corchete angular de cierre (por ejemplo, `` y `<input />`).

```
// Correcto  
función MiComponente() {  
    devolver (  
        <div>  
              
            <input type="text" name="nombre de usuario" />  
    )  
}
```

```
</div>
);
}

// Incorrect
función MiComponente() {
    devolver (
        <div>
            
            <input type="text" name="nombre de usuario">
        </div>
    );
}
```

Expresiones JavaScript: JSX permite incrustar expresiones JavaScript mediante llaves ({}). Esta función no está disponible en HTML, donde se deben usar literales de plantilla u otros mecanismos para lograr una funcionalidad similar.

- Componentes personalizados: JSX permite crear componentes personalizados que se pueden usar de la misma manera que los elementos HTML. En HTML, se requieren componentes web u otros métodos para lograrlo.

En resumen, S es una extensión de sintaxis potente y flexible para JavaScript que permite escribir código similar a HTML dentro de código JavaScript. Al comprender sus reglas de sintaxis, las mejores prácticas y las diferencias con el HTML tradicional, se puede usar JSX eficazmente para crear componentes React limpios y fáciles de mantener. Con la capacidad de incrustar expresiones JavaScript, renderizar contenido condicionalmente y recorrer arrays, S simplifica la creación de interfaces de usuario complejas y dinámicas.

Manejo de eventos en React y paso de datos entre componentes

Esta sección explicará cómo administrar eventos en React y pasar datos a los componentes.

Conceptos básicos del manejo de eventos de React

El manejo de eventos en React es similar al manejo de eventos en JavaScript simple.

Sin embargo, existen algunas diferencias y prácticas recomendadas que se deben tener en cuenta al trabajar con eventos en React, entre ellas:

- Nombres de eventos: en React, los nombres de eventos se escriben en camelCase, como onClick, onSubmit y onMouseMove.
- Controlador de eventos: Debe pasar una función como controlador de eventos, no una cadena. Esta función se llamará cuando se active el evento.

He aquí un ejemplo:

```
función handleClick() {  
    console.log("¡Botón hecho clic!");  
}  
  
función MiComponente() {  
    regresar <button onClick={handleClick}>Haz clic en mí</button>;  
}
```

Eventos sintéticos y agrupación de eventos

React utiliza un concepto llamado Eventos Sintéticos, que encapsulan los eventos nativos del navegador. Esto proporciona una interfaz consistente para gestionar eventos en diferentes navegadores y mejora el rendimiento mediante la agrupación de eventos. Puedes acceder al evento nativo mediante la propiedad nativeEvent del objeto Evento Sintético.

He aquí un ejemplo:

```
función handleClick(evento) {  
    console.log("Botón hecho clic:", evento.nativeEvent);  
}  
  
función MiComponente() {
```

```
regresar <button onClick={handleClick}>Haz clic en mí</button>;
}
```

Vinculación de controladores de eventos a componentes

Al usar componentes de clase, es fundamental vincular los controladores de eventos a la instancia del componente para acceder a ella correctamente. Existen varias maneras de vincular controladores de eventos, como las siguientes:

- Enlace del constructor

```
clase MyComponent extiende React.Component {
  constructor(props) {
    super(props);
    este.handleClick = este.handleClick.bind(este);
  }

  manejarClick() {
    console.log("Botón hecho clic:", this);
  }

  prestar() {
    devuelve <button onClick={this.handleClick}>Haz clic en mí</button>;
  }
}
```

- Funciones de flecha

```
clase MyComponent extiende React.Component {
  manejarClick = () => {
    console.log("Botón hecho clic:", this);
  };
}
```

```
prestar() {  
    devuelve <button onClick={this.handleClick}>Haz clic en mí</button>;  
}  
}
```

- Funciones de flecha en línea (úselas con precaución, ya que pueden tener problemas de rendimiento) trascendencia)

```
clase MyComponent extiende React.Component {  
    manejarClick() {  
        console.log("Botón hecho clic:", this);  
    }  
  
    prestar() {  
        devuelve <button onClick={() => this.handleClick()}>Haz clic en mí</button>;  
    }  
}
```

Pasando datos a través de accesorios

Los accesorios son el mecanismo principal para pasar datos entre componentes en React. Le permiten pasar valores de un componente principal a un componente secundario y representar el componente secundario con los datos proporcionados.

He aquí un ejemplo:

```
función Saludo(props) {  
    devolver <h1>¡Hola, {props.name}!</h1>;  
}
```

```
función App() {
    devolver <Saludo nombre="Juan" />;
}
```

Uso de funciones de devolución de llamada para la comunicación padre-hijo

Para comunicarse desde un componente secundario a un componente principal, se puede pasar una función de devolución de llamada como propiedad. El componente secundario puede entonces llamar a esta función cuando se produce un evento, como al hacer clic en un botón.

He aquí un ejemplo:

```
clase ParentComponent extiende React.Component {
    handleChildClick(nombre) {
        console.log("Botón hecho clic en el componente secundario:", nombre);
    }

    prestar() {
        devolver <ChildComponent onClick={this.handleChildClick} />;
    }
}

función ChildComponent(props) {
    devuelve <button onClick={() => props.onClick("Hijo")}>Haz clic en mí</but-ton>;
}
```

Elevación del estado y gestión del estado compartido

Cuando varios componentes necesitan compartir y modificar los mismos datos, es recomendable transferir el estado al ancestro común más cercano. Esto implica trasladar la gestión del estado a un componente de nivel superior, que a su vez puede transferir los datos a los componentes secundarios mediante propiedades.

He aquí un ejemplo:

```
clase App extiende React.Component {  
  constructor(props) {  
    super(props);  
    este.estado = { contador: 0 };  
    este.incrementCounter = este.incrementCounter.bind(este);  
  }  
  
  incrementCounter() {  
    este.setState((prevState) => ({ contador: prevState.counter + 1 }));  
  }  
  
  prestar() {  
    devolver (  
      <>  
      <CounterDisplay contador={este.estado.contador} />  
      <CounterButton onClick={este.incrementCounter} />  
      </>  
    );  
  }  
}  
  
función CounterDisplay(props) {  
  devuelve <h1>Contador: {props.counter}</h1>;  
}  
  
función CounterButton(props) {  
  devuelve <button onClick={props.onClick}>Contador de incremento</button>;  
}
```

En este ejemplo, el componente App administra el estado compartido (el contador)

valor) y lo pasa a los componentes CounterDisplay y CounterButton mediante propiedades. La función incrementCounter también se pasa al componente CounterButton , lo que le permite actualizar el estado compartido.

En resumen, la gestión de eventos y la transferencia de datos entre componentes en React es fundamental para crear interfaces de usuario dinámicas. Al comprender los fundamentos de la gestión de eventos de React, los eventos sintéticos y la vinculación de eventos, podrá gestionar eficazmente las interacciones del usuario en sus componentes. Además, aprender a transferir datos mediante props y usar funciones de devolución de llamada para la comunicación entre componentes padre-hijo le permite crear aplicaciones complejas y fáciles de mantener. Por último, la gestión del estado compartido y la elevación del estado permiten gestionar eficientemente los datos a los que varios componentes deben acceder y modificar.

Estado y propiedades de React

Comencemos a trabajar con el estado y las propiedades de React, incluido el flujo de datos entre componentes:

- Comprender el estado en los componentes de React

El estado en los componentes de React es una forma de almacenar y gestionar datos que cambian con el tiempo. A diferencia de las propiedades, que son de solo lectura y se transmiten desde los componentes principales, el estado es local y mutable dentro del componente. Los componentes funcionales con estado se pueden crear mediante el gancho useState .

- Inicialización y actualización de estado

Para inicializar un estado en un componente funcional, puede utilizar el gancho useState .

Toma un valor inicial como argumento y devuelve una matriz con dos elementos: el valor del estado actual y una función para actualizarlo. Puedes usar la desestructuración de matrices para asignar los elementos devueltos a variables:

```
importar React, { useState } de 'react';
```

```
función Contador() {
  const [count, setCount] = useState(0);

  función increment() {
    setCount(count + 1);
  }
}
```

```
devolver (
  <div>
    <p>Conde: {count}</p>
    <button onClick={increment}>Incrementar</button>
  </div>
);
}
```

- Manejo de la inmutabilidad del estado y mejores prácticas

En React, el estado debe considerarse inmutable. Esto significa que, en lugar de modificar directamente el estado actual, se debe crear un nuevo objeto de estado con los datos actualizados. Esta práctica ayuda a prevenir errores y optimizar el rendimiento, ya que React puede comparar eficientemente los objetos de estado antiguos y nuevos para determinar si es necesario volver a renderizarlos.

Al actualizar el estado en función del estado actual, utilice una actualización funcional pasando una función al actualizador de estado:

```
función increment() {
  setCount((prevCount) => prevCount + 1);
}
```

- Los accesorios y su papel en la comunicación de los componentes

Como se mencionó anteriormente, las propiedades son la principal forma de pasar datos entre componentes en React. Son de solo lectura y permiten que los componentes principales pasen valores a sus componentes secundarios. Los componentes secundarios pueden acceder a los datos pasados a través de su objeto de propiedades:

```
función Saludo({ nombre }) {
  devolver <h1>¡Hola, {nombre}!</h1>;
```

```
}
```

```
función App() {  
    devolver <Saludo nombre="Juan" />;  
}
```

- Validación de propiedades con PropTypes

PropTypes es una biblioteca que permite especificar los tipos de propiedades esperados para un componente. Es una herramienta útil para detectar errores y documentar los tipos de propiedades esperados de los componentes. Para usar PropTypes, es necesario importar la biblioteca y definir los tipos de propiedades esperados en el componente.

```
importar React desde 'react';  
importar PropTypes desde 'prop-types';  
  
función Saludo({ nombre }) {  
    devolver <h1>¡Hola, {nombre}!</h1>;  
}  
  
Saludo.propTypes = {  
    nombre: PropTypes.string.isRequired,  
};  
  
exportar saludo predeterminado;
```

- Estado de gestión y flujo de datos con contexto

En algunos casos, puede que necesites compartir el estado entre componentes que no están conectados directamente mediante una relación padre-hijo. La API de contexto es una función integrada en React que permite compartir el estado en todo el árbol de componentes sin tener que pasar manualmente las propiedades a través de varios niveles.

Para utilizar la API de contexto, debe crear un contexto usando `React.createContext()` Y envuelva su árbol de componentes con un proveedor de contexto. El proveedor de contexto toma una propiedad de valor , que puede ser cualquier dato que desee compartir entre los componentes. árbol:

```
importar React, { useState, useContext } de 'react';
```

```
constante ThemeContext = React.createContext();
```

```
función App() {
    const [tema, setTheme] = useState('luz');

    devolver (
        <ThemeContext.Provider valor={{ tema, establecerTema }}>
            <Barra de herramientas />
        </ThemeContext.Provider>
    );
}
```

```
función Barra de herramientas() {
    devolver <Cambio de tema />;
}
```

```
función ThemeSwitch() {
    const { tema, setTheme } = useContext(ThemeContext);

    función toggleTheme() {
        setTheme((prevTheme) => (prevTheme === 'claro' ? 'oscuro' : 'claro'));
    }
}
```

```
regresar ( <botón onClick={alternarTema}>
```

```
Cambiar al tema {tema === 'claro' ? 'oscuro' : 'claro'}  
</botón> );  
}
```

En este ejemplo, el componente App inicializa el estado del tema y lo proporciona a ThemeContext. El componente ThemeSwitch , anidado en la barra de herramientas, componente, luego puede acceder y actualizar el estado del tema usando useContext Gancho, sin necesidad de pasar el tema y la función de actualización a través de props.

En resumen, comprender el estado y las propiedades de los componentes funcionales de React es crucial para crear aplicaciones dinámicas e interactivas. Al usar useState Gancho para inicializar y actualizar el estado, puede administrar los datos del componente de manera efectiva. Seguir las mejores prácticas, como tratar el estado como inmutable y usar actualizaciones funcionales, ayuda a prevenir errores y optimizar el rendimiento. Las propiedades desempeñan un papel fundamental en la comunicación entre componentes, permitiendo pasar datos entre ellos y validar sus tipos con PropTypes. Finalmente, la API de contexto simplifica la gestión del estado y el flujo de datos en aplicaciones más complejas al compartir el estado en todo el árbol de componentes sin necesidad de explorar manualmente las propiedades.

Presentación de React Hooks y su uso

¿Qué son los Hooks de React y por qué usarlos? Los Hooks de React son un conjunto de funciones introducidas en React 16.8 que permiten usar el estado y otras características de React en componentes funcionales. Los Hooks proporcionan una forma más sencilla de gestionar el estado y los efectos secundarios en componentes funcionales sin la complejidad de los componentes de clase. Algunas de las ventajas de usar Hooks incluyen una mejor reutilización del código, un código más simple y pruebas más sencillas.

- Uso del gancho useState

El gancho useState permite añadir estado a componentes funcionales. Toma un estado inicial como argumento y devuelve un array con dos elementos: el estado actual y una función para actualizarlo. Se puede usar la desestructuración de arrays para asignar los elementos devueltos a variables:

```
importar React, { useState } de 'react';
```

```
función Contador() {  
    const [count, setCount] = useState(0);  
  
    función increment() {  
        setCount(count + 1);  
    }  
  
    devolver (  
        <div>  
            <p>Conde: {count}</p>  
            <button onClick={increment}>Incrementar</button>  
        </div>  
    );  
}
```

- Trabajar con useEffect y sus dependencias

El gancho useEffect permite ejecutar efectos secundarios, como obtener datos o actualizar el DOM, en componentes funcionales. Acepta dos argumentos: una función que contiene el efecto secundario y una matriz de dependencias opcional. La función del efecto secundario se ejecuta después de cada renderizado por defecto, pero se puede controlar su ejecución proporcionando una matriz de dependencias. El efecto solo se ejecutará cuando cambie una de las dependencias:

```
importar eact, { useState, useEffect } de react;  
  
función serrofile({ userd }) {  
    const [usuario, setUser] = useState(null);  
  
    usarEfecto(() => {  
        función asíncrona fetchUser() {  
            respuesta constante = await fetch('https://api.example.com/users/$  
            {userId}');  
            const userData = await respuesta.json();  
        }  
    });  
}
```

```

    setUser(datosUsuario);

}

buscarUsuario();
}, [ID de usuario]);

devolver (
<div>
{usuario ? (
<div>
<h1>{nombre_usuario}</h1>
<p>{usuario.correo electrónico}</p>
</div>
): (
<p>Cargando...</p>
)}
</div>
);
}

```

- Implementación de ganchos personalizados para lógica reutilizable

Los Hooks personalizados son una potente función que permite crear lógica reutilizable en todos los componentes. Un Hook personalizado es simplemente una función que empieza con la palabra "use" y contiene otros Hooks:

```

importar { useState, useEffect } de react;

función useFetch(url) {
  const [datos, setData] = useState(null); const
  [cargando, setLoading] = useState(true);

```

```
usarEfecto(() => {
    función asíncrona fetchData() {
        const respuesta = await fetch(url);
        const jsonData = await respuesta.json();
        establecerDatos(jsonData);
        setLoading(falso);
    }

    obtener datos();
}, [url]);

retorna { datos, cargando };
}

función App() {
    const { datos, cargando } = useFetch('https://api.example.com/data');

    devolver (
        <div>
            {cargando ? <p>Cargando...</p> : <p>Datos: {JSON.stringify(data)}</p>}
        </div>
    );
}
```

- useContext y useReducer para la gestión avanzada del estado

Los ganchos useContext y useReducer se pueden usar juntos para crear patrones avanzados de gestión de estados similares a Redux. El gancho useContext permite acceder al valor de contexto más cercano en el árbol de componentes, mientras que el gancho useReducer gestiona transiciones de estado complejas mediante una función reductora:

```
importar React, { createContext, useReducer, useContext } de 'react';
```

```
const initialState = { count: 0 };
```

```
función reductor(estado, acción) {
```

```
interruptor (acción.tipo) {
```

```
caso 'incremento':
```

```
retorna { count: estado.count + 1 };
```

```
caso 'decremento':
```

```
retorna { count: estado.count - 1 };
```

```
por defecto:
```

```
lanzar nuevo Error(Tipo de acción desconocido: ${action.type});
```

```
}
```

```
}
```

```
const CounterContext = crearContexto();
```

```
función CounterProvider({ children }) { const [estado,
```

```
despacho] = useReducer(reducer, initialState);
```

```
devolver (
```

```
<CounterContext.Provider valor={{ estado, despacho }}>
```

```
{niños}
```

```
</CounterContext.Provider>
```

```
);
```

```
}
```

```
función Contador() {
```

```
const { estado, despacho } = useContext(CounterContext);
```

```
devolver (
<div>
<p>Recuento: {estado.conteo}</p>
<button onClick={() => dispatch({ tipo: 'incremento' })}>
Incremento
</botón>
<button onClick={() => dispatch({ tipo: 'decremento' })}>
Decremento
</botón>
</div>
);
}

función App() {
devolver (
<ContadorProveedor>
<Contador />
</CounterProvider>
);
}
```

Otros ganchos integrados y sus casos de uso

React ofrece varios otros libros integrados, cada uno con casos de uso específicos.

- **useMemo:** Optimiza el rendimiento memorizando un valor, que se recalcula solo cuando cambian sus dependencias. Este Hook es útil al trabajar con funciones u objetos de alto consumo computacional.
- **useCallback:** Similar a useMemo, pero para memorizar funciones. Devuelve una versión memorizada de la función de devolución de llamada que solo cambia si una de las dependencias ha cambiado.
- **useRef:** Crea un objeto de referencia mutable que permite almacenar valores entre renderizaciones sin activar una nueva renderización. Algunos casos de uso comunes incluyen referenciar elementos del DOM y almacenar el estado anterior.

- `useLayoutEffect`: Similar a `useEffect`, pero se ejecuta sincrónicamente después de todas las mutaciones del DOM, lo que garantiza que el efecto y su limpieza se ejecuten en el mismo marco. Use este Hook cuando necesite leer el diseño o medir el DOM antes de que el navegador pinte.

En resumen, React Hooks proporciona una forma potente y sencilla de administrar el estado, los efectos secundarios y otras características de React en componentes funcionales.

Los Hooks integrados, como `useState`, `useEffect`, `useContext` y `useReducer`, abarcan una amplia gama de casos de uso, mientras que los Hooks personalizados permiten la creación de lógica reutilizable entre componentes. Se pueden implementar patrones avanzados de gestión de estados mediante `useContext` y `useReducer`, y otros Hooks integrados, como `useMemo`, `useCallback`, `useRef` y `useLayoutEffect`, ofrecen funcionalidad adicional para diversos escenarios.

Programación asincrónica en JavaScript y su aplicación en React

La programación asíncrona es un paradigma de programación que permite la ejecución simultánea de múltiples tareas sin bloquear el hilo de ejecución principal. JavaScript es uniproceso, lo que significa que solo puede ejecutar una operación a la vez.

La programación asíncrona permite que JavaScript realice tareas que consumen mucho tiempo, como obtener datos de IP o leer archivos, sin congelar la interfaz de usuario.

- Funciones de devolución de llamada y sus limitaciones

Las devoluciones de llamada son funciones que se pasan como argumentos a otras funciones, las cuales se ejecutan posteriormente. Las devoluciones de llamada son la forma más básica de gestionar código asíncrono en JavaScript:

```
función getData(devolución de llamada) {
    establecerTiempo de espera(() => {
        const data = 'Datos de muestra';
        devolución de llamada(datos);
    }, 1000);
}
```

```
obtenerDatos((datos) => {
    console.log(data); // 'Datos de muestra'
});
```

Sin embargo, las devoluciones de llamadas pueden conducir al infame infierno de las devoluciones de llamadas cuando se trata con múltiples operaciones asincrónicas anidadas, lo que da como resultado un código ilegible y difícil de mantener.

- Promesas y encadenamiento

Las promesas son una forma más avanzada de gestionar código asíncrono. Una promesa representa un valor que podría estar disponible en el futuro. Las promesas tienen tres estados posibles: pendiente, cumplida o rechazada. Puedes usar el método "then" para adjuntar devoluciones de llamada que se llamarán cuando se cumpla la promesa o el método "catch".

Método para manejar errores:

```
función obtenerDatos() {
    devolver nueva Promesa((resolver, rechazar) => {
        establecerTiempo de espera(() => {
            const data = 'Datos de muestra';
            resolver(datos);
        }, 1000);
    });
}

obtenerDatos().luego((datos) => {
    console.log(data); // 'Datos de muestra'
});
```

Las promesas se pueden encadenar, lo que permite un código más legible y fácil de mantener cuando se trabaja con múltiples operaciones asincrónicas.

- Sintaxis `async/await` para un código asíncrono más limpio

`Async/await` es una sintaxis moderna que simplifica aún más el trabajo con código asíncrono. Una función asíncrona devuelve una promesa, y la palabra clave `await` se usa para pausar la ejecución de la función hasta que se resuelva la promesa:

```
función asíncrona fetchData() {  
    constante respuesta = await fetch('https://api.example.com/data');  
    const data = await respuesta.json();  
    console.log(datos);  
}  
  
obtener datos();
```

- Integración de código asíncrono con componentes React

Para integrar código asíncrono en componentes funcionales de React, puedes usar el Hook `useEffect` para obtener datos o realizar otros efectos secundarios:

```
importar eact, { useState, useEffect } de react;  
  
función DataDisplay() {  
    const [datos, setData] = useState(nulo);  
  
    usarEfecto(() => {  
        función asíncrona fetchData() {  
            const respuesta = await fetch('https://api.example.com/data'); const jsonData =  
            await respuesta.json(); setData(jsonData);  
        }  
    })  
}
```

```
        obtener datos();  
    }, []);  
  
    devolver <div>{datos ? <p>{JSON.stringify(datos)}</p> : <p>Cargando...</p>}</div>;  
}
```

- Manejo de errores y mejores prácticas en código React asincrónico

Al trabajar con código asincrónico en React, es fundamental gestionar los errores correctamente. Puedes usar bloques try/catch dentro de funciones asincrónicas para detectar errores y mostrar los mensajes de error correspondientes al usuario:

```
importar eact, { useState, useEffect } de react;
```

```
función DataDisplay() {  
    const [datos, setData] = useState(nulo);  
    const [error, setError] = useState(null);  
  
    usarEfecto(() => {  
        función asíncrona fetchData() {  
            intentar {  
                constante respuesta = await fetch('https://api.example.com/data');  
                const jsonData = await respuesta.json();  
                establecerDatos(jsonData);  
            } atrapar (err) {  
                setError(err.mensaje);  
            }  
        }  
    }  
}
```

La programación síncrona es crucial para crear aplicaciones responsivas y eficientes en JavaScript y React. Desde devoluciones de llamadas hasta promesas y `async/await`: JavaScript ofrece varias formas de gestionar código asíncrono. En React, el gancho `useEffect` permite la integración fluida de operaciones asíncronas en componentes funcionales. Una gestión adecuada de errores, como el uso de bloques `try/catch`, garantiza una mejor experiencia de usuario en situaciones reales. Al comprender y aplicar estos conceptos, se pueden crear aplicaciones React más robustas y de mayor rendimiento que gestionen las operaciones asíncronas con fluidez.

Conclusión

React proporciona un marco potente y flexible para crear aplicaciones web modernas. A lo largo de estas siete secciones, exploramos los conceptos fundamentales de React, desde su arquitectura basada en componentes, la sintaxis JSX y la gestión de estados, hasta la gestión de eventos, los métodos de ciclo de vida y los Hooks. Además, profundizamos en las técnicas de programación asíncrona para crear aplicaciones más responsivas. Al comprender estos conceptos fundamentales y aplicar las mejores prácticas, podrá aprovechar al máximo el potencial de React y desarrollar aplicaciones web eficientes, fáciles de mantener y escalables.

En el próximo capítulo, Fundamentos de Next.js, se presentará el framework Next.js y sus ventajas clave. Le guiaremos a través del proceso de instalación y configuración, la creación de un nuevo proyecto Next.js y la comprensión de su estructura de carpetas. Aprenderá sobre la función esencial de las páginas en Next.js y cómo crear y renderizar una página básica. Finalmente, exploraremos la implementación de estilos CSS mediante módulos CSS, una potente función para gestionar los estilos de los componentes. Este capítulo le proporcionará los conocimientos básicos necesarios para crear aplicaciones Next.js sólidas y escalables.

Preguntas de opción múltiple

1. ¿Cuál es el propósito principal del DOM virtual en React?
 - A. Para mejorar el rendimiento reduciendo las manipulaciones directas del DOM
 - B. Para habilitar la programación asíncrona
 - C. Para simplificar la sintaxis del código JavaScript
 - D. Para reemplazar completamente el DOM real

2. ¿Cuáles son los dos tipos de componentes en React?

- A. Sincrónico y asincrónico
- B. Funcional y de clase
- C. Con Estado y sin Estado
- D. Padre e hijo

3. ¿Cuál es la principal diferencia entre JSX y HTML?

- A. JSX es un lenguaje de marcado, mientras que HTML es un lenguaje de programación.
- B. JSX permite incrustar expresiones JavaScript, mientras que HTML no.
- C. JSX no se puede utilizar para aplicar estilo, mientras que HTML sí.
- D. Se requiere un archivo separado para CSS, mientras que HTML no lo hace

4. ¿Cuál es el propósito principal de React Hooks?

- A. Para habilitar el uso del estado y otras características de React en funciones componentes
- B. Para mejorar el rendimiento reduciendo las manipulaciones directas del DOM
- C. Para simplificar la sintaxis del código JavaScript
- D. Para habilitar el uso de componentes de clase

5. ¿Qué React Hook se utiliza para realizar efectos secundarios en la función? ¿componentes?

- A. useState
- B. useEffect
- C. useContext
- D. useReducer

6. ¿Cuál es la principal ventaja de utilizar la sintaxis async/await sobre ¿Promesas?

- A. permite un código de mayor rendimiento
- B. hace que el código asincrónico sea más fácil de leer y escribir
- C. permite el uso del estado en componentes funcionales
- D. elimina la necesidad de gestión de errores

7. ¿En qué React Hook se utiliza principalmente la API de contexto?

- A. useState
- B. useEffect
- C. useContext
- D. useReducer

8. ¿Cuál es la función principal de las propiedades en los componentes de React?

- A. Gestión del estado dentro de un componente
- B. Habilitar el uso del estado y otras características de React en funciones componentes
- C. Pasar datos y funciones entre componentes
- D. Definición de la estructura y el diseño de un componente

9. ¿Qué método del ciclo de vida se llama solo una vez cuando se ejecuta un componente React? ¿montado?

- A. componentDidMount
- B. componentDidUpdate
- C. componentWillUnmount
- D. componentDidCatch

10. ¿Cuál de los siguientes NO es un estado válido en una promesa de JavaScript?

- A. Pendiente
- B. Cumplido
- C. Rechazado
- D. Completado

Respuestas

1 A	
2 B	
3 B	
4 A	
5 B	
6 B	
7 C	
8 C	
9 A	
10 D	

Capítulo 3

Fundamentos de Next.js

Introducción

En este capítulo, profundizamos en los fundamentos del framework Next.js. Comenzamos presentando Next.js, un potente framework basado en React que ofrece características únicas como renderizado del lado del servidor (SSR), generación de sitios estáticos (SSG) y regeneración estática incremental (ISR). Exploraremos las ventajas de usar Next.js y compararlo con otros marcos populares, al mismo tiempo que arroja luz sobre sus aplicaciones prácticas en proyectos del mundo real.

A continuación, lo guiaremos a través del proceso de configuración de su entorno de desarrollo, instalación de Next.js y creación de su primer proyecto Next.js. Lo ayudaremos a comprender la estructura central de un proyecto Next.js y le explicaremos cómo cada carpeta y archivo contribuye a su aplicación.

A medida que profundicemos, explicaremos el papel fundamental de las páginas en Next.js, su relación con el enrutamiento y el proceso de creación y renderizado de una página básica. Finalmente, presentaremos los módulos CSS y demostraremos cómo aprovecharlos para diseñar eficazmente su aplicación Next.js. Al finalizar este capítulo, comprenderá a fondo el funcionamiento básico de Next.js y estará bien preparado para empezar a crear sus propias aplicaciones Next.js.

Estructura

En este capítulo se tratarán los siguientes temas:

- Introducción al framework Next.js y sus ventajas
- Instalación, configuración y creación de un nuevo proyecto Next.js
- Comprender la estructura de carpetas de un proyecto Next.js
- Comprender el papel de las páginas en Next.js
- Creación y renderización de una página básica en Next.js
- Implementación de estilos CSS en Next.js usando módulos CSS

Presentamos el framework Next.js y sus ventajas

Next.js es un potente framework de código abierto basado en React para crear aplicaciones renderizadas del lado del servidor y generadas estáticamente. Desarrollado y mantenido por Vercel, está construido con JavaScript y aprovecha la potencia de React, Node.js y el ecosistema JavaScript. El framework está diseñado para ofrecer una experiencia de usuario optimizada con funciones como la división automática de código, la precarga y la recarga de código activo.

A continuación se muestra un ejemplo de un componente básico de Next.js :

```
// Un componente básico de Next.js

función Página de inicio() {
    regresar <div>¡Bienvenido a Next.js!</div>
}

}

exportar página de inicio predeterminada
```

Ventajas de usar Next.js

Next.js ofrece varias características atractivas que lo convierten en una opción destacada para el desarrollo web moderno. Estas características incluyen:

- Representación del lado del servidor (SSR): Next.js representa las páginas en el servidor y el archivo. Esta función acelera los envíos al cliente como una página inicial completa y mejora el SEO.

Generación de sitios estáticos (SSG): Next.js puede generar HTML estático durante la compilación, lo que reduce la necesidad de un servidor. Esta función es ideal para entradas de blog o sitios de documentación cuyo contenido no cambia con frecuencia.

Regeneración estática incremental (ISR): Es una combinación de SSR y SSG. ISR permite crear páginas estáticas que se pueden actualizar gradualmente tras el proceso de compilación, lo que proporciona un rendimiento óptimo con contenido actualizado.

- Enrutamiento automático Basado en el sistema de archivos en el directorio de páginas , Siguiente. js enruta automáticamente su aplicación, lo que reduce la configuración de enrutamiento manual.
- Rutas API: Next.js le permite crear sus rutas API junto con sus páginas, simplificando el desarrollo back-end.

Por ejemplo:

```
// Un ejemplo de ruta API en Next.js

exportar función predeterminada manejador(req, res) {
    res.status(200).json({ text: 'Hola desde la ruta de la API de Next.js' })
}
```

- Recarga de módulos en caliente: Next.js ofrece recarga de módulos en caliente lista para usar, lo que proporciona retroalimentación inmediata durante el desarrollo.
- Compatibilidad con CSS y Sass integrados: Next.js tiene soporte integrado para CSS y Sass, y también admite soluciones CSS-in-JS como componentes con estilo o emociones.
- Compatibilidad con TypeScript: Next.js admite TypeScript de forma predeterminada, lo que permite una tipificación fuerte en sus proyectos.

Comparación con otros marcos

Next.js se distingue de otros frameworks de JavaScript como Create React App (CRA) y Gatsby en varios aspectos. Si bien CRA es ideal para crear aplicaciones de página única (SPA), carece de las funciones SSR y SSG de Next.

js, que son cruciales para el SEO y el rendimiento. Gatsby, por otro lado, prioriza SSG, pero no ofrece compatibilidad nativa con SSR ni ISR. Además, el enrutamiento automático y las rutas API de Next.js lo convierten en una solución más completa para el desarrollo full-stack.

He aquí un ejemplo:

```
// Una ruta dinámica en Next.js

// páginas/publicaciones/[id].js

función Post({ post }) {
    // Publicación de renderizado...
}

exportar función asíncrona getServerSideProps(contexto) {
    const { id } = contexto.params
    // Obtener datos basados en id...
    devolver {props: {publicación}}
}

exportar publicación predeterminada;
```

Casos de uso reales de Next.js

Muchas empresas reconocidas utilizan Next.js en sus plataformas tecnológicas gracias a su versatilidad y robustez. Empresas como Netflix, GitHub y Uber han aprovechado el potencial de Next.js para crear aplicaciones web rápidas, escalables y optimizadas para SEO.

Estos usos en el mundo real validan la practicidad y la preparación empresarial del marco Next.js.

En resumen, Next.js es un marco integral que ofrece una combinación de rendimiento, experiencia de desarrollador y flexibilidad. Ya sea que esté creando un proyecto pequeño o una aplicación empresarial a gran escala, Next.js tiene las herramientas y las características para que el proceso sea eficiente y agradable.

El amplio conjunto de funciones de Next.js reduce la necesidad de configuraciones o plugins adicionales que podrían ser necesarios en otros frameworks. Sus capacidades de renderizado del lado del servidor permiten un mejor SEO, mientras que la generación de sitios estáticos y la regeneración estática incremental ofrecen diversas opciones para optimizar el rendimiento según sus necesidades específicas.

A continuación se muestra un ejemplo de generación de sitio estático en Next.js:

```
// Generación de sitios estáticos en Next.js
// páginas/publicaciones/[id].js
exportar función asíncrona getStaticPaths() {
    // Determinar las rutas en el momento de la compilación...
}

exportar función asíncrona getStaticProps({ parámetros }) {
    // Obtener datos según parámetros...
    devolver {props: {...}}
}

exportar función predeterminada Post({ ... }) {
    // Renderiza la publicación...
}
```

El sistema de enrutamiento integrado simplifica la creación de páginas, y la función de ruta PI nativa significa que puede manejar la funcionalidad del backend dentro del mismo marco, lo que da como resultado una experiencia de desarrollo de pila completa perfecta.

Además, Next.js ofrece compatibilidad inmediata con CSS y Sass, TypeScript y las soluciones CSS-in-JS más populares, eliminando así la necesidad de configurar estas funciones manualmente. Esta funcionalidad integrada, junto con la recarga automática de código activo, proporciona una excelente experiencia para el desarrollador.

Al comparar Next.js con otros frameworks populares como CRA y Gatsby, queda claro que ofrece un conjunto de herramientas más completo para crear aplicaciones web modernas. Si bien CRA y Gatsby tienen sus puntos fuertes, la versatilidad de Next.js lo convierte en un sólido candidato para muchos casos de uso.

La adopción de Next.js por parte de empresas destacadas como Netflix, Ber y Witch demuestra aún más sus capacidades y practicidad. Estas empresas requieren aplicaciones web de alto rendimiento, escalables y fiables, y su decisión de usar Next.js demuestra la capacidad del framework para satisfacer estos requisitos.

Ya sea que recién esté comenzando con el desarrollo de JavaScript o sea un desarrollador experimentado que busca optimizar su flujo de trabajo, Next.js ofrece una variedad de beneficios que lo convierten en una excelente opción para su próximo proyecto.

Instalación y creación de un nuevo proyecto Next.js

En esta sección, veremos los requisitos previos para instalar Next.js y analizaremos la estructura de carpetas de una aplicación Next.js.

Requisitos previos para instalar Next.js

Antes de empezar con Next.js, debemos asegurarnos de tener las herramientas necesarias instaladas en nuestro equipo. Node.js y npm (gestor de paquetes de Node) son cruciales para configurar un proyecto de Next.js. Node.js es un entorno de ejecución de JavaScript que nos permite ejecutar JavaScript en nuestro servidor, y npm es un gestor de paquetes para Node.js. Al momento de escribir este artículo, se recomiendan las últimas versiones LTS de Node.js (v14.xx o superior) y npm (v6.xx o superior).

Puedes comprobar si tienes Node.js y npm instalados y ver las versiones ejecutando los siguientes comandos en tu terminal:

```
intento  
nodo -v  
npm -v
```

Si no tienes Node.js instalado, puedes descargarlo del sitio web oficial de Node.js (<https://nodejs.org>).

Instalación de Next.js

Una vez que tenga Node.js y npm instalados, configurar una nueva aplicación Next.js es muy fácil, gracias al comando `create-next-app`, que configura una nueva aplicación Next.js con una plantilla de inicio predeterminada.

En tu terminal, navega hasta el directorio donde quieras que viva tu proyecto y ejecuta el siguiente comando:

```
intento  
npx crear-próxima-aplicación@última-mi-próxima-aplicación
```

En este comando, "my-next-app" es el nombre de tu nuevo proyecto. Reemplázalo con el nombre que deseas.

Creando nuestro primer proyecto Next.js

El comando `create-next-app` crea un nuevo directorio con el nombre de tu proyecto, configura los archivos y carpetas necesarios e instala las dependencias. Es básicamente una aplicación Next.js lista para usar.

Una vez completado el comando, navegue hasta el nuevo directorio del proyecto:

```
intento  
cd mi-próxima-aplicación
```

Para iniciar el servidor de desarrollo, utilice el comando `npm run dev`:

```
intento  
npm ejecutar dev
```

Abra su navegador y vaya a `http://localhost:3000`. ¡Debería ver su nueva aplicación Next.js funcionando!

Comprender la configuración inicial

El comando '`create-next-app`' configura una estructura de archivo Next.js estándar de la siguiente manera:

- páginas Este directorio contiene las páginas de su aplicación. Cada archivo corresponde a una ruta basada en su nombre de archivo.
- público : este directorio almacena archivos estáticos, como imágenes, que se sirven en la ruta URL raíz ('/').
- estilos Este directorio contiene archivos CSS globales. El archivo `globals.css` se incluye de forma predeterminada y se puede usar para estilos globales.
- `package.json` Este archivo contiene metadatos sobre su proyecto y sus dependencias.

Aquí está la estructura de directorio de un proyecto Next.js:

```
intento
mi próxima aplicación
    páginas
        aplicaciones
        API
        índice.s
    público
    estilos
        globals.css
        ome.module.css
    paquete.hijo
```

El directorio 'pages' incluye un archivo `index.js` , que corresponde a la ruta, y `_app.js`, un componente de aplicación personalizado que se utiliza para inicializar páginas. Puedes sobrescribirlo para controlar la inicialización de páginas o añadir estilos globales.

El directorio `api` en páginas es especial: es donde puedes crear funciones de backend directamente en tu aplicación Next.js.

Esta configuración inicial proporciona una base sólida para sus proyectos Next.js y aprenderá más sobre cada uno de estos componentes a medida que avance en su recorrido por Next.js.

Comprender la estructura de carpetas de un proyecto Next.js

Comprender la estructura de archivos y carpetas de un proyecto Next.js es crucial ya que proporcionará una mejor idea de la organización del proyecto y hará que el desarrollo sea más fluido.

Descripción general de la estructura de carpetas

Una aplicación Next.js recién creada tiene la siguiente estructura de carpetas:

```
mi próxima aplicación
  páginas
    aplicaciones
    API
    índice.s
  público
  estilos
    globals.css
    ome.module.css
  .gitignore
  paquete.hijo
  EDE.md
```

Repasemos cada uno de estos directorios y archivos para comprender mejor sus funciones.

Explorando el directorio 'páginas'

El directorio de páginas es uno de los directorios más importantes en un proyecto Next.js.

Aquí es donde agregarás todas las páginas para tu aplicación. Cada archivo dentro de este directorio corresponde a una ruta basada en su nombre de archivo.

Por ejemplo, si crea un archivo `about.js` dentro del directorio de páginas , será accesible en '`http://localhost:3000/about`'.

```
// páginas/acerca.js
exportar función predeterminada Acerca de() {
  volver <h1>Sobre nosotros</h1>
```

```
}
```

El directorio de páginas también incluye el archivo `_app.js`, un componente pp personalizado. Este archivo permite sobrescribir el componente pp predeterminado usado para inicializar las páginas. Puede usarlo para mantener el estado entre páginas, agregar CSS global o agregar un diseño global.

El directorio api dentro de las páginas es donde puedes crear tus rutas de API. Cada archivo dentro de este directorio corresponde a una ruta que permite crear una PI completa.

```
// páginas/api/hello.js
exportar función predeterminada manejador(req, res) {
    res.status(200).json({ texto: 'Hola' })
}
```

Explorando el directorio público

El directorio público es donde puedes agregar recursos estáticos como imágenes, fuentes o archivos robots.txt. Estos archivos se servirán desde la ruta raíz de R. Por ejemplo, si agregas un archivo 'logo.png' dentro del directorio público, puedes acceder a él a través de 'http://localhost:3000/logo.png'.

Este directorio también es útil para archivos como manifest.json, robots.txt y cualquier otro archivo estático que deseas agregar a la raíz de su dominio.

Explorando el directorio de estilos

Next.js admite la importación de CSS de forma predeterminada. El directorio de estilos es donde puedes agregar tus archivos CSS. De forma predeterminada, Next.js configura tu proyecto para que admita módulos CSS, lo que te permite importar CSS directamente a tus archivos avaScript.

Por ejemplo, puedes crear un archivo Button.module.css

```
/* Botón.modulo.css */
.botón {
```

```
relleno: 10px 20px;  
color de fondo: f3;  
color: fff;  
borde: ninguno;  
radio del borde: 5px;  
}
```

Y luego importarlo en tu componente:

```
// páginas/index.js  
importar estilos desde '../styles/Button.module.css'  
  
exportar función predeterminada Inicio() {  
    botón de retorno clase={styles.button}>click e/button>  
}
```

El archivo globals.css se incluye por defecto y se puede usar para estilos globales. Estos estilos se aplicarán a todas las páginas y componentes de la aplicación. También puede usar este archivo para importar otros archivos CSS como globales.

archivos ter

El archivo package.json contiene metadatos sobre el proyecto y enumera sus dependencias. Al ejecutar npm install <paquete>, el paquete se añade como dependencia.

Comprender el papel de las páginas en Next.js

En Next.js, una página es un componente de React asociado a una ruta según su nombre de archivo en el directorio de páginas . Este concepto es fundamental para Next.js y comprenderlo es crucial para crear aplicaciones con el framework.

Presentando páginas

En Next.js, cada archivo en el directorio de páginas se convierte en una ruta que se procesa y representa automáticamente. Digamos que tiene un archivo `pages/about.js`, será accesible en `'http://localhost:3000/about'`.

Aquí hay un ejemplo de una página básica:

```
// páginas/acercade.js
exportar función predeterminada Acerca de() {
    volver <h1>Sobre nosotros</h1>
}
```

Cuando navegue a `'/acerca de'`, se mostrará la función 'Acerca de'.

- Enrutamiento automático

Next.js sigue un enrutador basado en un sistema de archivos creado sobre el concepto de páginas. Cuando se agrega un archivo al directorio de páginas , automáticamente está disponible como ruta.

Los archivos y carpetas del directorio de páginas reflejan las rutas de las rutas. Por ejemplo, si crea un archivo `'agesostsf irstost.js'` , será accesible en `'postsfirst-post'`.

- Rutas dinámicas

Next.js admite rutas dinámicas. Si crea un archivo dentro del directorio de páginas con corchetes ('[]') en el nombre, se considerará una ruta dinámica.

Por ejemplo, `pages/posts/[id].js` coincidirá con `/posts/1'`, `'/posts/2'`, y así sucesivamente.

Puede acceder a la parte dinámica (id en este caso) en su componente usando el gancho `useRouter` desde `next/router`:

```
// páginas/publicaciones/[id].js
importar { useRouter } desde 'next/router'

exportar función predeterminada Post() {
```

```

enrutador constante = useRouter()

devolver <h1>Publicación: {router.query.id}</h1>
}

```

- Rutas anidadas

Puedes crear rutas anidadas añadiendo carpetas dentro del directorio de páginas . Por ejemplo, si creas un archivo `agesostsf/irstost.js` , será accesible en `/ostsf/irstost`.

- Rutas API

Next.js permite crear rutas API. Estas rutas son rutas del lado del servidor donde se puede implementar lógica de backend, como la obtención de datos de una base de datos o la gestión de envíos de formularios.

Las rutas PI se crean definiendo archivos dentro del directorio `pages/api` . Al igual que el resto del directorio `pages` , el sistema de archivos es el punto final principal de PI y los archivos se asocian con las rutas según su nombre.

Por ejemplo, si crea un archivo `pages/api/hello.js` , será accesible en `/api/Hola`:

```

// páginas/api/hello.js
exportar función predeterminada manejador(req, res) {
    res.status(200).json({ texto: 'Hola' })
}

```

- Páginas de error

Next.js tiene una página de error 404 integrada, que se muestra automáticamente si no se encuentra una ruta. Puedes personalizar la página de error 404 creando un archivo `pages/404.js`.

```
// páginas/404.js
```

```
exportar función predeterminada Custom404() {  
    devolver <h1>404 - Página no encontrada</h1>  
}
```

En conclusión, las páginas en Next.js son el corazón del sistema de enrutamiento. Comprender cómo crear y administrar páginas le permite aprovechar el potencial de Next.js para crear aplicaciones web dinámicas y de alto rendimiento.

Creación y renderización de una página básica en Next.js

Como hemos aprendido el papel de las páginas en Next.js, repasemos el proceso de creación y representación de una página básica.

Como se mencionó anteriormente, una página en Next.js es un componente de React exportado desde un archivo .js, .jsx, .ts o .tsx en el directorio de páginas . Cada página está asociada a una ruta según su nombre de archivo.

- Crear una nueva página

Crear una nueva página en Next.js es tan simple como crear un nuevo archivo en las páginas Directorio. Vamos a crear una nueva página "Acerca de" para nuestra aplicación. En tu proyecto, crea un nuevo archivo en pages/about.js y agrega el siguiente código:

```
// páginas/acercede.js  
exportar función predeterminada Acerde() {  
    devolver (  
        <div>  
            Acerde nosotros  
            <p>Somos un equipo de desarrolladores dedicados a crear aplicaciones web.  
            con  
            Next.js.</p>  
        </div>  
    )  
}
```

En el código anterior, creamos un componente funcional "Acerca de" que devuelve una estructura HTML simple. Este componente será nuestra página " Acerca de ".

- Visualización de la nueva página

Ahora que ha creado la nueva página, puede verla navegando a 'http://localhost:3000/about' en tu navegador. Deberías ver la página 'Acerca de' que acabas de crear.

- Vinculación entre páginas

Una de las operaciones más comunes en una aplicación web es navegar entre páginas. En Next.js, puedes usar el componente "Link" del paquete "next/link" para navegar entre páginas.

Creemos un enlace a la página "Acerca de" desde nuestra página de inicio . Abra las páginas/archivo index.js y actualice el contenido de la siguiente manera

```
// páginas/index.js
importar tinta desde next/link

exportar función predeterminada Inicio() {
    devolver (
        <div>
            Inicio
            ¡Bienvenido a nuestro sitio web!
            tinta href=/sobre>sobre s/tinta>
        </div>
    )
}
```

En este código, importamos el componente Enlace de next/link y lo usamos para enlazar a la página "Acerca de" . El atributo "href" se usa para especificar la ruta de destino del enlace.

Ahora, cuando navegues a http://localhost:3000, verás un enlace a Acerca de Al hacer clic en este enlace, accederá a la página " Acerca de" sin tener que actualizarla por completo, gracias a la navegación automática del lado del cliente de Next.js.

- Comprender el proceso de renderizado

Cuando solicitas una página en una aplicación Next.js, esto es lo que sucede:

1. El servidor recibe la solicitud: cuando navega a una ruta en su aplicación Next.js, el servidor recibe una solicitud para esa ruta.
2. El servidor obtiene el componente de la página: Luego, Next.js obtiene el componente React para esa ruta desde el directorio 'páginas'.
3. El servidor renderiza la página: El componente se renderiza a HTML en el servidor. Si el componente obtiene datos en una función `getServerSideProps` (que analizaremos en capítulos posteriores), la obtención de datos se realiza durante este paso.
4. El servidor envía la respuesta HTML: el resultado HTML se envía al cliente, junto con un código JavaScript mínimo para la hidratación.
5. La página es interactiva: El código JavaScript se ejecuta en el cliente, hidratando el HTML en una aplicación React totalmente interactiva.

Este proceso permite que Next.js proporcione tiempos de carga iniciales rápidos y funciones potentes como renderizado del lado del servidor y generación de sitios estáticos.

- Resumen

En esta sección, aprendiste a crear una página nueva en Next.js, a visualizarla y a enlazar diferentes páginas. También comprendiste el proceso de renderizado en Next.js. Con estos conceptos básicos, puedes empezar a crear aplicaciones web multipágina con Next.js.

- Creación de un componente de diseño

A menudo, querrás compartir componentes, como encabezados o pies de página, en varias páginas. Next.js no ofrece un sistema de diseño integrado, pero crear uno propio es sencillo.

Creemos un componente de diseño que represente una barra de navegación con enlaces a la Páginas de inicio y Acerca de :

```
// componentes/Layout.js
importar tinta desde next/link

exportar función predeterminada Diseño({ hijos }) {
```

```
devolver (
  <div>
    <navegación>
      tinta href=/>ome/tinta>
      tinta href=/sobre>sobre/tinta>
    </nav>
    {niños}
  </div>
)
}
```

En el componente Diseño , renderizamos una barra de navegación y los componentes secundarios pasan a Diseño.

Ahora, puedes usar este componente Diseño en tus páginas:

```
// páginas/index.js
importar diseño desde '../componentes/Layout'

exportar función predeterminada Inicio() {
  devolver (
    <Diseño>
      Inicio
      ¡Bienvenido a nuestro sitio web!
    </Diseño>
  )
}

// páginas/acercade.js
```

```

importar diseño desde '../componentes/Layout'

exportar función predeterminada Acerca de() {
    devolver (
        <Diseño>
            Acerca de nosotros
            <p>Somos un equipo de desarrolladores dedicados a crear aplicaciones web.
        con
        Next.js.</p>
        </Diseño>
    )
}

```

Con esta configuración, cada página se envuelve con el componente Diseño , lo que proporciona un diseño consistente en todas las páginas.

Crear y renderizar páginas en Next.js es un proceso sencillo. Aprovechando el enrutamiento automático y las potentes funciones de renderizado del framework, puedes crear rápidamente aplicaciones web dinámicas y de alto rendimiento.

En la siguiente sección, veremos cómo puedes diseñar tu aplicación Next.js usando módulos CSS.

Implementación de estilos CSS en Next.js usando módulos CSS

El estilo es una parte esencial de cualquier aplicación web. En Next.js, se pueden importar CSS directamente a archivos avaScript gracias a la compatibilidad integrada con CSS y Sass. En esta sección, nos centraremos en el uso de módulos CSS, un archivo CSS en el que todos los nombres de clases y animaciones tienen un alcance local por defecto.

- Introducción a los módulos CSS

Los módulos CSS son archivos CSS donde todos los nombres de clases y animaciones tienen un alcance local predeterminado. La idea clave de los módulos CSS es garantizar que todos los estilos de un solo componente:

Vivir en un lugar (un archivo .module.css)

Aplicar solo a ese componente y nada más

- Creación de un módulo CSS

Para crear un módulo CSS, necesitas crear un archivo `.module.css` . En el directorio de estilos de tu proyecto , crea un nuevo archivo llamado `About.module.css`:

```
/* Acerca de.module.css */  
  
.contenedor {  
    margen: 20px;  
    relleno: 20px;  
    borde: 1px sólido ddd;  
}  
  
.título {  
    color: f3;  
    tamaño de fuente: 2em;  
    margen inferior: 10px;  
}  
  
.contenido {  
    tamaño de fuente: 1.2em;  
    altura de línea: 1,6em;  
}
```

En este archivo, definimos tres clases CSS: contenedor, título y contenido.

- Cómo utilizar un módulo CSS

Para usar un módulo CSS, se importa en un archivo avaScript. El módulo CSS importado puede referenciarse como un objeto. Apliquemos estos estilos a nuestra página "Acerca de" :

```
// páginas/acercade.js  
importar estilos desde '../styles/About.module.css'  
importar diseño desde '../componentes/Layout'
```

```
exportar función predeterminada Acerca de() {  
    devolver (  
        <Diseño>  
        <div className={estilos.contenedor}>  
            <h1 className={styles.title}>Acerca de nosotros</h1>  
            <p className={styles.content}>Somos un equipo de desarrolladores dedicados a  
            crear aplicaciones web con Next.js.</p>  
        </div>  
    </Diseño>  
)  
}
```

En este archivo, importamos el módulo CSS que creamos anteriormente y usamos las clases en nuestro componente. Los estilos se aplican solo a este componente, incluso si otros componentes usan los mismos nombres de clase.

- Composición y clases globales

Los módulos CSS te permiten crear estilos, lo que significa que puedes usar estilos de otros módulos CSS en tu aplicación actual. También admiten estilos globales, que se aplican a toda la aplicación.

A continuación se muestra un ejemplo de uso de composición y estilos globales:

```
/* Global.module.css */  
.título {  
    color: f3;  
}  
  
/* estilos/Acerca de.module.css */  
@import './Global.module.css';  
  
.contenedor {  
    compone: título de './Global.module.css';  
    tamaño de fuente: 2em;  
}
```

En este ejemplo, el archivo `About.module.css` importa estilos de `lobal.module.css'` y utiliza la clase de título en su clase contenedora .

Los módulos CSS proporcionan una forma potente y flexible de darle estilo a sus aplicaciones Next.js. Al delimitar los estilos localmente de manera predeterminada, los módulos CSS garantizan que los estilos no se filtren entre componentes, lo que le ayuda a evitar conflictos de estilos y a mantener sus estilos organizados.

Conclusión

En este capítulo, presentamos Next.js, un framework robusto que mejora las capacidades de React, y exploramos sus características y ventajas clave. Hemos cubierto la instalación y configuración de un nuevo proyecto de Next.js, comprendido su estructura y profundizado en la función de las páginas. También hemos recorrido el proceso de creación y renderizado de una página básica y aprendido a implementar estilos CSS mediante módulos CSS. Con estos conocimientos básicos, estarás bien preparado para empezar a crear tus propias aplicaciones Next.js y explorar en profundidad las funciones avanzadas de este potente framework.

En el próximo capítulo, "Next 13 - Conceptos básicos", profundizaremos en las nuevas y emocionantes funciones y mejoras introducidas en la versión 13 de Next.js. Exploraremos cómo estas actualizaciones optimizan aún más tu experiencia de desarrollo y mejoran el rendimiento de tu aplicación. Desde el nuevo sistema de enrutamiento y las mejoras en las funciones de borde hasta el middleware y las mejoras en los componentes de servidor de React, obtendrás una comprensión completa del estado actual del framework, lo que te permitirá aprovechar al máximo sus capacidades avanzadas.

Preguntas de opción múltiple

1. ¿Qué es Next.js?
 - A. Una biblioteca de JavaScript para crear interfaces de usuario
 - B. Un marco CSS para diseñar páginas web
 - C. marco de back-end completo
 - D. Un marco de React para crear aplicaciones JavaScript
2. ¿Qué comando se utiliza para crear una nueva aplicación Next.js?
 - A. `npm create-next-app`
 - B. `npm start-next-app`

- C. npx crear-próxima-aplicación
- D. npx iniciar-siguiente-aplicación
3. ¿Qué tipo de archivo representa una página en Next.js?
- A. Archivo '.js' o '.jsx' en el directorio de las páginas
- B. Archivo '.css' en el directorio de estilos
- Archivo C. '.md' en el directorio markdown
- D. Archivo '.html' en el directorio público
4. ¿Cómo gestiona Next.js el enrutamiento?
- A. Definiendo manualmente rutas en un archivo route.js
- B. Automáticamente basado en el directorio 'páginas'
- C. Utilizando el sistema de enrutamiento Express.js
- D. Next.js no maneja el enrutamiento
5. ¿Cómo puedes navegar entre páginas en Next.js?
- A. Mediante el uso del elemento HTML '<a>'
- B. Utilizando la función 'navegar' del paquete 'next/navigate'
- C. Utilizando el componente de tinta del paquete 'next/link'
- D. Utilizando el método 'push' del paquete 'next/router'
6. ¿Qué sucede cuando solicitas una página en una aplicación Next.js?
- A. El navegador descarga la aplicación completa.
- B. El servidor envía el HTML de la página solicitada.
- C. El servidor envía el código JavaScript de la página solicitada
- D. El navegador renderiza la página desde cero.
7. ¿Qué es un módulo CSS en Next.js?
- Un archivo CSS en el que todos los nombres de clases y animaciones tienen un alcance global de forma predeterminada
- B. Archivo CSS en el que todos los nombres de clases y nombres de animaciones tienen un alcance local de forma predeterminada
- DO. Archivo avaScript en el que todos los estilos se definen como objetos avaScript
- D. Archivo CSS que se puede importar a un archivo avaScript como una cadena

8. ¿Cómo se aplica una clase de un módulo CSS en un componente Next.js?

- A. Utilizando el atributo 'className' y el nombre de la clase
- B. Utilizando el atributo 'style' y el nombre de la clase
- C. Al utilizar el atributo 'className' y hacer referencia a la clase desde el objeto de estilos importados
- D. Al utilizar el atributo 'style' y hacer referencia a la clase desde el objeto de estilos importados

9. ¿Cuál es el propósito del componente 'Layout' en Next.js?

- A. o definir la estructura de una página
- B. Para compartir componentes comunes en varias páginas
- C. Para darle estilo a una página.
- D. Para manejar la obtención de datos para una página.

10. ¿Cuál de las siguientes NO es una característica de Next.js?

- A. Enrutamiento automático
- B. Ámbito local para módulos CSS C.
- Representación del lado del cliente
- D. Traducción automática de código JavaScript a TypeScript

Respuestas

1 D	
2 C	
3 A	
4 B	
5 °C	
6 B	
7 B	
8 C	
9 B	
10 D	

Capítulo 4

Next.js 13

Introducción

En este capítulo, profundizamos en la última versión de Next.js, v13, anunciada en la Next.js Conf. Next.js representa una evolución significativa en el mundo de los frameworks de React, presentando conceptos y herramientas novedosas que redefinen la forma en que los desarrolladores crean aplicaciones web. Está diseñado para optimizar el proceso de desarrollo, proporcionando un amplio conjunto de funciones que facilitan la creación de aplicaciones web escalables, de alto rendimiento y compatibles con S. Con la introducción de funciones como Componentes de Servidor y TurboPack, Next.js 13 se sitúa a la vanguardia del desarrollo web moderno, ampliando los límites de lo posible con React y JavaScript.

Uno de los cambios clave es la introducción del enrutador de la aplicación, un cambio fundamental respecto de la estructura de archivos de la página tradicional. Este nuevo enfoque de enrutamiento proporciona un control granular sobre las transiciones de página y mejora el rendimiento general de la aplicación. Además, Next.js ha renovado la forma en que maneja las imágenes, aprovechando una nueva generación de tecnología de optimización de imágenes que mejora tanto la velocidad como la calidad visual.

En esencia, Next.js 13 no es sólo una actualización: es un salto cuántico que reimagina el núcleo mismo de Next.js, marcando el comienzo de una nueva era en el desarrollo web. Las siguientes secciones profundizarán en los detalles de estas nuevas y emocionantes características, ofreciendo una exploración integral de las capacidades de Next.js 13.

Estructura

En este capítulo se tratarán los siguientes temas:

- Configuración de una aplicación Next.js 13
- Enrutador de aplicaciones
- Componentes de cliente y servidor

- Enrutamiento
- Representación
- Obtención de datos

Configuración de una aplicación Next.js 13

o descargue la última versión de Next.js que es... a partir de ahora instale Node.js , primero, en su sistema y luego escriba el siguiente comando

```
npx crear-siguiente-aplicación@última
```

Recibirás las siguientes indicaciones:

- ¿Cómo se llama tu proyecto? my-app
- ¿Te gustaría añadir TypeScript a este proyecto? Sí/No
- ¿Te gustaría usar ESLint en este proyecto? Sí/No
- ¿Te gustaría usar Tailwind CSS en este proyecto? Sí/No
- ¿Te gustaría usar el directorio `src/` con este proyecto? S/N
- ¿Qué alias de importación le gustaría configurar? @/*

Después de las indicaciones, create-next-app creará una carpeta con el nombre de su proyecto e instalará las dependencias necesarias.

Si abre el archivo package.json, verá los siguientes scripts

```
{
  "guiones": {
    "dev": "próximo desarrollador",
    "build": "próxima compilación",
    "inicio": "próximo inicio",
    "pelusa": "siguiente pelusa"
  }
}
```

Estos scripts hacen referencia a las diferentes etapas del desarrollo de una aplicación Next.js 13:

- dev: ejecuta next dev para iniciar Next.js en modo de desarrollo.
- build: ejecuta next build para compilar la aplicación para uso en producción.
- start: ejecuta next start para iniciar un servidor de producción Next.js.
- lint : ejecuta next lint para configurar la configuración Sint integrada de Next.js.

Abra el archivo layout.tsx en la carpeta de la aplicación y verá el siguiente contenido

```
importar ./globals.css
importar { nter } desde next/font/google

const inter = nter({ subconjuntos: [latín] })

exportar const metadatos = {
    Título: crear ext pp,
    Descripción: Generado por crear la siguiente aplicación.
}

exportar función predeterminada RootLayout({
    niños,
}: {
    hijos: React.ReactNode
}) {
    devolver (
        html lang=es>
            cuerpo clase={inter.clase}>{hijos}/cuerpo>
        /html>
    )
}
```

Ahora abra la página page.tsx en la carpeta de la aplicación y reemplace su contenido con el siguiente código:

```
exportar función predeterminada age() {  
    devolver h1>ello, ext.s 13/h1>;  
}
```

Escribe npm run dev en tu consola para iniciar el servidor de desarrollo y visita <http://localhost:3000> en tu navegador. Verás la siguiente pantalla.

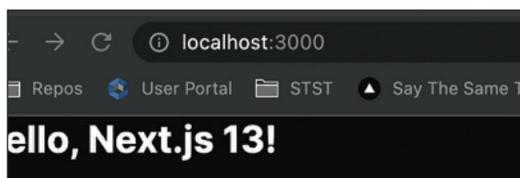


Figura 4.1: pantalla localhost

Hemos configurado con éxito una aplicación Next.js en nuestra computadora.

Enrutador de aplicaciones

Next.js 13 introduce una nueva estructura de App Router basada en componentes de servidor de React, que admite enrutamiento anidado, diseños compartidos, gestión de errores y estados de carga. App Router funciona dentro de una nueva carpeta llamada app, que puede funcionar junto con el directorio pages. Esto facilita la adopción gradual del nuevo pp Router y permite al desarrollador adoptar los nuevos conceptos a su propio ritmo. En este capítulo, nos centraremos únicamente en aprender las nuevas funciones introducidas en la versión 13.

Olers y archivos

En el enrutador pp, las carpetas se utilizan para definir rutas. Una ruta es una ruta única compuesta por carpetas anidadas que siguen la jerarquía desde la carpeta raíz de nivel superior hasta la última carpeta hoja que contiene un archivo page.js. Page.js se utiliza para definir la l mostrada para esa ruta en particular. Por ejemplo, si visitamos el enlace <https://mathewdony.comdashboard> en nuestro navegador para una aplicación Next.js 13, la aplicación renderizará el l desde el archivo page.tsx ubicado dentro de la carpeta del panel en la carpeta de la aplicación.

Aquí está la estructura de carpetas para my-app:

```
mi aplicación
  aplicación
    panel
      página.tsx
```

Convenciones de archivos

Next.js proporciona un conjunto de archivos especiales para crear I con un comportamiento específico en rutas anidadas. Veamos los principales que necesitamos:

- `page.js`: Se utiliza para representar la I de una ruta y hacer que la ruta sea accesible públicamente para un usuario.
- `layout.js`: Se utiliza para crear una interfaz de usuario compartida para un segmento y sus elementos secundarios. Un layout envuelve una página o un segmento secundario y puede usarse para renderizar elementos compartidos entre varias rutas, como un menú de navegación o un componente de encabezado.
- `loading.js`: se utiliza para crear una interfaz de usuario de carga para un segmento y sus hijos. `loading.js` muestra una I de carga mientras se cargan y funciona envolviendo una página o un segmento secundario en un React Suspense Boundary.
- `error.js`: se utiliza para crear una interfaz de usuario de error para un segmento y sus hijos. `error.js` muestra la I de error si se detecta un error y funciona envolviendo una página o un segmento secundario en un React Error Boundary.

Componentes de cliente y servidor

Con el lanzamiento de Next.js, se introdujo un método híbrido para crear aplicaciones que utiliza componentes de servidor y de cliente. Un principio de React, denominado "Componentes de servidor", busca diseñar aplicaciones sin paquetes de JavaScript y con un modelo mental basado en el servidor para interfaces de usuario modernas.

El componente del cliente se obtiene y se representa en el navegador del cliente. Mientras que un componente de servidor se obtiene y se procesa en el servidor.

Ahora, examinemos un programa sin un componente de servidor. En este escenario, un servidor responde a la solicitud de un usuario de una página web enviando información al navegador. El navegador descarga JavaScript al crear la página web. Esto implica que el navegador, al igual que los paquetes npm, debe instalar todo el código JavaScript necesario para crear la página web. Debido a todas las cascadas de solicitud-respuesta, la carga puede tardar mucho tiempo y ofrecer una mala experiencia de usuario.

El componente de servidor es útil en esta situación. El navegador solo necesita descargar archivos JavaScript para los componentes de cliente, mientras que un componente se define como componente de servidor, ya que el servidor lo compila y devuelve el HTML renderizado para el componente de servidor. Esto puede acelerar la renderización del navegador y mejorar la experiencia del usuario.

Next.js nos proporciona un sistema donde podemos especificar si un componente debe ser un Componente de Servidor o de Cliente a nivel de componente.

En el nuevo directorio de aplicaciones de Next.js, todos los componentes son componentes de servidor por defecto. Esto significa que todos los componentes y páginas se renderizan en el servidor, siempre que se especifique que el componente se renderice en el lado del cliente. Así es como se ve un componente de servidor en Next.js 13:

```
exportar función predeterminada age() {  
    devolver h1>ello, ext.s 13/h1>;  
}
```

Puedes ver en el código anterior que es solo un componente React normal, pero se convierte automáticamente en un componente de servidor ya que todos los componentes son componentes de servidor de forma predeterminada en Next.js 13.

Next.js recomienda usar componentes de servidor hasta que necesite usar componentes de cliente. Por ejemplo, los ganchos de React, como useState(), useEffect y useContext(), solo están disponibles en el lado del cliente. Además, si necesita acceder a elementos relacionados con el navegador, como eventos OC, ventanas o PI del navegador, debe usar el componente de cliente.

Si necesitamos agregar un contador a nuestra aplicación, podemos importar useState en nuestro componente React de la siguiente manera:

```
importar eact, { useState } de react;  
exportar función predeterminada age() {  
    const [count, setTount] = useState();  
    retorna h1>número actual: {count}/h1>;  
}
```

Pero el código anterior nos dará el siguiente mensaje de error en el navegador, como se muestra en la Figura 4.2:

```
Failed to compile
./app/page.tsx
ReactServerComponentsError:
You're importing a component that needs useState. It only works in a Client Component but
none of its parents are marked with "use client", so they're Server Components by default.

  ,-[/Users/mchittezhath/Downloads/my-app/app/page.tsx:1:1]
1 | import React, { useState } from "react";
  :           ^^^^^^
2 |
3 | export default function Page() {
4 |   const [count, setCount] = useState(0);
`-----


Maybe one of these should be marked as a client entry with "use client":
  ./app/page.tsx ⚡

An error occurred during the build process and can only be dismissed by fixing the error.
```

Figura 4.2: Mensaje de error

Para ejecutar el código anterior, necesitamos convertir nuestro Componente de Servidor en un Componente de Cliente. Esto se puede hacer agregando la directiva `useClient` en la parte superior de nuestro archivo de componente de la siguiente manera

utilizar cliente;

```
importar eact, { useState } de react;
```

```
exportar función predeterminada age() {
  const [count, setTount] = useState();
  retorna h1>número actual: {count}/h1>;
}
```

Esto resuelve el error y la aplicación funciona normalmente de la siguiente manera

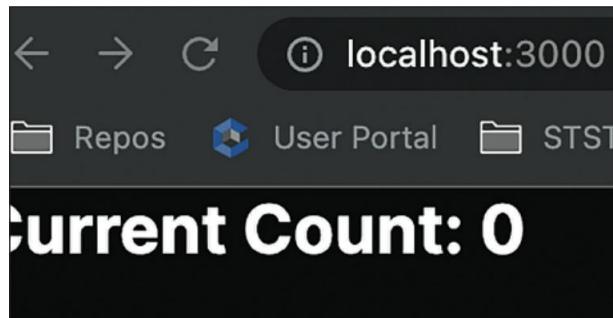


Figura 4.3: La aplicación funciona normalmente después de la resolución del error

Enrutamiento

Next.js utiliza un enrutador basado en un sistema de archivos en el que cada carpeta define una ruta. Cada carpeta representa un segmento de ruta que se asigna a un segmento de URL, y se pueden crear rutas anidadas anidando carpetas una dentro de otra.

Rutas dinámicas

Al desarrollar una aplicación web, puede ocurrir que el desarrollador o usuario desconozca con antelación los nombres exactos de los segmentos y desee crear rutas a partir de datos dinámicos. En tales casos, podemos usar segmentos dinámicos que se resuelven a petición o se pre-renderizan durante la compilación.

Se puede crear un segmento dinámico en Next.js envolviendo el nombre de una carpeta en corchetes.

Por ejemplo, si desea representar datos dinámicos que deben recuperarse del segmento URL, como app/blog/[slug]/page.js, donde slug es el segmento dinámico para las publicaciones del blog, puede escribir page.js de la siguiente manera

```
exportar función predeterminada age({params}) {  
devuelve div>y ost con D {params.slug}/div>;  
}
```

Si visitamos la URL app/blog/1234, obtendremos el valor 1234 dentro de la propiedad params, como se muestra en el ejemplo anterior.

Manejo de errores

La convención del archivo error.js permite a los desarrolladores gestionar errores de ejecución en rutas anidadas. Se puede usar para encapsular automáticamente una ruta y sus elementos secundarios anidados en un límite de error de React. Esto permite a los desarrolladores crear un error l adaptado a ese segmento específico. La principal ventaja de la convención del archivo error.js es que permite aislar errores sin afectar las partes restantes de la aplicación. Un error l se puede agregar fácilmente creando un archivo error.js dentro de un segmento de ruta.

exportando un componente React.

Middleware

Podemos ejecutar código usando middleware antes de que una solicitud haya terminado. La respuesta puede modificarse en función de la solicitud entrante reescribiendo, redirigiendo, cambiando los encabezados de solicitud o respuesta o simplemente respondiendo.

Representación

La ruta se puede renderizar estática o dinámicamente en Next.js. En una ruta estática, los componentes se renderizan en el servidor durante la compilación. Este trabajo se almacena en caché y se reutiliza en solicitudes posteriores, mientras que, en una ruta dinámica, los componentes se renderizan en el servidor durante la solicitud.

Representación estática

La representación estática, que es el patrón de representación predeterminado para una aplicación Next.js, mejora el rendimiento porque todo el trabajo de representación se realiza con anticipación y se puede servir desde una CDN de red de entrega de contenido geográficamente más cercana al usuario.

También podemos optar por la representación dinámica mediante el uso de una función dinámica o la obtención dinámica de datos en un diseño o página, lo que hará que Next.js represente toda la ruta dinámicamente en el momento de la solicitud.

Renderizado dinámico

Next.js cambiará a renderizar dinámicamente toda la ruta en el momento de la solicitud si se detecta una solicitud de función dinámica, etc. , en modo sin caché durante la renderización estática. Cualquier solicitud de datos almacenados en caché puede reutilizarse durante la renderización dinámica. Las solicitudes de obtención de datos dinámicos son solicitudes con la opción de caché establecida en "no-store" o "revalidate". En la siguiente sección, exploraremos más patrones de obtención de datos.

Obtención de datos

React y Next.js introdujeron una nueva forma de obtener y administrar datos en su aplicación. El nuevo sistema de obtención de datos funciona en el directorio de la aplicación y está construido sobre etc.

etc. es un eb PI utilizado para buscar recursos remotos que devuelve una promesa.

React extiende fetch para proporcionar eliminación automática de duplicados de solicitudes, y Next.js extiende el objeto de opciones de fetch para permitir que cada solicitud establezca su propio almacenamiento en caché y revalidación.

async y await en componentes del servidor

El siguiente código es un ejemplo del uso de async y await para obtener datos en el servidor.

Componentes:

```
función asíncrona getData() {  
  const res = await fetch(https://api.example.com/...);  
  
  si (res.ok) {  
    // Esto activará el límite de error `error.js` más cercano  
    lanzar nuevo Error(no se pudo obtener los datos);  
  }  
  
  devolver res.son();  
}  
  
exportar función asíncrona predeterminada age() {  
  const datos = esperar obtenerDatos();  
  
  devolver p>{datos}/p>;  
}
```

Obtención de datos estáticos

De forma predeterminada, la búsqueda se realizará automáticamente y almacenará los datos indefinidamente.
fetch(<https://...>); // caché: forcecache es el predeterminado

Obtención dinámica de datos

Para obtener datos nuevos en cada solicitud de búsqueda, utilice la opción de no almacenar en caché :

```
buscar(https://..., { caché: nostore });
```

Conclusión

En este capítulo, aprendimos sobre Next.js 13, la última versión del marco web, y exploramos sus nuevas características y ventajas clave. Hemos visto cómo ha adoptado el enrutador de aplicaciones, lo que facilita que los desarrolladores no se preocupen.

sobre la representación y el almacenamiento en caché de una aplicación Next.js, ya que estos casos son manejados automáticamente por el marco. Con todas estas nuevas características, Next.js se ha convertido en la opción obvia para crear una aplicación web en 2023.

En el próximo capítulo, Optimización de aplicaciones Next.js, aprenderemos sobre las diferentes estrategias que se pueden utilizar para optimizar una aplicación Next.js, lo que resulta en un mejor rendimiento y una mejor experiencia para el desarrollador. Next.js proporciona componentes integrados como `<Image />` y `L`, que ofrecen a los desarrolladores numerosas funciones listas para usar para optimizar sus aplicaciones web.

Preguntas de opción múltiple

1. ¿Cuál es el comando para iniciar un servidor de desarrollo Next.js localmente?
 - A. siguiente de
 - B. próxima construcción
 - C. próximo inicio
 - D. siguiente pelusa

2. ¿Cuál es el directorio principal en Next.js?
 - A. carpeta de diseño
 - B. Directorio de páginas
 - C. enrutador de aplicaciones
 - D. carpeta pública

3. ¿Cómo se llama una ruta única de carpetas anidadas?
 - A. Ruta
 - B. Página
 - C. Aplicación
 - D. Carpeta

4. ¿Cuál es el archivo desde el cual se renderiza la I para una ruta?
 - A. error.js
 - B. cargando.js
 - C. wait.js

- D. page.js
5. ¿Cuál es el tipo predeterminado de un componente en Next.js?
- A. Componente de inicio
 - B. Componente del servidor
 - C. Componente de la aplicación
 - D. Componente del enrutador
6. ¿Cuál es la directiva para convertir un componente de servidor en un componente cliente?
- A. utilizar cliente
 - B. utilizar servidor
 - C. atender al cliente
 - D. establecer cliente
7. ¿Qué archivo proporciona el límite de error de React en Next.js?
- A. boundary.js
 - B. fault.js
 - C. error.js
 - D. error-boundary.js
8. ¿Cuál es el comportamiento predeterminado de fetch PI en Next.js?
- A. Almacenamiento en caché forzado
 - B. No almacenar en caché
 - C. Almacenamiento en caché incremental
 - D. Almacenamiento en caché periódico
9. ¿Cuál es el comando para crear una aplicación Next.js para producción?
- A. construir producción
 - B. próximo desarrollo
 - C. próxima construcción

Next.js 13

103

D. próxima ejecución

10. ¿Cuál es la convención para establecer una ruta dinámica para id'?

Ayuda }

Licitación]

C. (id)

D. id>

Respuestas

1 A	
2 C	
3 A	
4 D	
5 B	
6 A	
7 C	
8 A	
9 C	
10 B	

Capítulo 5

Optimización de

aplicaciones Next.js

Introducción

Como desarrollador experto, sabes que tus aplicaciones Next.js necesitan optimizarse para obtener el máximo rendimiento. Afortunadamente, Next.js es un framework de código abierto que facilita la creación de aplicaciones React ultrarrápidas y renderizadas en servidor. Con funciones listas para usar como la división automática de código y la exportación estática, Next...

js se encarga del trabajo pesado, permitiéndote concentrarte en crear aplicaciones escalables y de alto rendimiento que encantarán a los usuarios. En este capítulo, exploraremos el apasionante mundo de Next.js, destacando sus numerosas funciones y explicando por qué es tan importante optimizar tus aplicaciones Next.js. ¡Prepárate para impulsar tus habilidades de desarrollo!

Estructura

En este capítulo cubriremos los siguientes temas:

- Importancia y beneficios de optimizar las aplicaciones Next.js
- Agregar metadatos a las páginas mediante el componente de encabezado
- Implementación del servicio de archivos estáticos en Next.js
- Comprender el uso del componente de imagen Next.js para imágenes mejoramiento
- Comprender la arquitectura de Next.js y cómo funciona
- Configuración de Next.js para un rendimiento óptimo
- Implementación de renderizado del lado del servidor (SSR) y pre-renderizado
- División de código e importaciones dinámicas
- Almacenamiento en caché y mejora de la obtención de datos

- Análisis y reducción del tamaño del paquete
- Estrategias de implementación y mejores prácticas
- Monitoreo y optimización del rendimiento

Mortane y los beneficios de optimizar las aplicaciones Next.js

Optimizar las aplicaciones Next.js es importante por varias razones. En primer lugar, puede mejorar el rendimiento general de la aplicación, lo que se traduce en una mayor interacción y retención del usuario. En segundo lugar, puede reducir la carga del servidor, lo que se traduce en un ahorro de costes para el desarrollador. Por último, optimizar las aplicaciones Next.js puede ayudar a preparar la aplicación para el futuro, garantizando su escalabilidad y mantenimiento a lo largo del tiempo.

Existen varios beneficios de optimizar las aplicaciones Next.js, incluidos:

- más errores: optimizar las aplicaciones Next.js puede ayudar a mejorar el rendimiento de la aplicación, lo que da como resultado tiempos de carga más rápidos y una mejor experiencia del usuario.
- Reducción de la carga del servidor: optimizar las aplicaciones Next.js puede ayudar a reducir la carga en el servidor, lo que puede resultar en ahorros de costos para el desarrollador.
- treroofing: Optimizar las aplicaciones Next.js puede ayudar a preparar la aplicación para el futuro, asegurando que siga siendo escalable y mantenible a lo largo del tiempo.
- Mayor productividad del desarrollador: optimizar las aplicaciones Next.js puede ayudar a aumentar la productividad del desarrollador, ya que puede ayudar a identificar y resolver problemas de rendimiento más rápidamente.

Agregar metadatos a las páginas mediante el componente Head

En esta sección, exploraremos la sintaxis y el uso del componente Head en Next.js. Aprenderemos a establecer el título de la página, definir metaetiquetas, agregar hojas de estilo externas e incluir bibliotecas JavaScript. Además, abordaremos las mejores prácticas para estructurar y organizar metadatos y maximizar su impacto.

El componente Head de Next.js permite definir metadatos para páginas individuales. Estos metadatos incluyen elementos como el título, la descripción y diversas etiquetas que optimizan la optimización para motores de búsqueda (SEO) y mejoran la visualización de las páginas al compartirlas en redes sociales. Mediante el componente Head...

De esta forma, puedes asegurarte de que tus páginas web tengan una buena representación en diferentes plataformas y una mayor visibilidad. Para usar el componente Head, debes importarlo desde el módulo next/head .

Consideremos un ejemplo en el que queremos establecer el título y la descripción de una página específica.

importar Head desde 'next/head';

función Página de inicio() {

devolver (

<div>

<Cabeza>

 Libros destacados - NextCommercePro

 <meta name="description" content=" Libros destacados en nuestra colección" />

</Cabeza>

 /* Resto del contenido de la página */

</div>

);

}

exportar página de inicio predeterminada;

La figura 5.1 muestra la página de inicio predeterminada con meta y título:

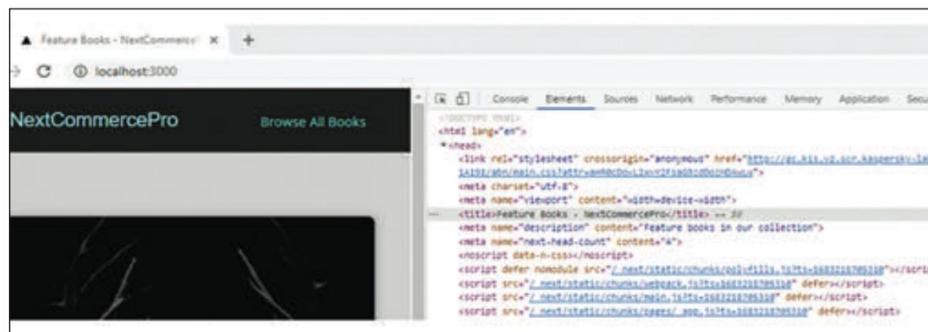


Figura 5.1: Página de inicio predeterminada

En el fragmento de código, importamos el componente Head del módulo 'next/head'. Dentro del componente HomePage, envolvemos los elementos de metadatos deseados dentro del componente Head. En este caso, establecemos el título como " Mi Siguiente".

js App - Inicio y meta descripción de ¡ Bienvenido a mi aplicación Next.js!

Optimización de aplicaciones Next.js

El componente Head también permite añadir otras etiquetas de metadatos. Por ejemplo, se pueden incluir etiquetas Open Graph (OG) para facilitar su uso compartido en redes sociales:

importar Head desde 'next/head';

```
función BookPage({ libro }) {
```

devolver (

<div>

<title>{ libro.título}</title>

```
<meta name="description" content={ libro.extracto} />
```

```
<meta propiedad="og:título" contenido={libro.título} />
```

```
<meta propiedad="og:description" contenido={libro.extracto} />
```

```
<meta property="og:image" content={ libro.miniatura} />
```

</Cabeza>

{/* Resto del contenido de la página */}

```
</div>
```

La figura 5.2 muestra la página de detalles del producto.

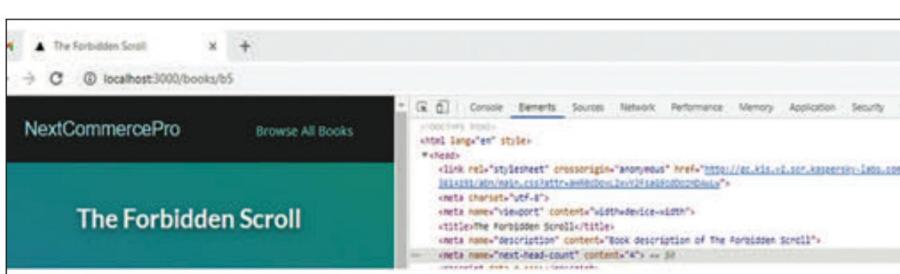


Figura 5.2: Página de libro dinámica

En este ejemplo, pasamos el objeto de publicación como propiedad al componente BookPage.

Usamos los datos de la publicación para configurar dinámicamente el título, la descripción y las etiquetas

Open Graph (OG). Las etiquetas og:title, og:description y og:image son esenciales para que las plataformas de redes sociales muestren el título, la descripción y la miniatura correctos al compartir la página.

Con el componente Head, puedes configurar fácilmente metadatos para cada página de tu aplicación Next.js, lo que mejora el SEO y la representación visual de tus páginas en redes sociales. Recuerda incluir metadatos relevantes, como títulos, descripciones y etiquetas OG, para asegurar que tus páginas estén optimizadas para motores de búsqueda y para compartir en redes sociales.

Integración de la publicación de archivos estadísticos en Next.js

Al implementar el servicio de archivos estáticos en Next.js, los diferentes tipos de archivos estáticos desempeñan un papel crucial en el desarrollo web, incluyendo imágenes, hojas de estilo CSS y archivos avaScript. En esta sección, exploraremos cómo servir archivos estáticos eficientemente en su aplicación Next.js. Abordaremos diversas estrategias y prácticas recomendadas para garantizar un rendimiento óptimo y una integración fluida de los recursos estáticos en sus proyectos Next.js.

- Sirviendo imágenes

Las imágenes son parte integral del desarrollo web moderno. En Next.js, puedes mostrar imágenes siguiendo estos pasos:

Cree una carpeta llamada pública en el directorio raíz de su Next.js proyecto.

Coloque su archivo de imagen, digamos my-image.jpg, dentro de la carpeta pública.

En su componente Next.js, use el componente <Image> de Next.js para mostrar la imagen.

He aquí un ejemplo:

```
importar imagen de 'next/image';
constante MiComponente = () => {
    devolver (
        <div>
            <Imagen src="/mi-imagen.jpg" alt="Mi imagen" ancho={500} alto={300}>
        />
    </div>
);
};

exportar predeterminado MyComponent;
```

El componente `<Image>` optimiza y sirve automáticamente la imagen, brindando beneficios como carga diferida y renderizado responsivo de la imagen.

- Sirviendo hojas de estilo CSS

Al implementar el servicio de archivos estáticos en Next.js, las hojas de estilo CSS son esenciales. Permiten estilizar las páginas web y hacerlas visualmente atractivas. En esta sección, profundizaremos en cómo servir eficientemente hojas de estilo CSS en su aplicación Next.js.

Para empezar, supongamos que tiene un archivo CSS llamado "styles.css" que contiene todos los estilos de su aplicación Next.js. Aquí tiene un ejemplo de cómo puede usar este archivo CSS con Next.js.

Cree una carpeta llamada pública en el directorio raíz de su Next.

Proyecto js. Esta carpeta contendrá todos sus archivos estáticos, incluidas las hojas de estilo CSS.

Mueva el archivo styles.css a la carpeta pública .

En su componente o archivo de diseño Next.js, importe el archivo CSS usando el

Etiqueta `<link>` dentro de la sección `<head>` .

He aquí un ejemplo:

```
importar Head desde 'next/head';
constante MiComponente = () => {
    devolver (
        <div>
            <Cabeza>
                <link rel="hoja de estilo" href="/styles.css" />
            </Cabeza>
            {/* El contenido de su componente */}
        </div>
    );
};

exportar predeterminado MyComponent;
```

Con la etiqueta `<link>` , especificamos el atributo `rel` como "stylesheet" para indicar que se trata de un archivo de hoja de estilos. El atributo `href` indica la ubicación del archivo CSS en relación con la raíz de la aplicación Next.js. En este caso, es "/styles.css" porque el archivo se encuentra en la carpeta pública.

Si sigue estos pasos, su aplicación Next.js servirá eficientemente la hoja de estilos CSS. Las reglas CSS definidas en “styles.css” se aplicarán a los componentes y elementos correspondientes en su aplicación.

Es importante tener en cuenta que Next.js maneja automáticamente el almacenamiento en caché y la optimización de archivos estáticos. Cuando realiza cambios en el archivo CSS, Next.js generará un nuevo nombre de archivo único para la versión actualizada, lo que garantiza que los clientes reciban los últimos estilos sin depender de la invalidación manual del caché.

Al adoptar este enfoque, puede integrar sin problemas hojas de estilo CSS en sus proyectos Next.js y garantizar un rendimiento óptimo al servir eficientemente archivos estáticos.

- Entrega de archivos JavaScript

Cuando se trata de servir archivos avaScript en Next.js, puede seguir un proceso sencillo para garantizar la entrega eficiente y la integración de estos archivos en su aplicación. Aquí hay una explicación detallada sobre cómo servir archivos avaScript en Next.js.

1. Crea un directorio raíz: Comience creando una carpeta llamada “public” en el directorio raíz de su proyecto Next.js si aún no existe. Esta carpeta contendrá todos sus archivos estáticos, incluidos los archivos avaScript.
2. Guarde sus archivos avaScript en la carpeta “public”. Por ejemplo, supongamos que tiene un archivo avaScript llamado script.js. Cópielo o muévalo a la carpeta “public”. La estructura del archivo debería verse como se muestra en la Figura 5.3:

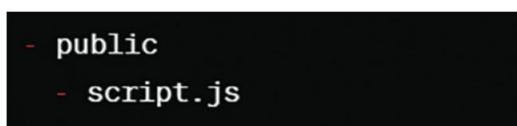


Figura 5.3: Estructura de la carpeta pública

3. Incluir archivos JavaScript en el HTML: Para incluir el archivo JavaScript en el archivo, puedes usar la etiqueta script. Next.js procesa automáticamente los archivos ubicados en la carpeta pública como archivos estáticos, y puedes referenciarlos por su ruta relativa desde la raíz de Next.js. En su archivo o componente Next.js, incluya el siguiente fragmento de código para importar el archivo avaScript <script src="/script.js"></script>

Aquí, el atributo src de la etiqueta script apunta a la ubicación del archivo avaScript en relación con la raíz de su aplicación Next.js. En

En este caso, el archivo se llama “script.js” y se encuentra directamente en la carpeta “public” .

4. tiling aarit ntonalit Una vez que el archivo avaScript está incluido en su componente HTML o Next.js, puede acceder y utilizar la funcionalidad que ofrece. El código JavaScript dentro de “script.js” puede contener diversas funciones, detectores de eventos u otra lógica que desee incorporar a su aplicación.

Por ejemplo, si “script.js” contiene una función llamada “myFunction”, puedes llamarla desde tu componente HTML o Next.js usando la siguiente sintaxis:

```
<script src="/script.js"></script>  
<guión>  
    myunction(); // todas las funciones definidas en script.s  
</script>
```

Siguiendo estos pasos, Next.js servirá el archivo avaScript como un recurso estático, lo que permitirá que se muestre eficientemente en el navegador del cliente. A continuación, podrá utilizar la funcionalidad de JavaScript en su aplicación según sea necesario.

Es importante tener en cuenta que Next.js gestiona automáticamente el almacenamiento en caché y la optimización de archivos estáticos, incluidos los archivos avaScript. Si modifica el archivo avaScript, Next.js generará un nuevo nombre de archivo único para la versión actualizada, lo que garantiza que los clientes reciban el código más reciente sin tener que recurrir a la invalidación manual de la caché.

Al servir archivos avaScript de esta manera, puede integrar sin problemas funcionalidad dinámica e interactividad en su aplicación Next.js mientras garantiza un rendimiento óptimo y una entrega eficiente de datos estáticos activos.

Introducción a la optimización de imágenes o al uso de Next.js

Las imágenes son parte integral de cualquier aplicación web moderna, y optimizar su rendimiento es esencial. En esta sección, profundizaremos en el componente Imagen de Next.js y sus funciones para optimizar imágenes en sus aplicaciones Next.js.

El componente de imagen de Next.js proporciona técnicas de optimización de imágenes integradas, como cambio de tamaño automático, carga diferida y entrega de imágenes responsivas, para garantizar una representación eficiente y tiempos de carga de página reducidos.

Obtendrá una comprensión integral de cómo aprovechar este componente para ofrecer imágenes eficientes y visualmente atractivas en sus proyectos Next.js. Para utilizar el componente de imagen de Next.js, debe importarlo desde el módulo 'next/image'.

Consideremos un ejemplo de uso del componente Image de Next.js para mostrar una imagen optimizada:

```
importar imagen de 'next/image';
función Página de inicio() {
    devolver (
        <div>
            <Imagen src="/images/my-image.jpg" alt="Mi imagen" width={800}
altura={600} />
            {/* Resto del contenido de la página */}
        </div>
    );
}
```

exportar página de inicio predeterminada;

En el fragmento de código anterior, importamos el componente Image desde 'next/módulo 'imagen' y use el componente Imagen para mostrar la imagen ubicada en '/ imágenes/mi-imagen.jpg'. También proporcionamos el texto alternativo para fines de accesibilidad. Además, especificamos el ancho y alto deseados de la imagen utilizando los accesorios de ancho y alto.

El componente Image de Next.js optimiza automáticamente la imagen según las dimensiones proporcionadas. Genera múltiples versiones optimizadas de la imagen durante la compilación, cada una adaptada a diferentes tamaños de pantalla y dispositivos. La imagen adecuada se entrega al cliente según las características de su ventana gráfica y dispositivo, lo que mejora el rendimiento y reduce el consumo de ancho de banda (Figura 5.4).

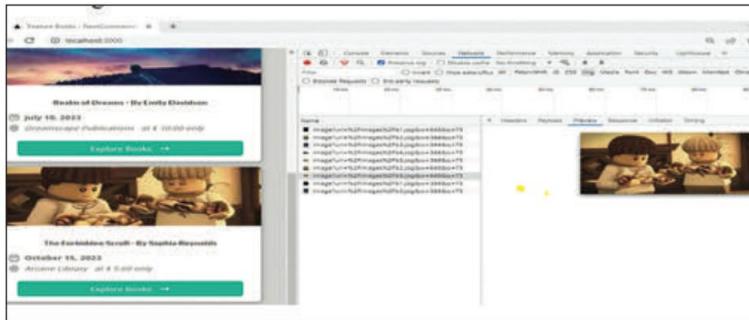


Figura 5.4: Página de lista de productos o página de lista de libros

El componente de imagen de Next.js admite varias otras propiedades que puedes usar para optimizar aún más la entrega de imágenes:

- `laot`

La propiedad de diseño determina cómo debe distribuirse la imagen en la página. Puede configurarse como "responsive" (predeterminado), "e" o "intrínseca".

El diseño "responsive" dimensiona automáticamente la imagen en función de las dimensiones del contenedor, mientras que "e" e "intrínseco" imponen dimensiones específicas.

```
importar imagen de 'next/image';
función Página de inicio() {
    devolver (
        <div>
            <Image
                src="/images/mi-imagen.jpg"
                alt="Mi imagen"
                diseño="responsivo"
                ancho={800}
                altura={600}
            />
            {/* Resto del contenido de la página */}
        </div>
    );
}
exportar página de inicio predeterminada;
```

En el ejemplo anterior, configuramos la propiedad de diseño como "responsive". Esto permite que la imagen ajuste su tamaño dinámicamente según las dimensiones del contenedor, garantizando un diseño responsivo.

- **objeto**

La propiedad objectFit determina cómo se debe escalar y posicionar la imagen dentro de su contenedor. Puede configurarse como "" (predeterminado), "contain", "cover", "none" o "scale-down".

En el siguiente ejemplo, configuraremos la propiedad objectFit como "cover". Esto garantiza que la imagen cubra todo el contenedor, manteniendo su relación de aspecto.

```
importar imagen de 'next/image';
función Página de inicio() {
    devolver (
        <div>
            <Image
                src="/images/mi-imagen.jpg"
                alt="Mi imagen"
                diseño=rellenar
                objectFit="cubierta"
            />
            {/* Resto del contenido de la página */}
        </div>
    );
}
exportar página de inicio predeterminada;
```

- **objetoposición**

La propiedad objectPosition especifica la posición horizontal y vertical de la imagen dentro de su contenedor. Acepta valores como "arriba", "abajo", "izquierda", "derecha", "centro" o una combinación de estos.

```
importar imagen de 'next/image';
función Página de inicio() {
    devolver (
```

```
<div>
  <Image
    src="/images/mi-imagen.jpg"
    alt="Mi imagen"
    diseño=fijo
    ancho={500}
    altura={300}
    objectFit="contiene"
    objectPosition="arriba a la derecha"
  />
  {/* Resto del contenido de la página */}
</div>
);
}
```

exportar página de inicio predeterminada;

En este ejemplo, configuramos la propiedad objectPosition en "arriba a la derecha". Esto posiciona la imagen en la esquina superior derecha del contenedor. •

cargando

La propiedad de carga controla la carga diferida de la imagen. Puede configurarse como "eager" (predeterminado) para cargar la imagen inmediatamente, o como "lazy" para posponer la carga hasta que la imagen entre en la ventana gráfica.

importar imagen de 'next/image';

```
función Página de inicio() {
  devolver (
    <div>
      <Image
        src="/images/mi-imagen.jpg"
        alt="Mi imagen"
        cargando="perezoso"
      />
    
```

```
    /* Resto del contenido de la página */
  </div>
);

}

exportar página de inicio predeterminada;
```

En este ejemplo, establecemos la propiedad de carga en “perezosa”, lo que permite que la imagen se cargue solo cuando ingresa a la ventana gráfica y mejora el rendimiento al reducir el tiempo de carga inicial de la página.

- propiedad prioritaria

La propiedad de prioridad del componente Imagen de Next.js permite priorizar la carga de imágenes, garantizando una mejor experiencia de usuario para contenido crítico. Al establecer una mayor prioridad para ciertas imágenes, se asegura de que se carguen primero, reduciendo así el tiempo de carga percibido de elementos visuales importantes en la página.

A continuación se muestra un ejemplo de cómo utilizar la propiedad de prioridad:

```
importar imagen de 'next/image';
función Página de inicio() {
  devolver (
    <div>
      <Image
        src="/images/mi-imagen.jpg"
        alt="Mi imagen"
        prioridad
      />
    /* Resto del contenido de la página */
  </div>
);
}

exportar página de inicio predeterminada;
```

En el fragmento de código anterior, establecimos la propiedad de prioridad en verdadero para el componente de imagen. Esto indica que se debe asignar una prioridad a la imagen.

mayor prioridad durante la carga.

Al establecer la propiedad de prioridad, Next.js priorizará la carga de la imagen, obteniéndola al principio del proceso de carga de la página. Esto garantiza que la imagen esté disponible lo antes posible, mejorando la experiencia del usuario al reducir el tiempo de carga percibido para elementos visuales importantes.

Es importante tener en cuenta que la propiedad de prioridad debe usarse de forma selectiva para imágenes críticas que son esenciales para la representación inicial de su página. Usarla para cada imagen de la página puede no brindar beneficios significativos y podría afectar potencialmente el rendimiento general de su aplicación.

En resumen, la propiedad de prioridad en el componente Image de Next.js le permite controlar la prioridad de carga de las imágenes, lo que garantiza una mejor experiencia de usuario al cargar primero el contenido crítico.

Al centrarse en los tres temas detallados mencionados anteriormente y brindar una descripción general de los temas restantes, este capítulo tiene como objetivo brindarle el conocimiento necesario para aprovechar Next.js de manera efectiva y optimizar sus aplicaciones para un rendimiento superior.

Entendiendo la arquitectura de Next.js y cómo funciona

Proporcionaremos una descripción general de alto nivel de la arquitectura de Next.js, sus componentes clave y cómo interactúan para ofrecer potentes aplicaciones renderizadas del lado del servidor.

Next.js es un framework React que ofrece una solución integral para crear aplicaciones web modernas, priorizando el rendimiento y la experiencia del desarrollador. Combina las ventajas del renderizado del lado del servidor (SSR) y del lado del cliente (CSR) para ofrecer experiencias de usuario rápidas e interactivas.

Next.js utiliza un enfoque de renderizado híbrido, lo que permite elegir entre SSR y CSR según las necesidades de la aplicación. Exploraremos la arquitectura y su funcionamiento con más detalle:

- Representación del lado del servidor (SSR)

En la renderización del lado del servidor, este genera el contenido HTML de cada página y lo envía al cliente. Esto significa que la carga inicial de la página contiene el HTML completamente renderizado, incluyendo el contenido obtenido de las API o bases de datos. Este enfoque ofrece varias ventajas, como las siguientes:

SEO mejorado: los motores de búsqueda pueden rastrear e indexar fácilmente el HTML completamente renderizado, lo que genera una mejor optimización del motor de búsqueda.

Carga de página inicial más rápida: dado que el servidor envía el HTML pre-renderizado al cliente, el usuario puede ver el contenido más rápido, incluso en conexiones más lentas.

mejor error o contenido estático: el contenido estático se genera una vez en el servidor y se puede almacenar en caché para solicitudes posteriores, lo que reduce la carga en el servidor.

Para habilitar SSR en Next.js, necesitas crear un tipo especial de archivo llamado página en el directorio de páginas. Cada archivo de página exporta un componente de React que representa el contenido de esa página. Durante el proceso de compilación, Next...
js pre-renderiza estas páginas y genera el correspondiente

archivos.

- Representación del lado del cliente (CSR)

En la renderización del lado del cliente, el HTML inicial enviado por el servidor contiene un esqueleto mínimo de la página, y el JavaScript del lado del cliente se encarga de obtener los datos y renderizar el contenido dinámicamente. Este enfoque ofrece las siguientes ventajas:

Experiencia de interacción con el cliente Con CSR, puedes crear aplicaciones altamente interactivas que obtienen y muestran datos en el lado del cliente sin necesidad de recargar páginas completas.

Actualizaciones en tiempo real Al obtener datos de forma asíncrona, puedes actualizar la interfaz de usuario en tiempo real sin actualizar toda la página.

Next.js admite CSR mediante su potente sistema de enrutamiento del lado del cliente. Cuando un usuario navega a otra página dentro de una aplicación Next.js, solo se obtienen el JavaScript y los datos necesarios, y la interfaz de usuario se actualiza sin recargar la página completa.

Arquitectura de Next.js: Next.js utiliza una arquitectura sin servidor, lo que permite implementar la aplicación en plataformas sin servidor como Vercel, AWS Lambda o Netlify. Esta arquitectura escala automáticamente según el tráfico entrante, lo que garantiza un rendimiento óptimo y reduce los costos de infraestructura.

Next.js también ofrece varias funciones de optimización del rendimiento listas para usar, como la división automática de código, la generación de sitios estáticos (SSG) y la regeneración estática incremental (ISR). Estas funciones permiten crear aplicaciones web optimizadas y de carga rápida.

Ejemplo de código:

A continuación se muestra un ejemplo de un componente de página básico de Next.js que utiliza SSR:

```
// páginas/index.js
importar React desde 'react';

función Página de inicio({ datos }) {
    devolver (
        <div>
            ¡Bienvenido a Next.js!
            <p>{datos}</p>
        </div>
    );
}

exportar función asíncrona getServerSideProps() {
    // Obtener datos de una API o base de datos
    const res = await fetch('https://api.example.com/data');
    const datos = await res.json();
    devolver {
        accesorios: {
            datos,
        },
    };
}

exportar página de inicio predeterminada;
```

En este código, tenemos un componente de página simple de Next.js llamado `HomePage`. Recibe datos como una propiedad y los renderiza en la página. La función `getServerSideProps` es una función especial de Next.js que se ejecuta en el servidor y obtiene datos de una API o base de datos. Los datos se obtienen y se pasan como una propiedad al componente `HomePage`.

Cuando un usuario solicita la página de inicio, Next.js ejecuta la función `getServerSideProps` en el servidor y obtiene los datos. Estos se pasan como propiedades al componente `HomePage`, que renderiza el contenido de la página.

Este es un ejemplo de SSR en Next.js. El servidor genera el HTML completamente renderizado con los datos obtenidos y lo envía al cliente, lo que resulta en una carga inicial de la página más rápida y un SEO mejorado.

Además de SSR, Next.js también admite CSR para contenido dinámico. Puedes usar el gancho `useEffect` u otros mecanismos para obtener datos en el lado del cliente y actualizar la interfaz de usuario sin recargar la página completa.

Comprender la arquitectura y las capacidades de SSR y CSR en Next.js le permite aprovechar el método de renderizado adecuado según los requisitos de su aplicación, equilibrando el rendimiento, la interactividad y las consideraciones de SEO.

Configurando Next.js o optimal Erormane

Analizaremos varias opciones de configuración disponibles en Next.js para optimizar el rendimiento de su aplicación. Esta descripción general le servirá como punto de partida para optimizar sus proyectos de Next.js.

Para lograr un rendimiento óptimo en sus aplicaciones Next.js, existen varias opciones de configuración y prácticas recomendadas que puede seguir. Esto incluye usar una compilación lista para producción, optimizar imágenes e implementar el almacenamiento en caché del lado del servidor. Exploraremos cada uno de estos aspectos en detalle aquí:

- cantar una rotionrea il

De forma predeterminada, cuando se ejecuta el servidor de desarrollo Next.js, se utiliza una compilación de desarrollo que incluye características como reemplazo de módulos en caliente y herramientas de depuración. Sin embargo, para un rendimiento óptimo en producción, se recomienda utilizar una compilación lista para producción.

Para crear una compilación de producción, puede utilizar el siguiente comando:

próxima construcción

Este comando genera una compilación optimizada y minimizada de su aplicación Next.js. A continuación, puede iniciar el servidor de producción con:

próximo comienzo

Al iniciar la compilación de producción se eliminan el código y las optimizaciones innecesarias específicas del desarrollo, lo que genera tiempos de carga más rápidos y un rendimiento mejorado.

Implementación del almacenamiento en caché del lado del servidor

Discutiremos brevemente los conceptos de renderizado del lado del servidor y pre-renderizado en Next.js, destacando sus beneficios y cómo implementarlos en sus aplicaciones.

El almacenamiento en caché del lado del servidor puede mejorar significativamente el rendimiento de su Next.js, reduciendo la carga del servidor y mejorando los tiempos de respuesta. Next.js ofrece compatibilidad integrada con el almacenamiento en caché del lado del servidor mediante varios mecanismos:

- Usar `getServerSideProps` con caché: La función `getServerSideProps` de Next.js se puede configurar para almacenar en caché la página renderizada por el servidor durante un tiempo específico. Al configurar la propiedad `revalidate`, se puede controlar la frecuencia con la que se debe actualizar la caché. Esto es útil para datos que no cambian con frecuencia.
- Implementar almacenamiento en caché externo. Las aplicaciones Next.js pueden beneficiarse del uso de mecanismos de almacenamiento en caché externo, como el almacenamiento en caché de CDN (Red de Entrega de Contenido) o el almacenamiento en caché proxy. Estos mecanismos almacenan el HTML generado o los recursos estáticos en ubicaciones periféricas, más cercanas al usuario, lo que resulta en tiempos de respuesta más rápidos.
- Usar plugins de caché: Next.js cuenta con un ecosistema dinámico de plugins que ofrecen soluciones de caché adaptadas a diferentes casos de uso. Puedes explorar los plugins de caché disponibles para optimizar aún más la estrategia de caché de tu aplicación.

Al implementar el almacenamiento en caché del lado del servidor, puede reducir la carga en su servidor, mejorar los tiempos de respuesta y brindar una experiencia de usuario más rápida.

Ejemplo de Código

A continuación se muestra un ejemplo del uso de la función `getServerSideProps` con almacenamiento en caché en Next.js:

```
importar React desde 'react';
función Página de inicio({ datos }) {
    devolver (
        <div>
            ¡Bienvenido a Next.js!
            <p>{datos}</p>
        </div>
    )
}
```

```
    );
}

exportación función asíncrona getServerSideProps() {

  const cacheSeconds = 60; // Almacenar en caché durante 60 segundos

  // Obtener datos de una API o base de datos

  const res = await fetch('https://api.example.com/data');

  const datos = await res.json();

  devolver {

    accesorios: {

      datos,

    },
    revalidar: cacheSeconds,
  };
}
```

En el ejemplo de código anterior, tenemos un componente básico de página de Next.js llamado `HomePage`. Recibe `data` como propiedad y lo renderiza en la página. La función `getServerSideProps` es una función especial de Next.js que se ejecuta en el servidor y obtiene datos de una API o base de datos.

Dentro de la función `getServerSideProps` , definimos una variable `cacheSeconds` para establecer la duración durante la cual la página debe almacenarse en caché (en este caso, 60 segundos). Los datos obtenidos se devuelven en el objeto `props` junto con la propiedad `revalidate` establecida en el valor de `cacheSeconds` .

Al configurar la propiedad `revalidate` , Next.js almacenará en caché la página renderizada por el servidor y actualizará el caché después de que haya transcurrido la duración especificada. Esto ayuda a reducir la carga en el servidor al servir contenido en caché para solicitudes posteriores dentro de la duración del caché.

Al implementar estrategias de almacenamiento en caché como esta, puede mejorar el rendimiento de su aplicación Next.js al reducir la necesidad de obtención frecuente de datos y renderizado del lado del servidor.

Recuerde ajustar la duración del almacenamiento en caché (`cacheSeconds`) según su caso de uso específico y la frecuencia de actualización de datos.

En resumen, configurar Next.js para un rendimiento óptimo implica utilizar una compilación lista para producción, optimizar imágenes e implementar almacenamiento en caché del lado del servidor. Estas prácticas ayudan a garantizar que su aplicación ofrezca una experiencia de usuario rápida y optimizada, mejorando el rendimiento y la escalabilidad.

División de código e importaciones dinámicas

Esta sección ofrecerá una breve introducción a la división de código y las importaciones dinámicas, y cómo ayudan a reducir el tamaño inicial del paquete y a mejorar el rendimiento en Next.js. Comprenderá cómo estas técnicas pueden mejorar el rendimiento al cargar solo el código necesario para una página o función específica.

La división de código y las importaciones dinámicas son técnicas que se utilizan para dividir un paquete de JavaScript en fragmentos más pequeños y manejables. Al dividir el código y cargarlo bajo demanda, se puede reducir el tamaño inicial del paquete y mejorar el rendimiento, especialmente en aplicaciones de gran tamaño. El capítulo 6, «Comprendiendo el enrutamiento en Next.js», profundizará en estos conceptos y proporcionará ejemplos prácticos para demostrar su aplicación práctica.

La división de código implica dividir su base de código en piezas más pequeñas, o fragmentos, en función de puntos de entrada específicos o dependencias de código. Esto le permite cargar solo el código requerido cuando es necesario, en lugar de cargar todo por adelantado. El resultado es un tamaño de paquete inicial más pequeño, lo que puede mejorar significativamente el tiempo que tarda un usuario en ver e interactuar con su aplicación.

Las importaciones dinámicas son un mecanismo clave para implementar la división de código. Permiten importar módulos de forma asíncrona, en tiempo de ejecución, en lugar de hacerlo estáticamente durante la compilación. Al importar módulos dinámicamente solo cuando son necesarios, se optimiza la carga de la aplicación y se reducen las solicitudes de red innecesarias.

A continuación se muestra un ejemplo de cómo se pueden utilizar la división de código y las importaciones dinámicas en una aplicación Next.js:

```
importar eact, { useState, useEffect } de react;
función Página de inicio() {
    const [datos, setData] = useState(nulo);
    usarEfecto(() => {
        importar('./api').luego((módulo) => {
            módulo.getData().then((respuesta) => {
                setData(respuesta);
            });
        });
    }, []);
}
```

```

devolver (
  <div>
    ¡Bienvenido a Next.js!
    <p>{datos}</p>
  </div>
);

}

exportar página de inicio predeterminada;

```

En el fragmento de código anterior, el componente HomePage utiliza importaciones dinámicas para cargar un módulo externo llamado api de forma asíncrona. El módulo api contiene una función llamada getData() que obtiene datos. Al montar el componente, el gancho useEffect activa la importación dinámica y, una vez cargado el módulo, obtiene los datos mediante la función getData() y actualiza su estado.

Al importar dinámicamente el módulo API, se garantiza que solo se cargue al renderizar el componente HomePage, lo que reduce el tamaño inicial del paquete. Esto es especialmente beneficioso si el módulo API solo se usa en una parte específica de la aplicación y no se necesita en todas las páginas.

Al aplicar estratégicamente la división de código y las importaciones dinámicas en toda la aplicación, se puede optimizar la carga y el rendimiento de la aplicación Next.js. Esta técnica ayuda a reducir el tamaño inicial del paquete, mejorar el tiempo de interacción y ofrecer una experiencia de usuario más fluida.

Caing an imroing ata eting

El almacenamiento en caché es crucial para mejorar el rendimiento de la obtención de datos en las aplicaciones Next.js. Al almacenar datos en caché, se puede reducir el número de solicitudes a API o bases de datos externas, lo que se traduce en tiempos de respuesta más rápidos y una mejor experiencia de usuario.

Next.js proporciona varios mecanismos de almacenamiento en caché que puedes aprovechar:

- Almacenamiento en caché del cliente: Next.js admite el almacenamiento en caché del lado del cliente mediante bibliotecas como SWR (Stale-While-Revalidate) o React Query. Estas bibliotecas gestionan el almacenamiento en caché, la revalidación y la sincronización de datos entre componentes, lo que permite almacenar en caché y reutilizar datos en toda la aplicación.

Ejemplo utilizando SWR:

```

importar useSWR desde 'swr';
función serrofile() {

    const { datos, error } = useSWR('/api/usuario', fetcher);
    si (error) devuelve <div>Error al cargar datos del usuario</div>;
    si (!data) devuelve <div>Cargando...</div>;
    devolver (
        <div>
            ¡Bienvenido, {data.name}!
            <p>Correo electrónico: {data.email}</p>
        </div>
    );
}

```

Next.js ofrece almacenamiento en caché integrado del lado del servidor mediante las funciones `getServerSideProps` y `getStaticProps`. Al especificar una configuración de almacenamiento en caché en estas funciones, puede almacenar en caché las páginas renderizadas por el servidor o los datos estáticos durante un tiempo específico. Esto reduce la carga del servidor y mejora los tiempos de respuesta.

xamle sing getererieros

```

exportar función asíncrona getServerSideProps(contexto) {
    datos constantes = esperar fetchUserFromAPI();
    devolver {
        accesorios: {
            datos,
        },
        revalidar: 60, // Almacenar en caché durante 60 segundos
    };
}

```

Almacenamiento externo. También puede utilizar mecanismos de almacenamiento en caché externo, como el almacenamiento en caché de CDN (Red de Entrega de Contenido) o el almacenamiento en caché de proxy. Estos mecanismos almacenan en caché las respuestas de la API o los recursos estáticos en ubicaciones periféricas, más cercanas al usuario, lo que agiliza las solicitudes posteriores.

Al implementar estrategias de almacenamiento en caché de manera adecuada, puede minimizar la necesidad

para la obtención frecuente de datos y mejorar el rendimiento general de su aplicación Next.js.

Análisis y reducción del tamaño del paquete

Exploraremos brevemente técnicas para analizar y reducir el tamaño del paquete de sus aplicaciones Next.js. Esta descripción general le presentará herramientas y enfoques que pueden ayudarle a optimizar el tamaño del paquete de su proyecto para un mejor rendimiento.

El tamaño del paquete de tu aplicación Next.js afecta el tiempo de carga inicial y el rendimiento. Analizar y reducir el tamaño del paquete es crucial para garantizar una experiencia de usuario rápida y fluida. Aquí tienes algunas técnicas para lograrlo:

- **División de código:** Next.js admite la división automática de código, donde cada página se agrupa por separado. Esto garantiza que solo se cargue el código necesario para cada página, reduciendo el tamaño del paquete. También puede configurar manualmente la división de código mediante importaciones dinámicas, como se explicó en el tema anterior.
- **Tree Shaking:** El Tree Shaking es un proceso en el que se elimina el código no utilizado del paquete durante el proceso de compilación. Next.js utiliza herramientas como webpack, que realiza automáticamente el Tree Shaking para su aplicación. Asegúrese de escribir código modular y utilizar módulos ES para optimizar el tree shaker.
- **Minificación:** La minimización es el proceso de eliminar del código caracteres innecesarios, como espacios en blanco y comentarios. Next.js realiza la minimización por defecto en la compilación de producción, lo que resulta en paquetes más pequeños.

Optimización de dependencias: Analice y optimice las dependencias utilizadas en su aplicación.

Considera usar bibliotecas alternativas más pequeñas u optimizar la configuración de las dependencias existentes para reducir el tamaño del paquete.

Estrategias de implementación y tasas de éxito . Implementar una aplicación Next.js de forma eficiente es fundamental para garantizar su disponibilidad, escalabilidad y rendimiento. A continuación, se presentan algunas estrategias de implementación y prácticas recomendadas:

Las aplicaciones Next.js son ideales para implementaciones sin servidor, donde la aplicación se implementa en plataformas sin servidor como Vercel, AWS Lambda o Netlify. Las implementaciones sin servidor ofrecen escalado automático, menores costos de infraestructura y flujos de trabajo de implementación simplificados .

Integración continua y desarrollo CCD Implementar una CI/CD para automatizar el proceso de implementación. Herramientas como GitHub Actions o Jenkins permiten compilar e implementar la aplicación Next.js cada vez que se envían cambios al repositorio.

Monitoreo de errores. Monitorea el rendimiento de tu aplicación implementada con herramientas como Lighthouse, New Relic o Google Analytics. Esto ayuda a identificar cuellos de botella en el rendimiento y a optimizar tu aplicación para una mejor experiencia de usuario.

Consideraciones de seguridad Implemente las mejores prácticas de seguridad como HTTPS, integraciones de API seguras y control de acceso adecuado para proteger su aplicación Next.js y sus datos.

Configura tu entorno de implementación para escalar horizontalmente mediante平衡adores de carga y mecanismos de escalado automático. Esto garantiza que tu aplicación Next.js pueda gestionar el aumento de tráfico sin reducir el rendimiento .

Estrategias de implementación y mejores prácticas

Ofreceremos una descripción general de diversas estrategias de implementación y prácticas recomendadas para aplicaciones Next.js. Si bien no es exhaustiva, esta sección le brindará información sobre las diferentes opciones para implementar sus proyectos Next.js en entornos de producción.

Implementar una aplicación Next.js de forma eficiente es fundamental para garantizar su disponibilidad, escalabilidad y rendimiento. Aquí tienes algunas estrategias de implementación y prácticas recomendadas:

- elementos sin fin

Las aplicaciones Next.js son ideales para implementaciones sin servidor, donde la aplicación se implementa en plataformas sin servidor como Vercel, AWS Lambda o Netlify. Las implementaciones sin servidor ofrecen escalado automático, menores costos de infraestructura y flujos de trabajo de implementación simplificados.

Ejemplo usando Vercel:

Instalar Vercel CLI:

```
npm install -g vercel
```

Implemente su aplicación Next.js:

Implementación de Vercel

- Integración y despliegue continuo (CI/CD):

Implemente una canalización de CI/CD para automatizar el proceso de implementación. Herramientas como GitHub Actions o Jenkins permiten compilar e implementar su aplicación Next.js cada vez que se envían cambios al repositorio.

- Ejemplo usando acciones de GitHub:

Crea un archivo de flujo de trabajo (por ejemplo, .gitworfloowsmain.ml) en tu repositorio:

nombre: Implementar la aplicación Next.js

en:

empujar:

sucursales:

- principal

trabajos:

desplegar:

se ejecuta en: ubuntu-latest

pasos:

- nombre: Código de pago

usos: acciones/checkout@v2

- nombre: Instalar dependencias

ejecutar: npm install

- nombre: Construir e implementar

usos: vercel/acciones@v21

con:

argumentos: "implementar"

- Monitoreo de errores Monitoree el rendimiento de su aplicación implementada utilizando herramientas como Lighthouse, New Relic o Google Analytics. Esto ayuda a identificar cuellos de botella en el rendimiento y optimizar su aplicación para una mejor experiencia de usuario.
- consideraciones de seguridad Implementar las mejores prácticas de seguridad, como

- utilizando HTTPS, protegiendo las integraciones de API e implementando un control de acceso adecuado para proteger su aplicación Next.js y sus datos.
- Configure su entorno de implementación para escalar horizontalmente mediante el uso de balanceadores de carga y mecanismos de escalamiento automático. Esto garantiza que su aplicación Next.js pueda manejar un mayor tráfico sin degradación del rendimiento .

Ejemplo de equilibrio de carga mediante AWS Elastic Load Balancer:

1. Cree un balanceador de carga elástico en AWS.
- Configurar el balanceador de carga para distribuir el tráfico entre varias instancias. de su aplicación Next.js.
3. Configure el escalado automático para agregar o eliminar instancias automáticamente según tráfico.

Siguiendo estas estrategias de implementación y prácticas recomendadas, puede garantizar que su aplicación Next.js se implemente de forma eficiente, tenga un rendimiento óptimo y sea segura. Estas prácticas permiten una escalabilidad fluida y brindan una experiencia de usuario fluida a los usuarios de su aplicación.

Monitoreo de errores

Abordaremos la importancia de monitorizar y optimizar el rendimiento de las aplicaciones Next.js. Este resumen destacará las métricas clave de rendimiento y le presentará herramientas y prácticas para monitorizar y mejorar el rendimiento general de sus proyectos Next.js.

- Lighthouse es una herramienta de código abierto de Google que audita el rendimiento, la accesibilidad, el SEO y las mejores prácticas de las páginas web. Proporciona informes detallados y sugerencias para mejorar el rendimiento. Puede ejecutar auditorías de Lighthouse en su aplicación Next.js para identificar cuellos de botella en el rendimiento y áreas de mejora.
- Las herramientas de monitoreo RUM como Google Analytics o New Relic capturan métricas de rendimiento e interacciones de los usuarios en tiempo real. Proporcionan información sobre el rendimiento de su aplicación en condiciones reales y le ayudan a identificar problemas de rendimiento que experimentan los usuarios reales.

Ejemplo usando Lighthouse:

```
# Instalar Lighthouse CLI
```

```
npm install -g faro
```

```
# Ejecute la auditoría de Lighthouse en su aplicación Next.js
```

faro http://url-de-tu-aplicación-nextjs

- optimización de errores

Algunas características de la optimización del rendimiento son las siguientes:

Como se mencionó anteriormente, Next.js admite la división automática de código, lo que permite cargar solo el código necesario para cada página. Al analizar las dependencias de la aplicación y optimizar la división de código, se puede reducir el tamaño del paquete y mejorar los tiempos de carga iniciales.

Next.js proporciona el componente Imagen, que optimiza y entrega imágenes de forma eficiente. Al aprovecharlo, puedes reducir el tamaño de los archivos de imagen, cargar imágenes de forma diferida y mostrar imágenes adaptables según el tamaño de la pantalla del dispositivo .

Next.js permite almacenar en caché páginas renderizadas por el servidor o datos estáticos mediante funciones como getServerSideProps o getStaticProps. Al implementar estrategias de almacenamiento en caché adecuadas, se puede reducir la carga del servidor y mejorar los tiempos de respuesta .

Obtención de errores. Define presupuestos de rendimiento para tu aplicación Next.js, especificando umbrales para métricas como el tamaño del paquete, las solicitudes de red y el tiempo de carga de la página. Supervisa estas métricas periódicamente y asegúrate de que tu aplicación se mantenga dentro de los umbrales de rendimiento definidos.

Ejemplo de optimización de la carga de imágenes con el componente Image de Next.js:

```
importar imagen de 'next/image';
función MiComponente() {
    devolver (
        <div>
            Bienvenido a Mi Componente
            <Imagen src="/ruta/a/imagen.jpg" alt="Descripción de la imagen"
            ancho={500} alto={300} />
        </div>
    );
}
```

En este ejemplo, el componente Imagen de Next.js se utiliza para cargar y optimizar una imagen. Se proporciona la ruta a la imagen, el texto alternativo para accesibilidad y el ancho y alto deseados. Next.js optimiza automáticamente la imagen generando múltiples tamaños y formatos, y ofrece la versión más adecuada según el dispositivo y el tamaño de la pantalla, lo que reduce el tamaño del archivo y mejora el rendimiento.

Al supervisar el rendimiento con herramientas como Lighthouse e implementar técnicas de optimización como la división de código, la optimización de imágenes, el almacenamiento en caché del lado del servidor y la gestión del rendimiento, puede garantizar que su aplicación Next.js ofrezca una experiencia de usuario óptima en producción. Supervise periódicamente las métricas de rendimiento y realice mejoras iterativas para optimizar continuamente el rendimiento de su aplicación.

Conclusión

Optimizar tus aplicaciones Next.js es vital para crear aplicaciones web ultrarrápidas, eficientes y muy atractivas. A lo largo de este capítulo, hemos cubierto diversas estrategias esenciales de optimización, como la adición de metadatos a las páginas, la implementación del servicio de archivos estáticos y el uso del componente de imagen Next.js para la optimización de imágenes.

También profundizamos en la arquitectura de Next.js y cómo configurarla para un rendimiento óptimo. Desde la implementación del renderizado del lado del servidor (SSR) y el prerenderizado, hasta la división de código y las importaciones dinámicas, el almacenamiento en caché y la mejora de la obtención de datos, y el análisis y la reducción del tamaño de los paquetes, lo tenemos cubierto.

Además, hemos analizado estrategias de implementación y mejores prácticas, así como la monitorización y optimización del rendimiento. Con estas técnicas, puede garantizar que sus aplicaciones Next.js ofrezcan una experiencia de usuario fluida y de alta calidad, a la vez que optimizan el rendimiento y reducen los tiempos de carga.

¡Puedes comenzar a implementar estas estrategias de optimización hoy y llevar tus aplicaciones Next.js al siguiente nivel!

En el próximo capítulo, profundizaremos en el enrutamiento basado en archivos de Next.js. El enrutamiento es un aspecto crucial de cualquier aplicación web, y Next.js lo simplifica con su sistema intuitivo y flexible de enrutamiento basado en archivos.

Preguntas de opción múltiple

1. ¿Cuál es el propósito de agregar metadatos a las páginas en Next.js?
 - A. Para mejorar el SEO
 - B. Para mejorar la experiencia del usuario
 - C. Para optimizar los tiempos de carga de la página
 - D. Todas las anteriores

2. ¿Qué componente se utiliza para agregar metadatos a las páginas en Next.js?

A. Título

B. Encabezado

C. Meta

D. Cabeza

3. ¿Cuál es el propósito de implementar el servicio de archivos estáticos en Next.js?

A. Para mejorar el SEO

B. Para mejorar la experiencia del usuario

C. Para optimizar los tiempos de carga de la página

D. Todas las anteriores

4. ¿Cuál es el propósito del componente de imagen Next.js?

A. o reducir el tamaño de los archivos de imagen

B. Para mejorar el SEO

C. Para mejorar la experiencia del usuario

D. Para que las imágenes se carguen más rápido

5. ¿Qué es la representación del lado del servidor (SSR) en Next.js?

A. Representación de páginas en el lado del cliente

B. Representar páginas en el lado del servidor y enviar el HTML al servidor.
cliente

C. Pre-renderizado de páginas en tiempo de compilación

D. Ninguna de las anteriores

Respuestas

1. D	2. D	3. C	4. D	5. B
------	------	------	------	------

Capítulo 6

Entendiendo el enrutamiento en Next.js

Introducción

El enrutamiento es como la columna vertebral de una aplicación web, garantizando una navegación fluida y una experiencia de usuario excepcional. En el mundo de Next.js, el enrutamiento es pan comido gracias a su enrutador integrado, que se encarga de todo el trabajo pesado al moverse entre páginas y gestionar rutas dinámicas. Así que, abróchense los cinturones y emprendamos un emocionante viaje por el mundo del enrutamiento de Next.js.

En general, este capítulo busca proporcionar una comprensión completa del funcionamiento del enrutamiento en Next.js. Abarca los fundamentos del enrutador de Next.js, la navegación entre páginas, la gestión de rutas dinámicas y el uso del componente Link para una navegación eficiente del lado del cliente.

Estructura

Los temas tratados en este capítulo son los siguientes:

- Comprender el rol del enrutador Next.js
- Comprender el componente Link de Next.js y su uso
- Navegar entre páginas en Next.js usando el enrutador
- Trabajar con rutas dinámicas en Next.js

Entendiendo el rol del enrutador Next.js

Esta sección proporciona una descripción general del enrutador Next.js y su importancia en el manejo de la navegación del lado del cliente. El enrutador Next.js es responsable de administrar

las rutas de la aplicación y la representación de los componentes apropiados en función de la URL solicitada.

El enrutador Next.js desempeña un papel fundamental en la gestión del enrutamiento dentro de una aplicación Next.js. Se encarga de gestionar la navegación entre diferentes páginas según las URL y de renderizar los componentes correspondientes. El enrutador Next.js se basa en React Router, una popular biblioteca de enrutamiento del lado del cliente, y añade funciones y optimizaciones adicionales específicas de Next.js.

Para comprender mejor la función del enrutador Next.js, consideremos un ejemplo. Supongamos que tenemos una aplicación Next.js con tres páginas: Inicio, Acerca de, y Contacto. Cada página tiene su propia URL y su componente correspondiente. Las URL de estas páginas son las siguientes:

Inicio: "/home"

Acerca de: "/acerca de"

Contacto: "/contacto"

Cuando un usuario ingresa la URL de nuestra aplicación en el navegador, el enrutador Next.js es responsable de hacer coincidir el R con las rutas definidas y renderizar el componente apropiado.

Por ejemplo, si un usuario introduce "/about" en la barra de direcciones del navegador y pulsa Intro, el enrutador de Next.js detecta la URL "/about" y muestra el componente About, que corresponde a esa ruta. El enrutador gestiona este proceso sin problemas, garantizando que se muestre el componente correcto al usuario sin tener que actualizar la página completa.

El enrutador Next.js también admite rutas anidadas, lo que nos permite crear estructuras de página más complejas. Por ejemplo, podríamos tener una ruta anidada para una entrada de blog:

Entrada del blog: "/blog/[slug]"

En este caso, [slug] representa un parámetro dinámico que puede ser cualquier valor.

El enrutador Next.js puede extraer el valor del parámetro slug de la URL y pasarlo como propiedad al componente correspondiente.

En general, el enrutador Next.js simplifica la gestión de rutas y la representación de componentes según la URL actual. Proporciona una experiencia de navegación fluida dentro de una aplicación Next.js.

Ejemplo de uso:

Para definir rutas en Next.js, creamos archivos individuales dentro del directorio de la página.

Por ejemplo, podemos crear un archivo llamado pages/about.js para definir la ruta de la página "Acerca de". Dentro del archivo about.js , exportaríamos un componente de React que representa la página " Acerca de ":

```
// páginas/acercede.js

importar React desde "react";

const Acerca de = () => {

    devolver (
        <div>

            Acerca de la página

            <p>Este es el contenido de la página Acerca de.</p>

        </div>

    );
};

exportar predeterminado Acerca de;
```

De manera similar, podemos crear archivos para otras páginas, como ome y Contacto, cada una exportando sus respectivos componentes React.

Con el enrutador Next.js instalado, podemos navegar entre estas páginas usando el componente <Link>. Por ejemplo, para navegar desde la página de inicio a la página "Acerca de ", podemos usar el siguiente código:

```
// pages/index.js (Página de inicio)

importar React desde "react";
importar enlace desde "siguiente/enlace";

const Inicio = () => {

    devolver (
        <div>

            Página de inicio

            <p>Bienvenido a la página de inicio.</p>
            <Link href="/acerca de">

                <a>Ir a la página Acerca de</a>

            </Enlace>

        </div>

    );
};
```

```
};  
exportar predeterminado Inicio;
```

En este fragmento de código, el componente `<Link>` se utiliza para crear un enlace a la página Acerca de ("about"). Cuando el usuario hace clic en el enlace "Ir a la página Acerca de", se abre el siguiente. El enrutador js maneja la navegación y renderiza el componente Acerca de sin actualizar la página completa.

El enrutador Next.js facilita la definición de rutas y la navegación entre páginas, proporcionando una experiencia de usuario perfecta dentro de una aplicación Next.js.

En la última versión 13+ de Next.js, la estructura de enrutamiento ha sufrido un ligero ajuste. Aquí está la estructura de carpetas actualizada con una breve descripción.

```
- aplicación  
  - acerca de  
    - page.js # Ahora se asigna a "/about"  
  - productos  
    - [ID del producto]  
      - page.js # Ahora se asigna a "/products/:productId"
```

Esta estructura muestra la jerarquía dentro del directorio de la aplicación . Cabe destacar que la página "Acerca de" y la ruta "Productos" ahora se encuentran en sus respectivas carpetas. Cabe destacar que los archivos dentro de estas carpetas, como `page.js`, se han optimizado para alinearse directamente con rutas específicas de R. Esta mejora mejora la organización general y la facilidad de uso del sistema de enrutamiento en Next.

js, promoviendo un enfoque más ágil e intuitivo para el desarrollo de aplicaciones web.

Comprender el componente Link de Next.js y su uso

El componente `Link` de Next.js es una potente herramienta para crear enlaces de navegación del lado del cliente dentro de una aplicación Next.js. Ofrece una forma optimizada de gestionar la navegación, garantizando transiciones de página rápidas y preservando el estado de la aplicación.

Esta sección explica el uso del componente Enlace y sus diversas propiedades y opciones.

El componente Enlace de Next.js es una función integrada que permite la navegación del lado del cliente entre páginas dentro de una aplicación Next.js. Proporciona una forma sencilla y optimizada de crear enlaces, garantizando transiciones de página rápidas y preservando la...

Estado de la aplicación. Aquí hay una explicación de las propiedades del componente de tinta, junto con un ejemplo de código para demostrar su uso:

- href (cadena u objeto):

La propiedad href especifica el destino R del enlace. Puede ser una cadena que representa una ruta estática o un objeto que contiene parámetros de ruta dinámicos.

```
importar enlace desde 'next/link';
// Ruta estática
<Link href="/acerca de">
    Acerca de
</Enlace>

// Ruta dinámica con parámetros
<Link href="/usuarios/[id]" as="/usuarios/1">
    perfil de usuario
</Enlace>
```

- como (cadena, opcional):

La propiedad "as" se utiliza en combinación con rutas dinámicas para especificar la URL que se mostrará en la barra de direcciones del navegador. Solo es necesaria al usar rutas dinámicas.

- reemplazar (booleano, opcional):

La propiedad "replace" determina si la navegación debe reemplazar la página actual en el historial del navegador o agregar una nueva entrada. Por defecto, está establecida en "false", lo que significa que se agregará una nueva entrada.

```
<Link href="/contacto" reemplazar>
    <a>Contacto</a>
</Enlace>
```

- desplazamiento (booleano o cadena, opcional):

La propiedad de desplazamiento controla el comportamiento de desplazamiento cuando se hace clic en el enlace.

Se puede configurar como verdadero para desplazarse hasta la parte superior de la página, falso para preservar la posición de desplazamiento actual o un valor de cadena para desplazarse hasta un ID de elemento específico en la página de destino.

```
<Link href="/blog" scroll={true}>  
<a>Blog</a>  
</Enlace>
```

- **superficial** (booleano, opcional):

La propiedad superficial permite el enrutamiento superficial, lo que significa que el componente de página no se volverá a renderizar si solo cambian los parámetros de consulta. Resulta útil cuando se desea actualizar la URL sin obtener nuevos datos ni volver a ejecutar las propiedades iniciales de la página.

```
<Link href="/productos" superficial>  
<a>Productos</a>  
</Enlace>
```

- **prefetch** (booleano, opcional):

La propiedad prefetch permite la precarga automática de los datos de la página enlazada en segundo plano. Esto puede mejorar la experiencia del usuario al reducir el tiempo de carga al visitar la página enlazada.

```
<Link href="/dashboard" prefetch>  
<a>Panel de control</a>  
</Enlace>
```

El componente Enlace envuelve la etiqueta de ancla () , proporcionando la funcionalidad necesaria para la navegación del cliente. Gestiona los clics y evita el comportamiento de navegación predeterminado del navegador.

Al usar el componente Link, Next.js optimiza la navegación y aprovecha sus funciones integradas para mejorar la experiencia del usuario. Permite transiciones fluidas entre páginas, mejora el rendimiento y...

permite a los desarrolladores aprovechar funciones avanzadas como el enrutamiento dinámico y la precarga.

- **passHref (booleano, opcional):**

La propiedad passref, cuando se establece en verdadero, transfiere la propiedad href a la etiqueta <a> subyacente. Esto resulta útil cuando se necesitan aplicar propiedades adicionales o estilos personalizados a la etiqueta de anclaje.

```
tinta href=/perfil passref>
    un nombre de clase=customlink>rofile/a>
</Enlace>
```

- **activo (booleano, opcional):**

La propiedad active permite aplicar un estilo o clase específicos al enlace cuando coincide con la ruta actual. Esto puede ser útil para resaltar el enlace activo en un menú de navegación.

```
<Link href="/about" active={true}>
    Acerca de
</Enlace>
```

- **onClick (función, opcional):**

La propiedad onClick permite definir una función personalizada que se ejecuta al hacer clic en el enlace. Esto puede ser útil para realizar acciones adicionales o activar eventos antes de la navegación.

```
const handleClick = () => {
    // Realizar acciones adicionales antes de la navegación
    console.log('Enlace hecho clic');
};
```

```
<Link href="/contacto" onClick={handleClick}>
    <a>Contacto</a>
</Enlace>
```

- Enlaces a URL externas:

Además de la navegación interna, el componente Enlace también permite crear enlaces a URL externas. Simplemente proporciona la URL completa como href. propiedad y Next.js manejará la redirección en consecuencia.

```
<Link href="https://www.ejemplo.com">
  Enlace externo
</Link>
```

El componente Link de Next.js simplifica la creación de enlaces de navegación en una aplicación Next.js. Garantiza una navegación eficiente del lado del cliente, gestiona rutas dinámicas y ofrece opciones de personalización y optimización. Al aprovechar el componente Link, puede crear una experiencia de usuario fluida e intuitiva al crear su aplicación web Next.js.

Navegar entre páginas en Next.js usando el enrutador

Esta sección explica cómo navegar entre diferentes páginas dentro de una aplicación Next.js usando el enrutador. Next.js proporciona una API sencilla e intuitiva para gestionar la navegación, lo que permite a los desarrolladores navegar programáticamente por diferentes rutas y transferir datos entre páginas.

La navegación entre páginas en Next.js se facilita gracias al enrutador de Next.js, que permite una navegación fluida del lado del cliente. El gancho useRouter es un gancho integrado de Next.js que proporciona acceso al objeto enrutador, lo que permite navegar entre páginas mediante programación. A continuación, se explica el proceso junto con un ejemplo de código.

- Importar el gancho useRouter:

En su componente Next.js, importe el gancho useRouter desde next/módulo de enrutador.

```
importar { useRouter } desde 'next/router';
```

- Acceder al objeto enrutador:

Llame al gancho useRouter dentro de su componente para obtener acceso al objeto enrutador.

```
enrutador constante = useRouter();
```

- Realizar navegación:

Utilice los métodos proporcionados por el objeto enrutador para navegar entre páginas. El método más común es "push", que permite navegar a una nueva página.

```
constante handleNavigation = () => {
  router.push('/acerca de');
};
```

También puedes utilizar otros métodos como reemplazar para reemplazar la página actual en el historial del navegador, o atrás para regresar a la página anterior.

- Manejar eventos de navegación:

Next.js ofrece varios eventos que puedes escuchar durante la navegación. Por ejemplo, puedes escuchar el evento routeChangeStart para realizar acciones antes de un cambio de ruta, o el evento routeChangeComplete para realizar acciones después de un cambio de ruta.

```
usarEfecto(() => {
  const handleRouteChangeStart = () => {
    // Acciones a realizar antes del cambio de ruta
  };
  constante handleRouteChangeComplete = () => {
    // Acciones a realizar después del cambio de ruta
  };
  router.events.on('routeChangeStart', handleRouteChangeStart);
  router.events.on('routeChangeComplete', handleRouteChangeCom-plete);

  devolver() => {
    router.events.off(cambio de rutalnicio, manejarcambio de rutalnicio);
    router.events.off(cambio de ruta completado, manejarcambio de ruta
completado);
  };
}, []);
```

Hay algunos detalles más sobre el gancho useRouter y sus propiedades en Next.js:

- consulta (objeto):

La propiedad de consulta del objeto enrutador contiene los parámetros de la cadena de consulta analizada de la URL. Permite acceder y recuperar los parámetros de consulta pasados a la página actual.

```
// URL: /productos?id=123  
console.log(router.query.id); // Salida: 123
```

- ruta de acceso (cadena):

La propiedad pathname representa la ruta de la URL actual, sin incluir los parámetros de consulta. Proporciona la ruta de la página actual dentro de la aplicación.

```
console.log(router.pathname); // Salida: /productos
```

- ruta (cadena):

La propiedad route devuelve el patrón de ruta coincidente actual. Representa la ruta asociada a la página actual.

```
console.log(router.route); // Salida: /productos
```

- basePath (cadena):

La propiedad basePath devuelve la ruta base de la aplicación. Representa la parte de la URL común a todas las páginas de la aplicación.

```
console.log(router.basePath); // Salida: /
```

- isFallback (booleano):

La propiedad isFallback indica si la página actual se genera estáticamente como reserva. Puede ser útil para renderizar contenido de reserva durante el estado de reserva.

```
si (enrutador.isFallback) {  
    regresar <div>Cargando...</div>;  
}
```

- eventos (EventEmitter):

La propiedad events proporciona un objeto EventEmitter que emite eventos relacionados con cambios de ruta. Puede usarlo para detectar diversos eventos, como el inicio, la finalización y el error de cambio de ruta.

```
router.events.on('routeChangeStart', () => {  
    console.log('Cambio de ruta iniciado');
```

```
});
```

Recuerde suscribirse a eventos usando `router.events.on()` y limpiar los escuchas de eventos usando `outeeetoff`.

- **empujar (método):**

El método `push` del objeto enrutador permite navegar a una nueva página mediante programación. Puede proporcionar la URL de destino como parámetro de cadena al método `push`.

```
constante handleNavigation = () => {  
    router.push('/acerca de');  
};
```

- **reemplazar (método):**

El método `replace` es similar a `push`, pero reemplaza la página actual en el historial del navegador en lugar de añadir una nueva entrada. Puede ser útil para implementar un botón de retroceso o cuando no se desea que la página actual sea accesible mediante dicho botón.

```
constante handleNavigation = () => {  
    router.replace('/acerca de');  
};
```

- **atrás (método):**

El método de retroceso permite volver a la página anterior en el historial del navegador. Es equivalente a la función del botón de retroceso del navegador.

```
constante handleNavigation = () => {  
    enrutador.back();  
};
```

- **prefetch (método):**

El método `prefetch` le permite precargar y almacenar en caché los recursos, avaScript, CSS, etc. de una página específica en segundo plano.

La precarga puede mejorar el rendimiento al precargar los recursos necesarios antes de que el usuario navegue a la página.

```
const prefetchPage = () => { router.prefetch('/acerca de'); };
```

- **isReady (booleano):**

La propiedad `isReady` indica si el objeto enrutador está listo y todo

Se han rellenado los valores iniciales (como los parámetros de consulta). Esto puede ser útil para la representación condicional según la disponibilidad del enrutador.

```
si (!router.isReady) {  
    regresar <div>Cargando...</div>;  
}
```

El gancho `useRouter` y sus propiedades en Next.js ofrecen potentes funciones para gestionar la navegación, acceder a parámetros de consulta, manipular la URL programáticamente y precargar recursos. Al utilizar estas funciones, puede crear aplicaciones Next.js dinámicas e interactivas con una navegación eficiente del lado del cliente y una carga de página optimizada.

Trabajar con rutas dinámicas en Next.js

Next.js admite rutas dinámicas, que incluyen parámetros o marcadores de posición en R. Esta sección explora cómo definir y trabajar con rutas dinámicas en Next.js. Abarca conceptos como los parámetros de ruta, el acceso a datos dinámicos dentro de los componentes y la generación de rutas dinámicas basadas en datos.

Trabajar con rutas dinámicas en Next.js le permite crear páginas que pueden aceptar parámetros dinámicos como parte de R. Esto le permite crear componentes flexibles y reutilizables que pueden representar contenido diferente en función de los parámetros proporcionados. Aquí hay una explicación de cómo trabajar con rutas dinámicas en Next.js, junto con un ejemplo de código que demuestra el uso del componente `Link` y el gancho `useRouter` para el enrutamiento dinámico:

- De finir una rutina nami

Para definir una ruta dinámica en Next.js, debe crear un archivo dentro del directorio de páginas con corchetes en el nombre del archivo. La porción dentro de los corchetes representa el parámetro dinámico.

Por ejemplo, si tiene una ruta dinámica para mostrar un producto con una ID, deberá crear un archivo llamado `productId.js` dentro del directorio de páginas.

- Vinculación a una ruta dinámica

El componente `Link` de Next.js permite navegar entre páginas. Para enlazar a una ruta dinámica, se debe proporcionar el parámetro dinámico como propiedad del atributo `href` del componente `Link`.

```
importar enlace desde 'next/link';
```

```
const ListaProductos = () => {
```

```
devolver (
  <div>
    <Link href="/productos/1">
      Producto 1
    </Enlace>
    <Link href="/productos/2">
      Producto 2
    </Enlace>
    {/* ... */}
  </div>
);
};

exportar lista de productos predeterminada;
```

En el ejemplo anterior, el componente Enlace se utiliza para crear enlaces a diferentes páginas de productos proporcionando el parámetro dinámico (por ejemplo, /productos/1, /productos/2) como valor href . •

Acceso al parámetro dinámico

Para acceder al parámetro dinámico en el componente de ruta dinámica, puede utilizar el gancho useRouter proporcionado por Next.js.

```
importar { useRouter } desde 'next/router';
const DetalleProducto = () => {
  enrutador constante = useRouter();
  const { productId } = router.query;
  devolver (
    <div>
      Detalle del producto: {productId}
      {/* ... */}
    </div>
  );
};

exportar predeterminado ProductDetail;
```

En el ejemplo anterior, se utiliza el gancho `useRouter` para acceder al objeto `router`. El objeto `router.query` contiene el parámetro dinámico, al que se puede acceder mediante la desestructuración de objetos (por ejemplo, `const { productId } = router.query`). A continuación, se puede utilizar el parámetro dinámico (`productId` en este caso) para obtener y mostrar los detalles relevantes del producto.

Trabajar con rutas dinámicas en Next.js le permite crear páginas que pueden manejar diferentes parámetros, proporcionando una forma flexible y reutilizable de mostrar contenido basado en los valores dinámicos. El componente `ink` simplifica la navegación a rutas dinámicas, mientras que el gancho `useRouter` permite el acceso al parámetro dinámico dentro del componente.

Conclusión

El capítulo "Entendiendo el Enrutamiento en Next.js" es una guía completa que explora conceptos y técnicas esenciales de enrutamiento, proporcionando a los desarrolladores una base sólida para navegar y gestionar rutas en aplicaciones Next.js. Abarca los fundamentos del enrutador de Next.js, la navegación de páginas, la gestión dinámica de rutas, el componente `Link`, el gancho `useRouter` y el enrutamiento basado en archivos. Al comprender la función del enrutador, los desarrolladores comprenden cómo se gestiona la navegación de páginas y cómo se asignan las diferentes rutas a sus componentes

correspondientes. El uso del componente `ink` simplifica la creación de etiquetas de anclaje para las transiciones de página, garantizando una navegación optimizada del lado del cliente y preservando las ventajas de la renderización en el servidor. Las rutas dinámicas permiten gestionar rutas con parámetros dinámicos, creando componentes flexibles y reutilizables que renderizan contenido diferente según los parámetros proporcionados. El gancho `useRouter` otorga acceso al objeto enrutador, proporcionando diversas propiedades y métodos para trabajar con el enrutamiento. El enrutamiento basado en archivos simplifica la organización de las rutas al asignar la estructura del archivo a la estructura de R. Este capítulo proporciona a los desarrolladores los conocimientos necesarios para navegar entre páginas, gestionar rutas dinámicas, utilizar el componente `Link` para optimizar la navegación del lado del cliente y aprovechar el gancho `useRouter` para mejorar el control y la interactividad del enrutamiento, creando experiencias de enrutamiento dinámicas, interactivas y fluidas en sus aplicaciones Next.js. El siguiente capítulo, "Gestión del estado en Next.js", explora las diversas técnicas y bibliotecas disponibles para la gestión del estado en aplicaciones

Consulte el repositorio `itub Building-Scalable-Web-Applications-with-Next.js-and-React` para los capítulos 5 y 6. Este repositorio presenta una pequeña aplicación de dos páginas que muestra una lista y detalles de libros, incorporando enrutamiento dinámico y técnicas de optimización. Puede usar este repositorio como una valiosa referencia para sus proyectos de desarrollo.

Preguntas de opción múltiple

1. ¿Qué es el enrutador Next.js?
 - A. Una herramienta para la renderización del lado del servidor en aplicaciones Next.js
 - B. Un componente para la navegación del lado del cliente en aplicaciones Next.js
 - C. Un mecanismo para gestionar el enrutamiento en aplicaciones Next.js
 - D. Función para organizar la estructura de archivos en aplicaciones Next.js
2. ¿Cuál es el propósito del componente Link en Next.js?
 - A. Para simplificar la creación de etiquetas de anclaje para las transiciones de página
 - B. Para manejar parámetros dinámicos en rutas
 - C. Otorgar acceso al objeto de enrutador y proporcionar propiedades y métodos para trabajar con el enrutamiento.
 - D. o organizar la estructura de archivos en proyectos Next.js
3. ¿Qué son las rutas dinámicas en Next.js?
 - A. Rutas que solo están disponibles en el lado del servidor
 - B. Rutas que manejan parámetros dinámicos como ID o slugs
 - C. Rutas que se asignan a la estructura de archivos en el directorio de páginas
 - D. Rutas que utilizan el gancho useRouter para un mejor control y interactividad
4. ¿Qué es el gancho useRouter en Next.js?
 - A. Una herramienta para la renderización del lado del servidor en aplicaciones Next.js
 - B. Un componente para la navegación del lado del cliente en aplicaciones Next.js
 - C. Un mecanismo para gestionar el enrutamiento en aplicaciones Next.js
 - D. Una herramienta poderosa que otorga acceso al objeto enrutador y proporciona varias propiedades y métodos para trabajar con el enrutamiento.

5. ¿Qué es el enrutamiento basado en archivos en Next.js?

- A. Una función para organizar la estructura de archivos en proyectos de Next.js
- B. Una herramienta para la navegación del lado del cliente en aplicaciones Next.js
- C. Un mecanismo para gestionar el enrutamiento en aplicaciones Next.js
- D. Una función para la renderización del lado del servidor en aplicaciones Next.js

Respuestas

1	C
2	A
3	B
4	D
5	A

Capítulo 7

Gestión de estados en Next.js

Introducción

La gestión de estados es un aspecto crucial en la creación de aplicaciones web, especialmente en aplicaciones complejas y dinámicas. En Next.js, la gestión de estados desempeña un papel importante en la gestión de datos, la interfaz de usuario y otros componentes de una aplicación. La gestión de estados se refiere a las técnicas y herramientas utilizadas para gestionar los datos y el estado de una aplicación de forma eficiente, escalable y fácil de mantener.

Next.js ofrece varias opciones para la gestión de estados, incluyendo React State, Redux y React Context. Cada una de estas opciones tiene sus ventajas y desventajas, y la elección de la más adecuada depende de las necesidades y requisitos específicos de la aplicación. En este capítulo, revisaremos cada una de estas opciones de gestión de estados y analizaremos sus ventajas e inconvenientes. También le guiaremos sobre cuándo usar cada una y cómo implementarlas en una aplicación Next.js.

Comenzaremos discutiendo los conceptos básicos de la administración de estado en Next.js, incluido por qué es importante y cómo puede mejorar el rendimiento y la escalabilidad de una aplicación. Luego profundizaremos en las diferentes opciones de administración de estado disponibles en Next.js, incluido el estado de React, Redux y el contexto de React, y compararemos sus fortalezas y debilidades.

Luego, el capítulo proporcionará instrucciones detalladas sobre cómo implementar la gestión de estado utilizando React State, incluido el uso de ganchos como useState y useContext. También discutiremos cómo configurar la tienda Redux, definir acciones y reductores y conectar componentes a la tienda.

Además, exploraremos cómo implementar la gestión estatal utilizando Contexto de React, incluida la creación de contexto, la definición de proveedores y consumidores,

y usar el contexto en los componentes. Proporcionaremos las mejores prácticas para administrar el estado en aplicaciones Next.js, incluyendo cómo evitar errores comunes y optimizar el rendimiento.

Finalmente, compartiremos casos prácticos y ejemplos sobre cómo implementar la gestión de estados en aplicaciones Next.js. Estos ejemplos cubrirán situaciones reales y las mejores prácticas para gestionar estados complejos.

Al final de este capítulo, los lectores tendrán una comprensión clara de las diferentes opciones de gestión de estados disponibles en Next.js, sus beneficios y desventajas, y cómo implementarlas de manera efectiva en sus aplicaciones.

Estructura

En este capítulo cubriremos los siguientes temas:

- Introducción a la gestión de estados en Next.js y su importancia
- Revisión de las diferentes opciones de gestión de estados disponibles en Next.js
- Pros y contras de cada opción de gestión estatal y cuándo utilizar cada una
- Cómo implementar la gestión de estados usando React State, incluido cómo Utilice ganchos como useState y useContext
- Mejores prácticas para administrar el estado en aplicaciones Next.js, incluyendo cómo evitar errores comunes y optimizar el rendimiento
- Cómo implementar la gestión de estados usando Redux, incluida la configuración de la tienda, la definición de acciones y reductores, y la conexión de componentes a la tienda
- Cómo implementar la gestión de estados usando el contexto de React, incluyendo la creación del contexto, la definición de proveedores y consumidores, y el uso del contexto en componentes
- Estudios de casos y ejemplos de cómo implementar la gestión de estados en aplicaciones Next.js, incluidos escenarios del mundo real y mejores prácticas para manejar estados complejos

Introducción a la gestión de estados en Next.js y su importancia

Al crear aplicaciones web, administrar el estado se convierte en una tarea crucial. El estado se refiere a los datos y la información que determinan el comportamiento y la apariencia de una interfaz de usuario en un momento dado. Incluye elementos como las entradas del usuario, las respuestas del servidor y el estado actual de varios componentes.

En Next.js, la gestión de estados es esencial porque nos ayuda a organizar y gestionar estos datos eficazmente. Nos permite mantener la coherencia, sincronizar diferentes partes de nuestra aplicación y actualizar la interfaz de usuario en respuesta a cambios en los datos o las interacciones del usuario.

Imagina que estás creando una función de carrito de compras para un sitio web de comercio electrónico con Next.js. El estado del carrito incluye artículos, cantidades y el precio total. Sin una gestión de estado adecuada, podrías encontrar problemas como la desaparición de artículos del carrito, cantidades incorrectas o precios inconsistentes. La gestión de estado minimiza estos problemas y proporciona una experiencia de usuario fluida y fiable.

Pero ¿es importante la gestión de estado en Next.js? Next.js es un potente framework para crear sitios web estáticos y renderizados en servidor. Ofrece funciones como renderizado SSR del lado del servidor, división automática de código y transiciones rápidas de página. Estas funciones ofrecen numerosas ventajas, pero también complican la gestión de estado.

Las aplicaciones Next.js suelen tener varias páginas con diferentes componentes, cada uno con sus propios requisitos de estado. A medida que los usuarios navegan por la aplicación, es necesario compartir datos entre páginas y componentes, lo que requiere una gestión cuidadosa del estado para garantizar un comportamiento y un rendimiento consistentes.

Una gestión de estado eficaz en Next.js puede mejorar el rendimiento y la escalabilidad de su aplicación. Permite gestionar flujos de datos complejos, evitar re-renderizados innecesarios y optimizar el uso de los recursos de red y servidor. Además, simplifica la depuración y el mantenimiento, ya que el estado está organizado de forma estructurada y se puede rastrear y modificar fácilmente.

Al adoptar técnicas adecuadas de gestión de estados en Next.js, puede crear aplicaciones sólidas y fáciles de mantener que brinden una experiencia de usuario fluida e interactiva.

En las siguientes secciones de este capítulo, exploraremos las distintas opciones de gestión de estados disponibles en Next.js, compararemos sus fortalezas y debilidades y proporcionaremos ejemplos prácticos y mejores prácticas para implementar la gestión de estados de manera efectiva.

Diferentes opciones de gestión de estados disponibles en Next.js

En Next.js, hay varias bibliotecas de gestión de estados disponibles que pueden ser

Se utilizan junto con React para gestionar el estado eficazmente. Estas bibliotecas ofrecen diversos enfoques y funciones para gestionar el estado en tus aplicaciones Next.js.

Estas son algunas de las bibliotecas de gestión de estados populares que se utilizan en React con Next.js:

- Redux: Redux es un contenedor de estado predecible que ayuda a gestionar el estado de la aplicación de forma centralizada. Sigue un flujo de datos unidireccional y proporciona un almacenamiento global para almacenar el estado. Redux es ampliamente utilizado y cuenta con una gran comunidad, lo que lo convierte en una opción popular para gestionar estados complejos en aplicaciones Next.js.
- React Context: React Context es una función integrada en React que permite crear y compartir estados entre componentes sin tener que pasar propiedades manualmente. Permite pasar datos a través del árbol de componentes sin pasarlo explícitamente por cada nivel. React Context es una opción ligera para la gestión de estados en aplicaciones pequeñas o para compartir datos entre componentes con pocos niveles de anidamiento.
- MobX es una biblioteca de gestión de estados sencilla y escalable que prioriza la observabilidad automática y la reactividad detallada. Permite crear objetos observables que permiten rastrear cambios y activar la re-renderización de componentes cuando se actualiza el estado. MobX se integra perfectamente con React y proporciona una API intuitiva para la gestión de estados en aplicaciones Next.js.

Zustand : Zustand es una biblioteca ligera de gestión de estados que aprovecha Reacthooks y Context PI. Ofrece un enfoque sencillo y funcional para la gestión de estados, con un enfoque en el rendimiento y una configuración minimalista. Zustand es una buena opción para aplicaciones pequeñas y medianas que requieren una solución de gestión de estados ligera y fácil de usar.

- recoil Recoil es una biblioteca de gestión de estados diseñada específicamente para aplicaciones React. Proporciona una forma sencilla y eficiente de gestionar el estado compartido entre componentes. Recoil utiliza átomos y selectores para definir y acceder al estado, y ofrece funciones como selectores asíncronos y compatibilidad integrada con estados derivados.

Estos son solo algunos ejemplos de bibliotecas de gestión de estado disponibles para aplicaciones Next.js. Cada biblioteca tiene sus propias características, ventajas y soporte de la comunidad. Dependiendo de los requisitos específicos y la complejidad de su proyecto, puede elegir la biblioteca más adecuada para gestionar el estado de su aplicación Next.js.

En la Tabla 7.1 se muestra una revisión de las seis opciones de gestión de estado populares disponibles en React Next.js, junto con una comparación:

Opción de gestión estatal	Características principales	alailit	Aprendiendo Curva	Comunidad Apoyo
	Estado de reacción Integrado en React	Adecuado para aplicaciones más pequeñas	Fácil de aprender	Fuerte apoyo de la comunidad
	Gestión centralizada del estado global	Altamente escalable	Mod-erar aprendiendo curva	Gran comunidad y extensa ecosistema
	Proporciona estado compartido a componentes anidados	Escalabilidad limitada	Fácil de aprender	Parte de Ecosistema de React
	Observabilidad automática y reactividad de grano fino	Altamente escalable	Fácil de aprender	Comunidad activa y buena documentación.
	Gestión de estados ligera y funcional	Adecuado para niños pequeños a aplicaciones de tamaño mediano	Fácil de aprender	Comunidad en crecimiento y centrada en la simplicidad
	Gestión de estados para React con átomos y selectores	Adecuado para medianos-s aplicaciones izadas	Fácil de aprender	Desarrollo de la comunidad y específico de Re-act

Tabla 7.1: Varias opciones de gestión del estado

Tenga en cuenta que la escalabilidad y la curva de aprendizaje son subjetivas y pueden variar según la complejidad de su aplicación y su familiaridad con la opción de gestión de estados elegida. Es importante considerar los requisitos específicos de su proyecto, la experiencia de su equipo y el apoyo de la comunidad al tomar una decisión.

Cada opción ofrece diferentes ventajas y desventajas, y la elección depende de factores como el tamaño de su aplicación, la complejidad de sus necesidades de gestión de estados y las preferencias de su equipo de desarrollo. Tómese el tiempo para evaluar estas opciones y considere las fortalezas y debilidades de cada una para encontrar la que mejor se adapte a su proyecto Next.js.

Pros y contras de las opciones de gestión estatal

La Tabla 7.2 muestra los pros y contras de cada opción de gestión estatal y cuándo utilizar cada una:

Estado Hombre-gestión Opción	Ventajas	Contras	Cuándo utilizarlo
Estado de reacción	Integrado en React, no requiere configuración adicional API simple e intuitiva para el estado del componente local	Limitado al estado del componente local No es adecuado para gestionar estados complejos o compartidos.	Pequeñas aplicaciones con necesidades de estado localizadas
Redux	Gestión centralizada del estado global Actualizaciones de estado predecibles Amplio ecosistema y soporte comunitario	Requiere configuración y código repetitivo Aprendizaje moderado curva	Aplicaciones grandes con necesidades complejas de gestión de estados, o cuando se desea depuración en tiempo real y herramientas extensas
Reaccionar Estafa-texto	Proporciona estado compartido a componentes anidados Evita la perforación de la hélice Parte de Ecosistema de React	Escalabilidad limitada para aplicaciones grandes. Puede haber una sobrecarga de rendimiento en componentes profundamente anidados.	Aplicaciones de tamaño mediano con requisitos moderados de intercambio de estado o cuando la simplicidad y la integración con React son importantes
o	Automático Observabilidad y reactividad de grano fino Altamente escalable Fácil de aprender	Puede requerir configuración adicional para Integración de React Puede tener una curva de aprendizaje para conceptos avanzados.	Aplicaciones que requieren una gestión de estados altamente escalable y reactiva, con preferencia por la simplicidad y la facilidad de usar
Estado actual	Gestión de estados ligera y funcional Fácil de aprender	Puede tener herramientas limitadas y soporte de la comunidad.	Aplicaciones pequeñas y medianas que priorizan la simplicidad, el rendimiento y un enfoque minimalista.

aceite de oliva	Gestión de estados para React con átomos y selectores Fácil de aprender	Biblioteca relativamente nueva y en evolución Comunidad en desarrollo	Aplicaciones creadas con Reaccionar que requiere flexibilidad y eficiencia gestión estatal, especialmente para proyectos de tamaño mediano
-----------------	--	--	--

Tabla 7.2: Pros y contras de cada opción de gestión estatal

administración estatal con Estado de React y el uso de ganchos

Aprendamos cómo implementar la gestión de estados usando React State y cómo usar ganchos como useState y useContext.

useState es un gancho integrado en React que permite añadir estado a componentes funcionales. Permite gestionar y actualizar el estado de un componente sin necesidad de usar componentes de clase.

Para implementar la gestión de estados mediante useState, puede seguir estos pasos:

1. Importe las dependencias necesarias:

```
importar React, { useState } de 'react';
```

- . Defina e inicialice el estado usando useState

```
const [estado, establecerEstado] = useState(valorInicial);
```

estado representa el valor actual de la variable de estado.

setState es una función utilizada para actualizar el estado.

initialalue es el valor inicial que desea asignar al estado.

3. Utilice el estado en sus componentes:

```
devolver (
  <div>
    <p>Valor del estado actual: {estado}</p>
    <button onClick={() => setState(newValue)}>Actualizar estado</button>
  </div>
);
```

4. Acceda al valor del estado usando `{state}` en JSX.

Para actualizar el estado, llama a `useState` con el nuevo valor que deseas asignar `nuevoValor`.

A continuación se muestra un ejemplo completo para ilustrar la implementación:

```
importar React, { useState } de 'react';
constante MiComponente = () => {
  const [count, setCount] = useState(0);

  const incrementCount = () => { setCount(count + 1);
};

  devolver (
    <div>
      <p>Conde: {count}</p>
      <button onClick={incrementCount}>Incrementar</button>
    </div>
  );
};

exportar predeterminado MyComponent;
```

En este ejemplo, `count` es la variable de estado que se administra mediante `useState` para registrar un valor de conteo. La función `incrementCount` actualiza el estado del conteo incrementándolo. El componente muestra el valor de conteo actual y un botón para activar el incremento.

Siguiendo estos pasos, puede implementar fácilmente la gestión de estados utilizando `useState` en sus componentes funcionales. Recuerde que el uso de la gestión de estado `useState` es local al componente en el que se declara.

"`useContext`" es un gancho integrado en React que permite acceder al valor de un objeto de contexto. Permite consumir valores de contexto en componentes funcionales sin necesidad de encapsular componentes con consumidores de contexto.

Para implementar la gestión de estados usando `useContext`, puede seguir estos pasos

- Crea un objeto de contexto usando React.createContext

```
constante MiContexto = React.createContext();
  •
    Rap su componente o componentes con el proveedor de contexto y
    Proporcionar un valor:
```

```
devolver (
  <MyContext.Provider valor={contextValue}>
    {/*Sus componentes aquí*/}
  </MiContexto.Proveedor>
);
```

contextValue representa el valor que desea compartir dentro del contexto.

- Consuma el valor del contexto en sus componentes usando useContext

```
const contextValue = useContext(MyContext);
Aquí, contextValue contendrá el valor proporcionado por el contexto.
```

A continuación se muestra un ejemplo completo para ilustrar la implementación:

```
importar React, { useContext } de 'react';
constante
MiContexto = React.createContext(); constante
Componente principal = () => {
  const contextValue = 'Hola desde el contexto'; return (
    <MyContext.Provider valor={contextValue}>
      <ComponenteHijo />
    </MiContexto.Proveedor>
  );
};
```

```
const ChildComponent = () => { const
  contextValue = useContext(MyContext); return (
    <div>
      <p>Valor de contexto: {contextValue}</p>
    </div>
  );
};
```

```
</div>
);
};

exportar componente principal predeterminado;
```

En este ejemplo, el componente principal encapsula el componente secundario con el proveedor de contexto, proporcionando el valor de contexto "Hola" del contexto. El componente secundario consume el valor de contexto usando useContext y lo renderiza.

Siguiendo estos pasos, puede implementar la gestión de estados mediante useContext para compartir y consumir valores de contexto dentro de sus componentes funcionales.

Recuerde ajustar el código según sus casos de uso y componentes específicos. estructura.

Mejores prácticas para gestionar el estado en las aliaciones de Next.js

Al gestionar el estado en aplicaciones Next.js, se recomienda seguir varias prácticas para evitar errores comunes y optimizar el rendimiento. A continuación, se presentan algunas prácticas clave a considerar.

- Inicie el estado global. Generalmente, se recomienda minimizar el uso del estado global tanto como sea posible. El estado global puede generar complejidad y dificultar el análisis del flujo de datos en la aplicación.
En lugar de ello, procure lograr una gestión estatal localizada dentro de componentes individuales o contextos de alcance más pequeño.
- Utilice las funciones integradas de gestión de estado de React, como useState , para gestionar el estado local de los componentes. Esto es ideal para gestionar el estado específico de un componente en particular y que no necesita compartirse entre varios componentes.
- Elija bibliotecas de gestión de estado prioritarias. Si tiene requisitos complejos de gestión de estado o necesita compartir el estado entre varios componentes, considere usar bibliotecas de gestión de estado como Redux o MobX. Estas bibliotecas ofrecen funciones avanzadas para la gestión de estado y pueden ayudar a organizar y escalar la gestión de estado de su aplicación.
- Normalizar la estructura del estado Normalice la estructura de su estado para mantenerla simple

Y organizado. Esto implica descomponer estructuras de datos complejas en entidades separadas y almacenarlas de forma normalizada. La normalización mejora el rendimiento y simplifica la recuperación y manipulación de datos.

- Evite re-renderizados innecesarios: optimice sus componentes para evitar re-renderizados innecesarios mediante el uso de técnicas como la memorización y React.memo. La memorización ayuda a evitar que se repitan cálculos costosos en cada renderizado, mientras que React.memo Puede evitar la nueva representación de componentes cuando sus propiedades no han cambiado.
- Los datos inmutables ayudan a optimizar el rendimiento, ya que permiten que React determine eficientemente si es necesario volver a renderizar un componente. Evite modificar directamente los objetos de estado y, en su lugar, cree nuevas copias o utilice estructuras de datos inmutables como Immutable.js o Immer.
- React Context es una herramienta potente para compartir el estado entre componentes, pero puede afectar el rendimiento si se usa en exceso. Use el contexto con criterio y considere si alternativas como la perforación de propiedades o la elevación de estado podrían ser más apropiadas para su caso de uso específico .

Optimizar las solicitudes de API: al obtener datos de los PI, implementar técnicas como el almacenamiento en caché, la eliminación de rebotes o la paginación para minimizar las solicitudes innecesarias y mejorar el rendimiento. Considerar el uso de bibliotecas como SR o React uery, que ofrecen capacidades de obtención y almacenamiento en caché de datos diseñadas específicamente para aplicaciones React.

- Monitoreo del rendimiento: Monitoree el rendimiento de su aplicación con herramientas como Lighthouse, Chrome Devtools o bibliotecas de perfiles de rendimiento. Identifique y solucione cualquier cuello de botella o problema de rendimiento relacionado con la gestión del estado.

Pruebas y refactorización: Realice pruebas unitarias exhaustivas para su código de gestión de estados para garantizar su corrección y mantenibilidad. Revise y refactorice periódicamente su código de gestión de estados para mantenerlo limpio, organizado y optimizado.

Si sigue estas prácticas recomendadas, podrá administrar eficazmente el estado de sus aplicaciones Next.js, a la vez que optimiza el rendimiento y evita errores comunes.

Recuerde que las necesidades específicas de su aplicación pueden variar, así que siempre considere el contexto y los requisitos de su proyecto al implementar soluciones de gestión de estado.

Gestión del estado de mplementing sing ex en Next.js liation

Redux es una biblioteca de gestión de estados que se usa comúnmente con frameworks de JavaScript como React y Next.js. Proporciona una forma predecible de gestionar el estado de una aplicación mediante la aplicación de un conjunto estricto de principios y patrones. Redux sigue un flujo de datos unidireccional, lo que facilita la comprensión y la depuración de los cambios de estado.

Los tres principios de Redux son:

- Fuente única de información: Redux promueve la idea de tener una única fuente de información para el estado de la aplicación. El estado completo se almacena en un objeto centralizado llamado almacén Redux. Esto permite que todos los componentes de la aplicación accedan y actualicen el estado de forma consistente y predecible.
- El estado es de solo lectura e inmutable: En Redux, el estado se trata como de solo lectura, lo que significa que no se puede modificar directamente. En su lugar, se envían acciones para describir los cambios de estado. Estas acciones son objetos JavaScript simples que contienen información sobre lo que sucedió en la aplicación. Los reductores, funciones puras en Redux, toman el estado actual y la acción enviada como entrada y devuelven un nuevo objeto de estado, sin modificar el estado original. Esta inmutabilidad garantiza que el estado siga siendo predecible, facilita la detección eficiente de cambios y habilita funciones como la depuración temporal.

Los cambios se realizan con funciones puras: Redux utiliza funciones puras llamadas reductores para actualizar el estado. Los reductores toman el estado actual y una acción, y devuelven un nuevo estado. No tienen efectos secundarios ni interactúan directamente con recursos externos. Al usar funciones puras, Redux hace que los cambios de estado sean predecibles y comprobables. Además, permite código modular y reutilizable, ya que los reductores pueden configurarse para gestionar diferentes partes del estado.

lx

Flux es un patrón arquitectónico introducido por Facebook que complementa la arquitectura basada en componentes de React. Flux aborda los desafíos de la gestión del estado en aplicaciones a gran escala. Implantó un flujo unidireccional de datos y promovió una clara separación de intereses.

El flujo consta de varios componentes clave

- Acciones: Las acciones describen eventos o intenciones del usuario y contienen los datos asociados a cada evento. Se activan mediante interacciones del usuario o eventos del sistema.
- Despachador: El Despachador se encarga de recibir acciones y enviarlas a las tiendas registradas. Actúa como un centro neurálgico en la arquitectura Flux.
- Almacenes: Los almacenes almacenan el estado de la aplicación y la lógica de negocio. Reciben acciones del despachador y actualizan su estado según corresponda. Los almacenes son responsables de determinar cómo debe cambiar el estado en respuesta a las acciones.
 - Las vistas representan los componentes de la interfaz de usuario de su aplicación. Recuperan datos de las tiendas y representan la interfaz de usuario en función del estado.
 - Flux aplica un flujo de datos unidireccional, donde las acciones fluyen a través del Despachador a los Almacenes, que luego actualizan el estado. Las vistas escuchan los cambios en el estado y vuelven a renderizarse según sea necesario.

Viniendo ex un Ix

Redux y Flux comparten conceptos y principios similares, lo que los hace compatibles para la gestión de estados. Redux puede considerarse una implementación de la arquitectura Flux, donde el almacén de Redux reemplaza la función de los almacenes de Flux, y sus reductores reemplazan la lógica de estos últimos.

En un enfoque combinado de Redux y Flux, puede usar Redux como la biblioteca de gestión de estado subyacente y seguir los principios de Redux de fuente única de verdad, inmutabilidad y reductores puros. El flujo de datos unidireccional y la separación de preocupaciones que proporciona Flux se pueden integrar con Redux para mejorar la organización y la escalabilidad de la gestión del estado de su aplicación.

Aquí tienes una explicación paso a paso de cómo puedes combinar Redux y Flux:

- Utilice Redux como biblioteca de gestión de estado: configure Redux en su aplicación Next.js creando una tienda Redux, definiendo reductores para manejar actualizaciones de estado y conectando sus componentes a la tienda Redux usando la función de conexión de React Redux.
- Adopte el flujo de datos unidireccional de Flux. Si bien Redux ya sigue un flujo de datos unidireccional, puede mejorarlo aún más adoptando el patrón de Flux. Las acciones se envían desde sus componentes al almacén de Redux, que actualiza el estado mediante reductores. El estado actualizado se propaga a los componentes, activando nuevos renderizados.

- Aproveche la separación de preocupaciones de Flux organizando sus reductores Redux según dominios o características específicos de su aplicación. Cada reductor puede manejar una porción específica del estado, de manera similar a cómo los almacenes de Flux administran diferentes partes de los datos de la aplicación.
- Combinar acciones y reductores: Crea acciones en Redux para describir cambios de estado, tal como lo harías en Flux. Envía estas acciones al almacén de Redux, que a su vez invoca los reductores correspondientes. Los reductores gestionan las acciones y actualizan el estado según corresponda, siguiendo los principios de inmutabilidad y funciones pures.

Al combinar Redux y Flux, puede beneficiarse de las potentes capacidades de gestión de estado de Redux, a la vez que aprovecha el flujo de datos claro y la separación de preocupaciones de Flux. Esta combinación le ayuda a mantener una única fuente de información veraz para el estado de su aplicación, garantizar la inmutabilidad y las actualizaciones de estado predecibles, y organizar su lógica de gestión de estado de forma modular y escalable.

Recuerde que, si bien Redux y Flux se pueden combinar, es fundamental considerar la complejidad y las necesidades de su aplicación. Si su aplicación es relativamente pequeña o no requiere una gestión de estado exhaustiva, usar Redux solo puede ser suficiente. Sin embargo, para aplicaciones más grandes y complejas, combinar Redux con Flux puede proporcionar una solución eficaz para gestionar el estado de forma estructurada y fácil de mantener.

La figura 7.1 muestra el diagrama de flujo básico que ilustra el flujo básico de datos en Flux y Redux

- La vista de servicio representa los componentes de la interfaz de usuario de su aplicación.
- Las interacciones del usuario en la vista activan acciones. Las acciones describen al usuario, intenciones o eventos del sistema.
- Las acciones se envían al despachador.
- El Despachador actúa como un centro para recibir acciones y enviarlas a las Tiendas registradas en Flux, o a los Reductores en Redux.

En Flux, los almacenes almacenan el estado de la aplicación y la lógica de negocio. En Redux, los reductores son funciones pures que actualizan el estado según la acción ejecutada.

- Una vez que los Stores Flux o Reducers Redux actualizan su estado, emiten un evento de cambio.
- La vista del servicio escucha los eventos de cambio emitidos por Stores Flux o se suscribe a Store Redux y actualiza su visualización en función de los cambios. nuevo estado.

Tanto Flux como Redux siguen un flujo de datos unidireccional, donde las acciones desencadenan cambios de estado a través del Dispatcher Flux o los Reducers Redux. El estado se actualiza en Stores Flux o Store Redux, y la vista del servicio refleja los cambios al volver a renderizarse.

Tenga en cuenta que Flux y Redux presentan algunas diferencias en la gestión del estado, como la presencia de Acciones y Despachadores en Flux y la introducción de Reductores en Redux. Sin embargo, tanto Flux como Redux buscan proporcionar un enfoque claro y estructurado para la gestión del estado en aplicaciones complejas.



Figura 7.1: Ilustración del flujo básico de datos en Lux y Redux

Para implementar la gestión de estados usando Redux con Flux en una aplicación simple Next.js, puedes seguir estos pasos:

Paso 1: Configurar un proyecto Next.js

Empieza por configurar un proyecto básico de Next.js con tu método preferido. Puedes

Utilice Next.js CI o cree un nuevo proyecto manualmente. Asegúrese de tener instaladas las dependencias necesarias.

Paso Instalar Redux y paquetes relacionados

En el directorio de tu proyecto, instala los paquetes necesarios para la integración de Redux y Flux. Ejecuta el siguiente comando:

```
npm instalar redux react-redux redux-thunk
```

Paso 3: Crea la tienda Redux

En su proyecto, cree un nuevo archivo llamado store.js. Este archivo contendrá la configuración para su tienda Redux. Agregue el siguiente código

```
importar { createStore, applyMiddleware } desde 'redux'; importar thunk desde
'redux-thunk';

// Importa tus reductores aquí
importar rootReducer desde './reducers';

const store = createStore(rootReducer, applyMiddleware(thunk)); exportar tienda predeterminada;
```

Asegúrese de tener sus reductores definidos en archivos separados dentro del reductor . directorio.

Paso Define tus reductores

Cree archivos separados para cada reductor en el directorio de reductores. Cada archivo de reductor debe exportar una función que tome el estado actual y una acción como parámetros, y devuelva un nuevo estado. Por ejemplo:

```
// Ejemplo de reductor en counterReducer.js const initialState
= {
  cuenta: 0,
};
```

```
const counterReducer = (estado = estadioinicial, acción) => {
```

```

switch (acción.tipo) { caso 'INCREMENTO':
    return { ...estado, conteo: estado.conteo + 1 }; caso 'DECREMENTO':
    retorna { ...estado, conteo: estado.conteo - 1 }; predeterminado:
    estado de retorno;
}
};

exportar contador predeterminadoReducer;

```

Paso 5: Crea tus acciones

Cree archivos separados para sus acciones en el directorio de acciones. Cada archivo debe exportar funciones del creador de acciones que devuelvan un objeto de acción. Por ejemplo:

```

// Ejemplo de acciones en la exportación de counterActions.js
incremento constante = () => ({
    tipo: 'INCREMENTO',
});

exportar const decremento = () => ({ tipo:
    'DECREMENTO',
});

```

Paso 6: Combinar reductores

En el directorio de reductores , crea un archivo index.js para combinar todos tus reductores en un único reductor raíz. Por ejemplo:

```

// reductores/index.js
importar { combineReducers } de 'redux'; importar counterReducer
de './counterReducer';

```

```
const rootReducer = combineReducers({ contador: contadorReducer,  
});  
exportar rootReducer predeterminado;
```

Paso 7: Crea tus componentes

Cree componentes de React que usen las acciones de almacenamiento y envío de Redux. Por ejemplo,

```
// Componente de ejemplo en Counter.js import  
React from 'react';  
importar { connect } desde 'react-redux';  
importar { incremento, decremento } de './actions/counterActions';
```

```
const Contador = ({ contar, incrementar, decrementar }) => { return (  
  
<div>  
  <h1>Conteo: {count}</h1>  
  <button onClick={increment}>Incrementar</button>  
  <button onClick={decrement}>Decrementar</button>  
</div>  
);  
};
```

```
const mapStateToProps = (estado) => ({ count:  
  estado.contador.conteo,  
});
```

```
constante mapDispatchToProps =  
{ incremento,  
decremento,
```

```
};
```

```
exportar predeterminado connect(mapStateToProps, mapDispatchToProps)(Counter);
```

Paso 8: Configura tus páginas Next.js

Crea una página de Next.js donde quieras renderizar tus componentes conectados a Redux. Importa los componentes y úsalos en tu página. Por ejemplo:

```
// páginas/index.js
importar React desde 'react';
importar Contador desde '../componentes/Contador'; const
HomePage = () => {
    devolver (
        <div>
            Ejemplo de Redux con Flux
            <Contador />
        </div>
    );
};

exportar página de inicio predeterminada;
```

Paso Rapea tu aplicación Next.js con el proveedor Redux

En el archivo pagesapp.js, envuelva su aplicación Next.js con el componente Redux Provider y proporcione la tienda Redux creada en el Paso 3. Por ejemplo,

```
// páginas/_app.js
importar { Proveedor } desde 'react-redux'; importar
tienda desde '../tienda';

función MyApp({ Componente, pageProps }) {
```

```
devolver (  
<Tienda del proveedor={tienda}>  
<Componente {...pageProps} />  
</Proveedor>  
);  
}  
  
exportar predeterminado MyApp;
```

Paso 10: Ejecuta tu aplicación Next.js

Inicie su servidor de desarrollo Next.js y navegue a la página donde usó el componente conectado a Redux. Ahora debería ver el componente contador con botones para aumentar y disminuir el contador.

¡Listo! Has implementado la gestión de estados usando Redux con Flux en una aplicación sencilla de Next.js. Redux proporciona la fuente única de información y capacidades de gestión de estados, mientras que los principios de Flux de acciones, despachador y almacenamiento están integrados en la arquitectura de Redux.

Puedes añadir más acciones, reductores y componentes según sea necesario, siguiendo los mismos patrones. Este enfoque te permite gestionar el estado de tu aplicación Next.js eficazmente y gestionar interacciones de estado complejas con facilidad.

Redux Thunk

Redux Hunk es un middleware para Redux, una popular biblioteca de gestión de estados en JavaScript. Permite escribir creadores de acciones que devuelven funciones en lugar de objetos de acción simples. Esto permite realizar operaciones asíncronas, como llamadas a la API, dentro de los creadores de acciones antes de ejecutar las acciones.

es ex n neee

En Redux, las acciones suelen ser objetos simples con un tipo y una carga útil. Sin embargo, existen escenarios en los que podría ser necesario realizar operaciones asíncronas, como obtener datos de un PI, antes de ejecutar una acción. Redux Hunk proporciona una forma de gestionar estas operaciones asíncronas y ejecutar acciones en el momento oportuno.

Implementing thunk with an example

Para usar Redux Thunk, debes configurarlo en tu tienda Redux y crear el thunk creadores de acción.

Configurar el middleware Redux thunk

```
importar { createStore, applyMiddleware } desde 'redux'; importar thunk
desde 'redux-thunk';

importar rootReducer desde './reducers';

const store = createStore(rootReducer, applyMiddleware(thunk)); exportar tienda
predeterminada;
```

Aquí, importamos el middleware thunk y lo aplicamos a la tienda usando applyMiddleware de Redux. rootReducer representa sus reductores combinados.

Crear creadores de acciones thunk

Los creadores de acciones de Thunk son funciones que pueden enviar acciones y realizar operaciones asíncronas. A continuación, un ejemplo:

```
// acciones.js

importar axios desde 'axios'; exportar const
fetchData = () => {
  devolver asíncrono (despacho) => {
    despacho({ tipo: 'OBTENER_DATOS_SOLICITUD' }); intentar {
      constante respuesta = await axios.get('https://api.example.com/data'); despacho({ tipo:
      'FETCH_DATA_SUCCESS', carga útil: respuesta.data });
    } captura (error) {
      despacho({ tipo: 'FETCH_DATA_FAILURE', carga útil: error.message });
    }
  };
};
```

En este ejemplo, fetchData es un creador de acciones de procesador que obtiene datos de un PI.

Devuelve una función que recibe la función de envío como argumento.

Dentro de esta función, puedes enviar múltiples acciones para representar las diferentes etapas de la operación asíncrona.

Despachar creadores de acciones thunk

Para enviar un creador de acciones de procesador, debe usar la función de envío proporcionada por Redux. Aquí tiene un ejemplo de cómo enviar fetchData. creador de acciones thunk

```
// componente.js
importar eact, { useEffect } de react;
importar { useDispatch, useSelector } de 'react-redux';
importar { fetchData } desde './actions';
constante Componente = () => {
  constante despacho = useDispatch();
  const datos = useSelector((estado) => estado.datos);
  usarEfecto(() => {
    despacho(fetchData());
  }, [despacho]);
}
```

```
// Representar el componente usando los datos obtenidos
// ...
devolver <div>{datos}</div>;
};

exportar componente predeterminado;
```

En este ejemplo, usamos el gancho useDispatch de la biblioteca react-redux para obtener acceso a la función de despacho . Luego, enviamos el creador de la acción del thunk fetchData dentro de un gancho useEffect para activar la llamada PI cuando el componente monturas.

Este es un resumen básico de Redux Hunk y cómo implementarlo. Permite gestionar operaciones asincrónicas dentro de las acciones de Redux y gestionar el estado en consecuencia.

Nota: El Capítulo 12, Desarrollo de una Aplicación CRD con Next.js, cubre una implementación completa de Redux y Flux en nuestra aplicación, proporcionando una potente solución de gestión de estados. Continúe leyendo para aprender a configurar el almacén de Redux, definir acciones y reductores, y conectar componentes para un flujo de datos eficiente.

Gestión del estado de administración, incluyendo el texto en una simple aliasación Next.js

El contexto de React es una función de React que permite compartir datos entre componentes sin pasar propiedades explícitamente en cada nivel del árbol de componentes. Permite crear un estado global al que cualquier componente puede acceder dentro de su árbol.

Visualización del problema

- Imagina que tienes dos componentes, A y B, ambos dentro de la aplicación principal componente.
- El componente B necesita un valor de estado del componente A.
- Pasar este valor directamente es complicado debido a la estructura de sus componentes.

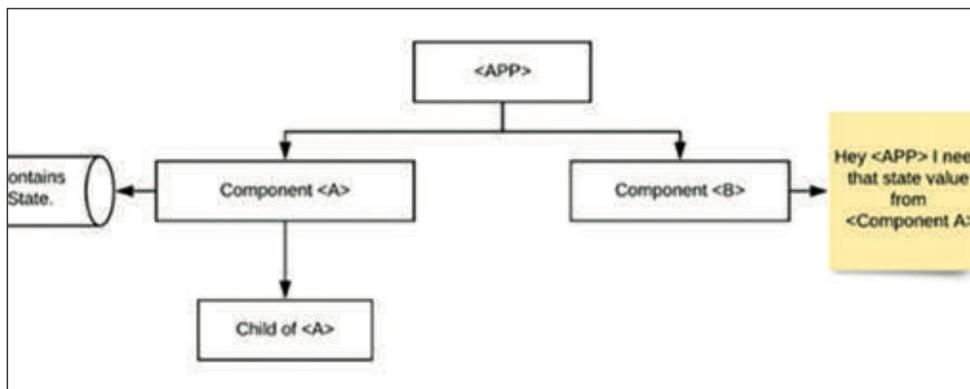


Figura 7.2: Descripción general del enunciado del problema

Solución 1: Estado Elevador:

- Mueva el estado requerido del Componente A a la aplicación principal componente.
- Esto resuelve el problema inmediato, pero no es práctico para grandes aplicaciones.

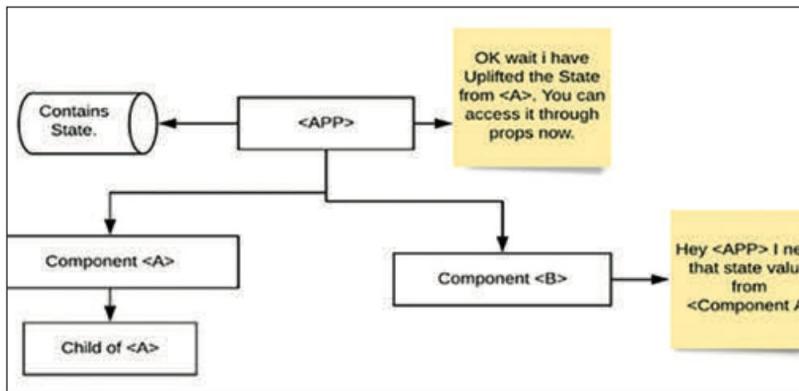


Figura 7.3: estado de lifting

Solución 2: API de contexto:

- La API de contexto proporciona una forma de crear variables globales compartidas entre componentes.
- Es una alternativa a la perforación manual de la hélice.
- Se utiliza mejor cuando los datos necesitan ser accesibles para muchos componentes a la vez.
- diferentes niveles de anidación.

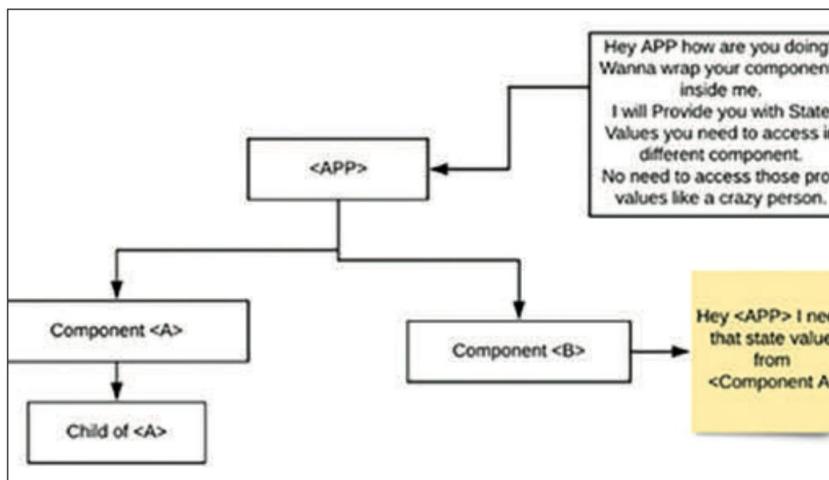


Figura 7.4: Solución donde la API de contexto entra en escena

Creación de API de contexto:

- Configurar la estructura de directorio con la aplicación principal, el componente A y el componente B.
- Cree un archivo llamado Context.js con código para crear un contexto usando createContext.

Creación de un proveedor:

- Extraer el proveedor del contexto creado.
- Cree un componente AppProvider que proporcione valores a los consumidores componentes.
- Dentro de AppProvider, configure el estado usando useState para los datos que desea compartir.
- Rap su componente principal PP con AppProvider.

Consumiendo el contexto:

- Importa ApplicationContext y useContext en el componente donde quieras consumir el contexto.
- Utilice el gancho useContext para acceder a los valores de contexto.

Consejos para el uso del contexto:

- No reemplace el estado local con el contexto; use el estado local siempre que sea posible.
- Cambiar los valores de contexto provoca nuevas representaciones en los componentes que utilizan ese contexto.
- Evite envolver toda la aplicación con el proveedor de contexto; úselo en el el padre más pequeño necesario.

En resumen:

- Utilice createContext para crear el contexto.
- Extraer Proveedor del contexto.
- Rap el componente padre con el proveedor.
- Consumir contexto usando los ganchos useContext .

Context PI simplifica el intercambio de datos entre componentes sin necesidad de realizar perforaciones. Sin embargo, es importante usarlo con prudencia y evitar el uso excesivo. Sirve como mecanismo para actualizar y acceder a ese estado en diferentes componentes.

Este flujo representa el flujo básico de datos en el contexto de React. Describe los pasos necesarios para crear el contexto, configurar el proveedor, definir acciones, consumir el contexto, acceder y actualizar el estado, y volver a renderizar los componentes.

Para implementar la gestión de estados usando el contexto React en una aplicación simple Next.js, puedes seguir estos pasos:

Paso 1: Configurar un proyecto Next.js

Empieza por configurar un proyecto básico de Next.js con tu método preferido. Puedes

Utilice Next.js CI o cree un nuevo proyecto manualmente. Asegúrese de tener instaladas las dependencias necesarias.

Paso Crear un nuevo archivo de contexto

Crea un nuevo archivo llamado AppContext.js en tu proyecto. Este archivo contendrá la configuración de tu contexto de React. Agrega el siguiente código:

```
// AppContext.js

importar React, { createContext, useState } de 'react';

//Crear el contexto

exportar const AppContext = createContext();

// Crear un componente proveedor

exportar const AppProvider = ({ children }) => { const [estado,
setState] = useState({
    // Define tu estado inicial aquí
    cuenta: 0,
});

// Define cualquier acción para actualizar el estado
incremento constante = () => {
    establecerEstado((EstadoAnterior) => ({
        ...estado anterior,
        contar: prevState.count + 1,
    }));
};

constante decremento = () => {
    establecerEstado((EstadoAnterior) => ({
        ...estado anterior,
        recuento: prevState.count - 1,
    }));
};

// Proporcionar el estado y las acciones a los componentes secundarios
devolver (
    <AppContext.Provider valor={{ estado, incremento, decremento }}>
```

```
{niños}  
</AppContext.Provider>  
);  
};
```

Paso Rap tu aplicación Next.js con ppProvider

En el archivo '_app.js' del directorio 'pages', encapsula tu aplicación Next.js con el componente AppProvider del archivo AppContext.js . Por ejemplo:

```
// páginas/_app.js  
importar { AppProvider } desde '../AppContext';
```

```
función MyApp({ Componente, pageProps }) { devolver  
(  
    <Proveedor de aplicaciones>  
    <Componente {...pageProps} />  
    </Proveedor de aplicaciones>  
);  
}
```

```
exportar predeterminado MyApp;
```

Paso 4: Crea tus componentes

Cree componentes de React que usen el estado administrado por el contexto. Importe AppContext y use el gancho useContext para acceder al estado y las acciones. Por ejemplo:

```
// componentes/Counter.js  
importar React, { useContext } de 'react'; importar {
```

```
ApplicationContext } de '../ApplicationContext'; const Contador =  
() => {  
const { estado, incremento, decremento } = useContext(ApplicationContext); return (  
<div>  
  <h1>Recuento: {estado.count}</h1>  
  <button onClick={increment}>Incrementar</button>  
  <button onClick={decrement}>Decrementar</button>  
</div>  
);  
};  
exportar contador predeterminado;
```

Paso 5: Configura tus páginas Next.js

Crea una página de Next.js donde quieras renderizar tus componentes conectados al contexto. Importa los componentes y úsalos en tu página. Por ejemplo:

```
// páginas/index.js  
importar React desde 'react';  
importar Contador desde '../componentes/Contador';
```

```
const HomePage = () => { return (  
  <div>  
    Ejemplo de contexto de React  
    <Contador />  
  </div>  
);  
};
```

exportar página de inicio predeterminada;

¡Listo! Has implementado la gestión de estados mediante el contexto de React en una aplicación sencilla de Next.js. El componente AppProvider encapsula tu aplicación, proporcionando el estado y las acciones a todos los componentes anidados en ella. El componente Counter consume el estado y las acciones del contexto y los utiliza para gestionar y mostrar el recuento.

Puedes agregar más archivos de contexto, acciones y componentes según sea necesario, siguiendo los mismos patrones. El contexto de React proporciona una forma sencilla y eficiente de gestionar el estado de tu aplicación Next.js sin necesidad de dependencias externas como Redux.

Paso Acceder al contexto fuera de los componentes es opcional.

Si necesita acceder al contexto fuera de los componentes, como en funciones de utilidad o ganchos personalizados, también puede usar el gancho useContext . Por ejemplo:

```
// utilidades/Helper.js

importar { useContext } de 'react';
importar { AppContext } desde './AppContext';

const Ayudante = () => {
  const { estado, incremento, decremento } = useContext(AppContext);
  // Utilice el estado y las acciones aquí
  // ...
  devolver <div>...</div>;
};

exportar ayudante predeterminado;
```

Paso 7: Ejecuta tu aplicación Next.js

Inicie su servidor de desarrollo Next.js y navegue a la página donde usó el componente conectado al contexto. Ahora debería ver el componente contador con botones para aumentar y disminuir el contador.

¡Listo! Has implementado correctamente la gestión de estados usando el contexto de React en una aplicación sencilla de Next.js. El contexto de React te permite compartir estados.

y acciones entre componentes sin necesidad de realizar perforaciones. Simplifica la gestión y actualización del estado compartido en la aplicación.

Recuerda que, a medida que tu aplicación crece, podrías necesitar funciones más avanzadas como middleware, acciones asíncronas o una gestión de estados compleja. En estos casos, puedes considerar usar Redux u otras bibliotecas de gestión de estados.

Sin embargo, para aplicaciones más pequeñas con requisitos de estado más simples, React Context puede ser una solución liviana y efectiva.

Nota: Consulte el código de muestra del capítulo en el repositorio [itub, uildingScalableebApplicationswithNext.jsandReact](#), para explorar ejemplos prácticos e implementaciones relacionadas con el contenido del capítulo.

Casos de estudio y ejemplos

En esta sección, veremos cómo se puede implementar la gestión de estados en escenarios del mundo real en aplicaciones Next.js. Exploraremos cinco estudios de caso junto con las mejores prácticas para manejar estados complejos:

- Carrito de compras de comercio electrónico:

Escenario: Imagina que estás creando un sitio web de comercio electrónico con un Función de carrito de compras.

Implementación : Para gestionar el estado del carrito de compras, puede usar el contexto de Redux o React. Almacene los artículos, las cantidades y el precio total del carrito en el estado global. Implemente acciones y reductores para gestionar la adición, eliminación o actualización de artículos en el carrito. Conecte los componentes relevantes a la solución de gestión de estado para mostrar y actualizar el carrito.

Mejores prácticas:

- Normalizar la estructura del estado del carrito para gestionar de manera eficiente una gran cantidad de artículos.
- Optimice la renderización utilizando selectores o técnicas de memorización para evitar re-renderizaciones innecesarias de componentes relacionados con el carrito.
- Implementar la persistencia para guardar el estado del carrito, permitiendo a los usuarios regresar a su carrito incluso después de cerrar el navegador.

- Feed de redes sociales:

Escenario: Supongamos que estás creando una aplicación de redes sociales con un feed de publicaciones de diferentes usuarios.

Implementación : Para gestionar el estado del feed, puedes utilizar el estado o el contexto de React. Almacena las publicaciones y sus metadatos en el estado o contexto del componente. Implementa funciones para gestionar acciones.

Por ejemplo, añadir nuevas publicaciones, eliminarlas o actualizar los "me gusta" y los comentarios. Renderiza los componentes del feed según los datos de estado.

Mejores prácticas:

- Considerar técnicas de paginación o desplazamiento infinito para manejar de manera eficiente una gran cantidad de datos en el feed.
- Utilice acciones asíncronas o llamadas API para obtener y actualizar publicaciones, garantizando una experiencia de usuario perfecta.
- Implementar mecanismos de almacenamiento en caché para mejorar el rendimiento, evitando llamadas API redundantes para las mismas publicaciones.
- Implementación de Redux para una aplicación meteorológica:

Escenario: Desarrollo de una aplicación meteorológica que muestre el clima Información para diferentes ubicaciones.

Implementación : Usar Redux para gestionar el estado de los datos meteorológicos. Almacenar la información meteorológica, como la temperatura, las condiciones y la ubicación, en el almacén de Redux. Implementar acciones y reductores para obtener y actualizar los datos meteorológicos. Conectar los componentes relevantes al almacén mediante la biblioteca react-redux.

Mejores prácticas:

- Estructurar la tienda Redux con datos normalizados para administrar y actualizar de manera eficiente la información meteorológica.
- Utilice middleware como Redux Hunk o Redux Saga para manejar acciones asíncronas, como obtener datos meteorológicos de una API.
- Aproveche los selectores para calcular datos derivados de la tienda, reduciendo cálculos innecesarios y optimizando el rendimiento.
- Implementación del contexto React para la autenticación de usuarios:

Escenario: Creación de una aplicación con funcionalidad de autenticación de usuarios.

Implementación : Usar el contexto de React para gestionar el estado de autenticación del usuario. Almacenar la información del usuario, como el nombre de usuario y el token de autenticación, en el contexto. Implementar proveedores y consumidores de contexto para gestionar el inicio de sesión, el cierre de sesión y las comprobaciones de autenticación del usuario. Acceder al estado de autenticación en toda la aplicación mediante el gancho useContext.

Mejores prácticas:

- Utilice el contexto con moderación y solo para el estado compartido globalmente que sea verdaderamente necesario.
- Implementar la lógica de autenticación en un módulo separado o un enlace personalizado para reutilización y facilidad de mantenimiento.

- Considere usar bibliotecas de terceros como react-router para manejar rutas protegidas según el estado de autenticación.
- Implementación del estado de React para una lista de tareas pendientes:

Escenario: Creación de una aplicación de lista de tareas sencilla.

Implementación : Usa el estado de React para gestionar la lista de tareas. Almacena las tareas como un array en el estado del componente. Implementa funciones para añadir nuevas tareas, marcarlas como completadas y eliminarlas. Usa el gancho useState para gestionar el estado y actualizar la lista según las acciones del usuario.

Mejores prácticas:

- Divida el estado en partes más pequeñas y manejables para evitar re-renderizaciones innecesarias.
- Utilice inmutabilidad al actualizar el estado para garantizar un comportamiento predecible y evitar errores.
- Considere usar el gancho useeffect para conservar la lista de tareas pendientes en el almacenamiento local, permitiendo a los usuarios acceder a su lista incluso después de cerrar el navegador.

En estos casos prácticos, la implementación de la gestión de estados con contexto de Redux o React ofrece soluciones centralizadas y eficientes para gestionar estados complejos en aplicaciones Next.js. Siguiendo las mejores prácticas, como la normalización de estructuras de estados, la optimización del renderizado con selectores o memorización, y la implementación de persistencia o almacenamiento en caché, se puede garantizar un rendimiento fluido y una gestión eficaz de estados complejos.

Recuerde adaptar estos enfoques a los requisitos específicos de su aplicación y considerar las necesidades de su proyecto al implementar soluciones de gestión de estado.

Conslión

Este capítulo exploró a fondo los aspectos fundamentales de la gestión de estados en aplicaciones Next.js y enfatizó su importancia. Comenzó con una introducción a la gestión de estados en Next.js, destacando su papel crucial en el desarrollo de aplicaciones web robustas.

Se realizó una revisión exhaustiva de diversas opciones de gestión de estados en Next.js, incluyendo el estado de React, Redux y el contexto de React. Se analizaron las fortalezas y debilidades de cada opción para facilitar la toma de decisiones informadas según los requisitos específicos.

A lo largo del capítulo se hicieron hincapié en las mejores prácticas para una gestión de estados eficaz, incluyendo la prevención de errores comunes y la optimización del rendimiento. Estas directrices permiten a los desarrolladores optimizar el proceso de gestión de estados en aplicaciones Next.js.

Se exploraron técnicas de implementación para la gestión de estados usando React State, incluido el uso de ganchos como `useState` y `useContext`. Este conocimiento permite a los desarrolladores gestionar eficazmente el estado dentro de componentes.

También se abordó la implementación de la gestión de estado usando Redux, que abarca la configuración de la tienda, la definición de acciones y reductores y la integración con componentes. Este tutorial completo proporciona a los desarrolladores las habilidades necesarias para aprovechar Redux para la gestión de estado avanzada en aplicaciones Next.js.

Además, el capítulo abordó la gestión de estados mediante el contexto de React, abarcando la creación del contexto, la definición de proveedores y consumidores, y su uso dentro de los componentes. Esta comprensión permite a los desarrolladores aprovechar el contexto de React eficazmente.

Se analizaron casos prácticos y ejemplos del mundo real para reforzar la comprensión y demostrar implementaciones prácticas de la gestión de estados en aplicaciones Next.js. Estos escenarios mostraron las mejores prácticas para gestionar estados complejos, aportando información valiosa a los desarrolladores.

El siguiente capítulo se centrará en la introducción de RS y raph Pls en el contexto de las aplicaciones Next.js. Se explicarán sus conceptos y se mostrará su uso para crear aplicaciones web robustas y flexibles con Next.js.

pequeñas cuestiones de la oie

1. ¿Cuál es el propósito de la gestión de estados en una aplicación Next.js?
 - A. Para gestionar la autenticación del usuario
 - B. Para gestionar el enrutamiento de aplicaciones
 - C. o gestionar y compartir datos de manera eficiente entre componentes
 - D. Para mejorar la organización y mantenibilidad del código
2. ¿Qué opción de gestión de estado está integrada y es proporcionada por el propio React?
 - A. Redux
 - B. Estado de reacción

- C. Contexto de React
- D. MobX
3. ¿Qué opción de gestión de estado se basa en el patrón de arquitectura Flux?
- A. Redux
- B. Estado de reacción
- C. Contexto de React
- D. MobX
4. ¿Cuál es la principal ventaja de usar el contexto de React para la gestión de estados?
- A. Gestión centralizada del estado con un flujo de datos predecible
- B. Soporte integrado para la depuración de viajes en el tiempo
- C. Simplicidad y facilidad de uso
- D. Alto rendimiento y actualizaciones eficientes
5. ¿Cuándo se suele recomendar Redux como una opción de gestión de estados?
- ¿Sigiente.js?
- A. Para aplicaciones pequeñas y sencillas.
- B. al tratar con estados complejos y flujos de datos
- C. La renderización del lado del servidor es un requisito
- D. cuando se necesitan actualizaciones de datos en tiempo real

Respuestas

1. c	
2. b y c	
3. un	
4. c	
5.b	

Nota: Estas preguntas están diseñadas para evaluar los conocimientos generales sobre la gestión de estados en Next.js. Es importante consultar el contenido y los conceptos específicos tratados en el material de aprendizaje para obtener respuestas precisas.

Capítulo 8

Restful y GraphQL Implementación de API

Introducción

En el mundo del desarrollo web, las API (Interfaces de Programación de Aplicaciones) son esenciales para crear aplicaciones robustas. Facilitan la comunicación y el intercambio de datos entre sistemas de software. Este capítulo explora la importancia de las API en el desarrollo web moderno.

En este capítulo, presentaremos las API y destacaremos su importancia como puentes entre aplicaciones, proporcionando acceso a funcionalidades externas. Analizaremos las diferencias entre las API RESTful y GraphQL, y sus beneficios para un diseño de API bien fundamentado. La implementación práctica abarca la configuración de API RESTful en Next.js y la creación de endpoints para operaciones CRUD. También exploraremos la configuración de API GraphQL en Next.js con Apollo Server, creando potentes endpoints de API para una recuperación de datos precisa.

La integración con aplicaciones Next.js se centrará en la renderización del lado del servidor, mejorando la experiencia del usuario con cargas de página más rápidas. También se abordarán la gestión de errores y las mejores prácticas para la seguridad y autenticación de API.

A lo largo del capítulo, se incluyen ejemplos prácticos y consejos que enriquecen la comprensión del lector. Al finalizar, comprenderá a fondo las API en el desarrollo con Next.js. ¡Desbloqueemos el poder de las API en el desarrollo web!

Estructura

En este capítulo cubriremos los siguientes temas:

- Introducción a las API y su importancia en el desarrollo web moderno
- Comprender las diferencias entre las API RESTful y GraphQL

Configuración de un RSful PI en Next.js

Configuración de un marco de trabajo raph como PI en Next.js usando popular Apollo Server

Creación de puntos finales de API RESTful y GraphQL para operaciones CRUD (Crear, Leer, Actualizar, Eliminar)

- Integración de los puntos finales de la API con Next.js • Manejo de errores y excepciones en llamadas API
- Mejores prácticas para la seguridad y autenticación de API en aplicaciones Next.js

Introducción a las API y su importancia

en el desarrollo web moderno

PI puede considerarse un conjunto de reglas y protocolos que define cómo deben interactuar los componentes de software. Especifica los métodos, formatos de datos y puntos finales que los desarrolladores pueden usar para solicitar e intercambiar información entre aplicaciones.

Tipos de API:

- API web: Estas API están diseñadas para permitir la comunicación entre sistemas web. Utilizan HTTP (Protocolo de Transferencia de Hipertexto) como protocolo subyacente y suelen proporcionar datos en formatos como JSON (Notación de Objetos JavaScript) o XML (Lenguaje de Marcado Extensible).
- API de servicio: Estas API son expuestas por diversos servicios de software, lo que permite a los desarrolladores acceder a funcionalidades específicas de dichos servicios. Por ejemplo, un servicio de pasarela de pagos puede proporcionar una API que los desarrolladores pueden usar para procesar pagos en sus aplicaciones.

Importancia de las API en el desarrollo web moderno:

- Integración: Las API permiten que diferentes aplicaciones se integren y comparten datos, funcionalidades y servicios sin problemas. Esta integración permite a los desarrolladores aprovechar las plataformas y servicios existentes para optimizar sus aplicaciones rápidamente.
- Reutilización: Las API promueven la reutilización y la modularidad del código. En lugar de crear todo desde cero, los desarrolladores pueden usar las API para acceder a funcionalidades predefinidas, lo que ahorra tiempo y esfuerzo.
- Escalabilidad: Las API facilitan la escalabilidad de las aplicaciones. Los desarrolladores pueden aprovechar servicios o plataformas externas para gestionar tareas como el almacenamiento, la autenticación o el procesamiento de pagos, liberando así recursos para centrarse en las funciones principales de la aplicación.
- Colaboración: Las API promueven la colaboración entre desarrolladores y

Equipos. Al exponer funcionalidades específicas mediante PI, los desarrolladores pueden trabajar en diferentes componentes simultáneamente, lo que garantiza ciclos de desarrollo más rápidos.

¿Cómo funcionan las API?

Cuando un desarrollador desea interactuar con una PI, realiza una solicitud a un punto final específico (URL) mediante el método HTTP adecuado (GET, POST, PUT, DELETE, etc.). La API procesa la solicitud, realiza las acciones necesarias y devuelve una respuesta con los datos solicitados o un estado que indica si la operación se ha realizado correctamente o no.

Protocolos y arquitecturas API

Los protocolos y arquitecturas de API se refieren a los estándares y marcos que rigen el diseño, la comunicación y los patrones de interacción de las API. A continuación, se presentan algunos protocolos y arquitecturas de API de uso común:

- REST (Transferencia de Estado Representacional): REST es un estilo arquitectónico ampliamente adoptado para el diseño de aplicaciones en red. Las API RESTful utilizan métodos HTTP (GET, POST, PUT, DELETE, etc.) para interactuar con los recursos identificados por los localizadores de recursos uniformes de RS. Las API RS priorizan la comunicación sin estado, la escalabilidad y la separación de intereses. Se utilizan comúnmente en el desarrollo web y siguen principios como el diseño orientado a recursos, la ausencia de estado y la interfaz uniforme.
- SOAP (Protocolo Simple de Acceso a Objetos): SOAP es un protocolo para el intercambio de información estructurada en servicios web mediante . Los PI de SP definen formatos de mensajes, operaciones y contratos mediante contratos basados en XML descritos por WSDL (Lenguaje de Descripción de Servicios Web). SOAP es conocido por sus amplias funciones, compatibilidad con tipos de datos complejos, seguridad y capacidades transaccionales. Se utiliza comúnmente en integraciones empresariales y arquitecturas orientadas a servicios.
- gRPC (Llamada a Procedimiento Remoto de Google): gRPC es un framework de alto rendimiento, independiente del lenguaje, desarrollado por Google. Utiliza búferes de protocolo (protobuf) para definir servicios y mensajes. Los PI de gRPC permiten una comunicación eficiente y multiplataforma entre servicios mediante llamadas a procedimiento remoto (RPC). gRPC es conocido por su velocidad, compatibilidad con streaming bidireccional, control de flujo y autenticación. Se utiliza frecuentemente en arquitecturas de microservicios y sistemas distribuidos.
- GraphQL: GraphQL es un lenguaje de consulta y entorno de ejecución de código abierto para investigadores principales. Permite a los clientes solicitar datos específicos mediante la construcción de consultas flexibles y precisas. A diferencia de REST, donde el servidor determina la forma de...

En respuesta, GraphQL pone el control en manos del cliente. Las API de GraphQL proporcionan un único punto final y devuelven únicamente los datos solicitados por el cliente, lo que reduce la sobrecaptura y la subcaptura de datos. GraphQL se utiliza comúnmente en escenarios donde la flexibilidad, la eficiencia en la recuperación de datos y los requisitos de datos definidos por el cliente son esenciales.

Estos son algunos de los protocolos y arquitecturas de API más utilizados. Cada protocolo tiene sus propias ventajas y casos de uso, y la elección depende de factores como la naturaleza de la aplicación, los requisitos de rendimiento, la compatibilidad con los sistemas existentes y las preferencias del desarrollador.

En este capítulo, comprenderá a fondo estos dos tipos de API populares: REST y GraphQL. Analizaremos sus características, casos de uso y cómo implementar y utilizar API de cada tipo. Este conocimiento le permitirá tomar decisiones informadas al elegir entre REST y GraphQL según los requisitos de su proyecto y optimizar sus estrategias de diseño e integración de API.

API RESTful versus API GraphQL

La siguiente tabla explica la diferencia entre las API RESTful y GraphQL:

	API RESTful	API de GraphQL
Obtención de datos	Obtiene múltiples recursos utilizando múltiples puntos finales	Obtiene datos de un único punto final, lo que permite a los clientes especificar exactamente qué datos necesitan
Exceso de obtención de ingresos	Ocurrencia común donde los clientes reciben más datos de los necesarios	Elimina la búsqueda excesiva al permitir que los clientes soliciten solo los campos obligatorios
Subestimación	Requiere múltiples solicitudes para recuperar datos relacionados	Resuelve el problema de obtención insuficiente al permitir que los clientes soliciten datos anidados en una sola consulta
Control de versiones	A menudo requiere control de versiones para introducir cambios importantes	No es necesario crear versiones, ya que el cliente puede controlar la estructura de la respuesta.
Punto final Estructura	Basado en recursos, con diferentes puntos finales para diferentes recursos	Punto final único que maneja todas las consultas de datos, mutaciones y suscripciones

Pedido Flexibilidad	Flexibilidad limitada ya que el servidor determina el formato de respuesta	Altamente flexible, ya que el cliente especifica los requisitos de datos en la consulta.
Almacenamiento en caché	Admite almacenamiento en caché en el nivel de punto final	El almacenamiento en caché puede ser un desafío debido a consultas dinámicas y requisitos de datos impredecibles.
Complejidad Gestión	Puede volverse complejo al tratar con recursos anidados o relacionados	Proporciona simplicidad en la consulta de datos relacionados y evita la complejidad mediante una selección precisa de datos.
Cliente-Servidor Acuerdo	El servidor determina la estructura y el formato de respuestas	El cliente decide la estructura y el formato de las respuestas especificando los campos que necesita
Aprendiendo Curva	Relativamente más fácil de entender e implementar	Requiere aprendizaje y comprensión adicionales del lenguaje de consulta GraphQL y las funciones de resolución.
Ecosistema Madurez	Bien establecido con un ecosistema y herramientas maduros	Ecosistema en crecimiento con herramientas en evolución, pero no tan maduro como RESTful API

Tabla 8.1: La diferencia entre las API RESTful y GraphQL

Configurar y configurar Next.js

yo

en

Para configurar un RSful PI en Next.js con diferentes métodos P (GET, POST, PUT, PATCH y DELETE), siga los pasos que se proporcionan aquí:

Paso 1: Configurar un nuevo proyecto Next.js

- Cree un nuevo directorio para su proyecto y navegue hasta él usando una terminal o un símbolo del sistema.
- Inicialice un nuevo proyecto Next.js ejecutando el siguiente comando:

```
npx crear-próxima-aplicación
```

Esto configurará un proyecto básico de Next.js en el directorio actual.

Paso 2: Crear una ruta API

- Dentro del directorio de su proyecto, cree un nuevo directorio llamado `pages/api`. • En el directorio api, cree un nuevo archivo avaScript, como example.js.
Este archivo representará su ruta PI.

Paso 3: Definir rutas PI para diferentes

Métodos P

1. Método GET: recupera datos de un servidor.

```
En example.js, defina la ruta PI para el método. Por ejemplo
exportar función predeterminada manejador(req, res) {
    si (req.método === 'GET') {
        // Lógica para manejar solicitudes GET
        res.status(200).json({ mensaje: 'Solicitud GET recibida.' });
    }
    } demás {
        // Manejar otros métodos HTTP
        res.status(405).json({ message: 'Método no permitido' });
    }
}
```

2. Método POST: envía datos a un servidor para crear un nuevo recurso.

Para manejar una solicitud POST, agregue el siguiente código a `example.js`:

```
exportar función predeterminada manejador(req, res) {
    si (req.método === 'POST') {
        // Lógica para manejar solicitudes POST
        const datos = req.cuerpo;
        res.status(200).json({ mensaje: 'Solicitud POST recibida.', datos });

    } demás {
        // Manejar otros métodos HTTP
        res.status(405).json({ message: 'Método no permitido' });
    }
}
```

Nota: Para el método PUT, es necesario analizar el cuerpo de la solicitud para acceder a los datos. Para ello, puede usar 'body-parser' o el middleware express.json() integrado .

3. Método PUT: envía datos a un servidor para actualizar un recurso existente.

Para el método P, puede utilizar el siguiente código

```
exportar función predeterminada manejador(req, res) {  
    si (req.método === 'PUT') {  
        // Lógica para manejar solicitudes PUT  
        const { id } = req.query; // Acceder a URL dinámica  
        parámetro  
        const datos = req.cuerpo;  
        res.status(200).json({ mensaje: 'Solicitud PUT recibida  
        para ID: ${id}', datos });  
    } demás {  
        // Manejar otros métodos HTTP  
        res.status(405).json({ message: 'Método no permitido' });  
    }  
}
```

En este ejemplo, el método PUT espera un parámetro de URL dinámico 'id' para identificar el recurso que se está actualizando.

PARCHE: Envía datos a un servidor para actualizar parcialmente un recurso existente, modificando sólo los campos o propiedades especificados sin afectar el resto del recurso.

La principal diferencia entre los métodos PUT y PATCH es la siguiente:

- PUT: El método PUT se utiliza para reemplazar completamente un recurso existente con una nueva representación. Al enviar una solicitud PUT, el recurso actualizado completo se envía generalmente en la carga útil de la solicitud. Esto significa que cualquier campo no incluido en la carga útil de la solicitud se eliminará o se restablecerá a sus valores predeterminados. PUT se utiliza generalmente para actualizaciones completas y es idempotente, lo que significa que realizar la misma solicitud PUT varias veces tendrá el mismo resultado.
- PATCH: El método PATCH se utiliza para actualizar parcialmente un recurso existente. En lugar de enviar el recurso completo, una solicitud PATCH suele incluir solo los campos o propiedades específicos que deben modificarse. Esto permite actualizaciones más granulares sin afectar...

El resto del recurso. PATCH se utiliza para actualizaciones parciales y no es necesariamente idempotente, lo que significa que realizar la misma solicitud PATCH varias veces puede tener resultados diferentes.

En resumen, PUT se utiliza para actualizaciones completas reemplazando todo el recurso, mientras que PC se utiliza para actualizaciones parciales modificando campos o propiedades específicos de un recurso existente.

- D eto ens un rest a un serer para eliminar un seif i
recurso.

Para la D método, puedes utilizar el siguiente código

```
exportar función predeterminada manejador(req, res) {  
    si (req.método === 'BORRAR') {  
        // Lógica para manejar solicitudes DELETE  
        const { id } = req.query; // Acceder al parámetro URL dinámico  
        res.status(200).json({ mensaje: `Solicitud DELETE recibida  
para
```

```
Identificación: ${id}`});
```

```
} demás {
```

```
    // Manejar otros métodos HTTP
```

```
    res.status(405).json({ message: 'Método no permitido' });
```

```
}
```

```
}
```

En este ejemplo, el método DELETE espera un parámetro de URL dinámico `id` para identificar el recurso que se está eliminando.

Paso 4: Probar las rutas API

Inicie el servidor de desarrollo Next.js ejecutando el siguiente comando en el directorio de su proyecto:

```
npm ejecutar dev
```

Utilice una herramienta como Postman o cURL para enviar solicitudes a sus rutas API.

Por ejemplo, puede enviar una solicitud a `http://localhost/api/` (ver Figura 8.1).`



Figura 8.1: Llamada a la API de llamada de Postman

Configurar y configurar un Next.js en usando Apollo Server

En una API GraphQL, los tres componentes básicos incluyen consultas, mutaciones y solucionadores.

Consulta : Una consulta en GraphQL se utiliza para solicitar datos del servidor. Sigue una estructura específica definida por el esquema de GraphQL y permite a los clientes especificar los campos y relaciones exactos que desean recuperar. Las consultas se ejecutan en paralelo y la respuesta se ajusta a la forma de la consulta.

- **Mutación:** una mutación en GraphQL se utiliza para modificar datos en el servidor. Permite a los clientes crear, actualizar o eliminar recursos. Al igual que las consultas, las mutaciones se definen en el esquema de GraphQL y tienen su propia estructura. Las mutaciones se utilizan normalmente al realizar cambios que tienen efectos secundarios en el servidor.
- **Resolver:** Un resolver es una función responsable de resolver una consulta o mutación en GraphQL. Actúa como enlace entre la API y las fuentes de datos subyacentes. Los resolvers se definen para cada campo del esquema y contienen la lógica para obtener o manipular los datos solicitados. Se ejecutan para atender la solicitud del cliente y devolver los datos correspondientes.

En general, las consultas se utilizan para recuperar datos, las mutaciones se utilizan para modificar datos y los resolutores manejan la ejecución de estas solicitudes y proporcionan los datos solicitados o realizan las acciones necesarias.

Para configurar un raph siga estos pasos: PI en Next.js usando pollo Server, siga

Paso 1: Configurar un nuevo proyecto Next.js

- Cree un nuevo directorio para su proyecto y navegue hasta él usando una terminal o un símbolo del sistema.

- Inicialice un nuevo proyecto Next.js ejecutando el siguiente comando:

```
npx crear-próxima-aplicación
```

Esto configurará un proyecto básico de Next.js en el directorio actual.

Paso 2: Instalar las dependencias necesarias

- En el directorio de su proyecto, instale los paquetes necesarios para GraphQL y Apollo Server ejecutando el siguiente comando:

```
npm instalar apollo-server-micro graphql
```

Paso 3: Crear una ruta API para GraphQL

- Dentro del directorio de su proyecto, cree un nuevo directorio llamado `pages/api`.
- En el directorio api, cree un nuevo archivo avaScript, como graphql.js. Este archivo representará su gráfico.

Paso 4: Definir el esquema raph y los solucionadores

En graphql.js, defina el esquema de raph mediante la función makeExecutableSchema del paquete graphql-tools . Por ejemplo:

```
importar { ApolloServer, gql } desde 'apollo-server-micro';
constante typeDefs = gql'
  tipo Libro {
    ¡hice!
    Título: ¡Cuerda!
    autor: ¡Cuerda!
  }
  tipo Consulta {
    libros:[¡Libro!]!
    libro(id: ID!): Libro
  }
  tipo Mutación {
    addBook(título: String!, autor: String!): ¡Libro!
    updateBook(id: ID!, título: String, autor: String): Libro
    deleteBook(id: ID!): Libro
  }
';
'
```

Implementación de API Restful y GraphQL

```

const libros = [
  { id: '1', título: 'Libro 1', autor: 'Autor 1' },
  { id: '2', título: 'Libro 2', autor: 'Autor 2' },
];
resolvers constantes = {
  Consulta: {
    libros: () => libros,
    libro: (padre, args) => libros.find((libro) => libro.id === args.id),
  },
  Mutación: {
    addBook: (padre, argumentos) => {
      const newBook = { id: String(libros.length + 1), ...args };
      libros.push(nuevoLibro);
      devolver nuevoLibro;
    },
    updateBook: (padre, argumentos) => {
      const index = libros.findIndex((libro) => libro.id === args.
        identificación);
      si (índice >= 0) {
        const updatedBook = { ...libros[índice], ...args };
        libros[índice] = updatedBook;
        devolver updatedBook;
      }
      devuelve nulo;
    },
    deleteBook: (padre, argumentos) => {
      const index = libros.findIndex((libro) => libro.id === args.
        identificación);
      si (índice >= 0) {
        const removedBook = libros[índice];
        libros.splice(index, 1);
        devolver removedBook;
      }
    }
  }
};

```

```
        libros.splice(índice, 1);
        devolver libroeliminado;
    }
    devuelve nulo;
},
},
};

const apolloServer = nuevo ApolloServer({ typeDefs, resolvers });
dejar manejador;
función asíncrona getHandler() {
    si (!controlador) {
        esperar apolloServer.start();
        manejador = apolloServer.createHandler({ ruta: '/api/graphql' });
    }
    manejador de retorno ;
}

exportar función asíncrona predeterminada graphqlHandler(req, res) {
    const apolloHandler = await getHandler(); return
    apolloHandler(req, res);
}

exportar const config = {
    api: {
        bodyParser: falso,
    },
};

1. Importación de las dependencias necesarias:
a. `ApolloServer` y `gql` se importan desde `apollo-server-micro`.
```

`ApolloServer` se utiliza para crear una instancia del servidor Apollo y gql se utiliza para definir el esquema raph.

Definición del esquema raph

- a. El esquema se define mediante la función de etiqueta gql . Especifica tipos y operaciones disponibles en la API.
- b. En este esquema, hay tres tipos: «Libro», «Consulta» y «Mutación».
- c. El tipo `Libro` representa un objeto de libro y tiene tres campos: ``ID` (tipo ID), ``título` (tipo String) y ``autor` (tipo String).
- d. El tipo de consulta define las consultas disponibles. Tiene dos campos: libros (devuelve una matriz de objetos "Libro") y libro (toma un argumento "id" y devuelve un único objeto "Libro").
- e. El tipo `Mutación` define las mutaciones disponibles. Tiene tres campos: `addBook` (toma los argumentos `title` y `author` y devuelve un nuevo objeto `Book`), `updateBook` (toma los argumentos `id` , `title` y `author` y devuelve el objeto `Book` actualizado) y `deleteBook` (toma el argumento `id` y devuelve el objeto `Book` eliminado).

Definición de las funciones de resolución

- a. La matriz `book` contiene algunos objetos de libro como datos de muestra.
- b. Las funciones de resolución definen cómo se ejecutan las operaciones de raph. resuelto.
- c. Para el tipo Consulta , las funciones de resolución para `libros` y `libro` simplemente devuelven los datos correspondientes de la matriz `libros` en función de los argumentos proporcionados.

Para el tipo Mutation , las funciones de resolución para `addBook` , `updateBook` y `deleteBook` gestionan la adición, actualización y eliminación de datos de libros, respectivamente. Actualizan la matriz `books` según corresponda y devuelven los datos relevantes.

4. Creación de una instancia de Apollo Server:

- a. Se crea una instancia de `ApolloServer` con los typeDefs` y `resolvers` definidos.

5. Configuración del controlador de funciones sin servidor:

- a. La función `getHandler` está definida para crear y devolver de forma diferida el Controlador del servidor Apollo.

- b. La función `graphqlHandler` es el punto de entrada principal para la función sin servidor.
- c. Llama a `getHandler` para obtener el controlador del servidor Apollo y pasa el objetos `req` y `res` entrantes .
- d. Luego se devuelve el controlador.

6. Exportar la función y configuración sin servidor

- a. La función `graphqlHandler` se exporta como función predeterminada a se utilizará como función sin servidor.
- b. El `co a` El objeto se exporta para configurar el PI, estableciendo `bodyParser` `false` para deshabilitar el análisis automático del cuerpo de la solicitud.

7. En general, este código configura una API GraphQL utilizando Apollo Server Micro, define un esquema con consultas y mutaciones para administrar una lista de libros y proporciona funciones de resolución para manejar la lógica de consulta, adición, actualización y eliminación de datos de libros.

Paso 5: Pruebe la API GraphQL

1. Inicie el servidor de desarrollo Next.js ejecutando el siguiente comando en el directorio de su proyecto:

npm ejecutar dev

Abra su navegador y navegue a `http://localhost:port/graphql`.
Deberías ver la interfaz GraphQL Playground.

Utilice la interfaz de Playground para enviar consultas y mutaciones a su PI. Por ejemplo, puede enviar las siguientes consultas y mutaciones de raph:

Consulta: Obtener todos los libros

```
consulta {
```

```
    libros {
```

```
        título
```

```
        autor
```

```
}
```

```
}
```

Mutación: Agregar un nuevo libro

```

mutación {
    addBook(título: "El gran Gatsby", autor: "F.
    Scott Fitzgerald")
}

    título
    autor
}
}
```

Mutación: Actualizar un libro

```

mutación {
    updateBook(id: "bookId", title: "Nuevo título",
    autor: "Nuevo Autor"
)

    título
    autor
}
}
```

Reemplace `bookId` con el ID real del libro que desea actualizar.

Mutación: Eliminar un libro

```

mutación {
    eliminarLibro(id: "bookId") {

        título
        autor
    }
}
```

Reemplace `bookId` con el ID real del libro que desea eliminar.

Puede utilizar las consultas y mutaciones anteriores en la interfaz GraphQL Playground para interactuar con su API GraphQL y realizar acciones como agregar, actualizar y eliminar libros (consulte las Figuras 8.2 y 8.3).

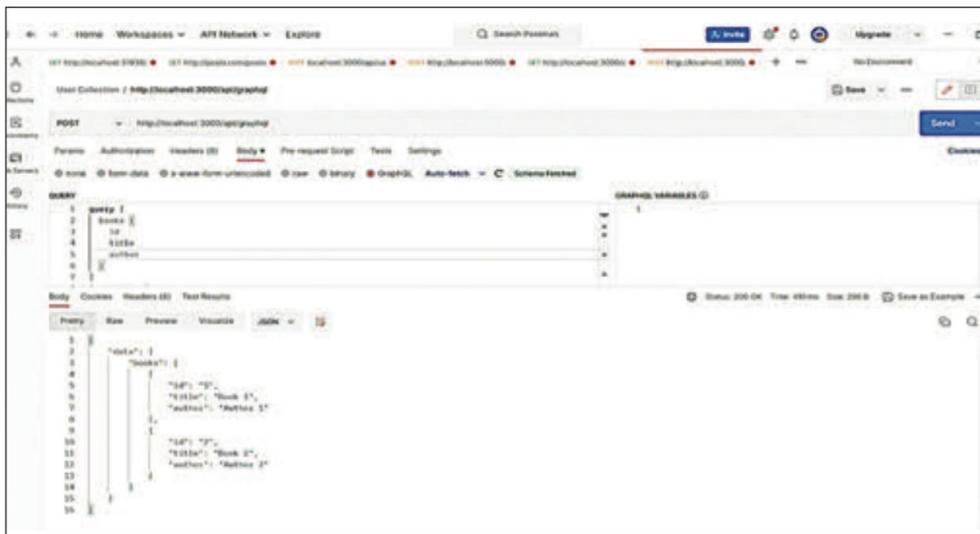


Figura 8.2: Obtener datos de la API GraphQL

La figura 8.3 muestra los datos agregados desde la API GraphQL:

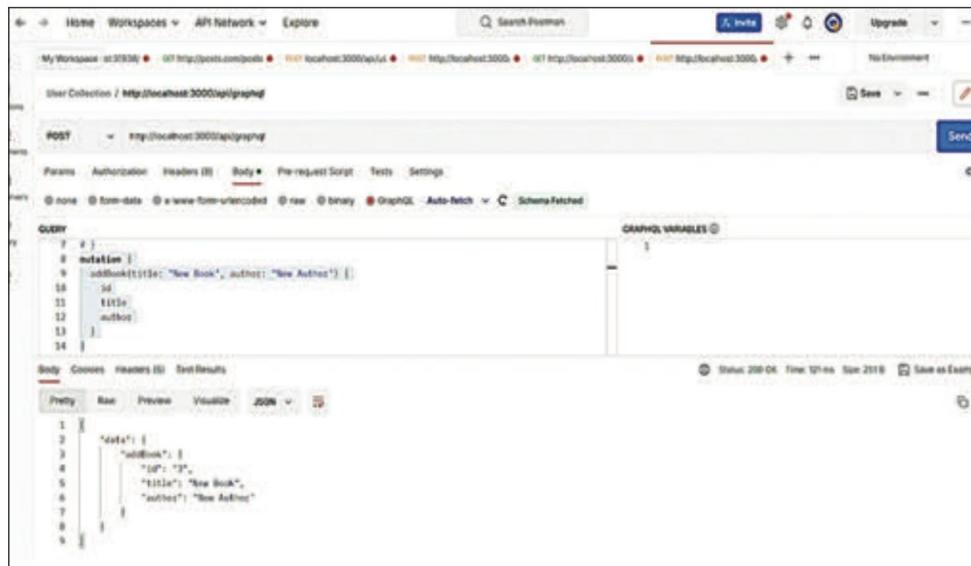


Figura 8.3: Agregar datos desde la API GraphQL

Por convención, las consultas GraphQL deben enviarse como solicitudes POST, ya que la complejidad de la consulta y los requisitos de datos pueden variar considerablemente. Las solicitudes POST también permiten enviar cargas útiles más grandes y estructuras de consulta más complejas.

Sin embargo, GraphQL también admite solicitudes GET para consultas simples que solo recuperan datos sin efectos secundarios. En una solicitud GET, la consulta se suele pasar como parámetro de URL.

o uso Para realizar solicitudes con GraphQL, debe modificar la configuración del servidor y la implementación del cliente en consecuencia. En el lado del servidor, deberá configurar su servidor GraphQL para aceptar solicitudes GET y manejarlas adecuadamente.

En el lado del cliente, construiría la consulta GraphQL como un parámetro de URL y la incluiría en la solicitud GET.

Es importante tener en cuenta que, por lo general, no se recomienda el uso de solicitudes GET para consultas complejas o de mutación debido a limitaciones en el tamaño de la solicitud y posibles problemas de seguridad. Las solicitudes POST son el método recomendado para la mayoría de las interacciones GraphQL.

Entonces, si bien es cierto que las consultas GraphQL pueden funcionar con métodos POST y GET, es recomendable utilizar solicitudes POST para la mayoría de las operaciones GraphQL y solo utilizar solicitudes GET para consultas simples de solo lectura cuando sea necesario.

Integración de los puntos finales de la API con el lado del cliente en Next.js

Antes de integrar las API REST con Next.js, debe comprender cómo usar el gancho SWR en Next.js. Además, para integrar las API GraphQL, puede utilizar la biblioteca cliente Apollo.

Permitíame explicar cada biblioteca por separado, junto con sus propiedades, mediante un breve ejemplo de código.

El gancho SWR (Stale-While-Revalidate) de Next.js es una biblioteca integrada de obtención de datos que simplifica la obtención, el almacenamiento en caché y la sincronización entre el cliente y el servidor. Proporciona una forma intuitiva de gestionar las actualizaciones de datos en tiempo real y optimizar el rendimiento de la aplicación. A continuación, se detallan las propiedades del gancho SWR y su uso:

Cadena de clave useSWR , función de obtención, opciones? Configuración. Esta es la función principal del gancho SWR para obtener y gestionar datos. Acepta tres parámetros:

- Identificador único de clave para los datos que se obtienen. Puede ser un R, un punto final de PI o cualquier otra cadena que represente la fuente de datos.
- fetcher: una función responsable de realizar la llamada API real y devolverla

La respuesta. La función de obtención puede ser asíncrona y, por lo general, debería devolver los datos obtenidos en un formato específico, como JSON.

- Objeto no opcional que permite configurar comportamientos y opciones adicionales para el gancho SWR. Incluye propiedades como `revalidateCount`, `revalidateFocus`, `revalidateReconnect`, `refreshInterval` y más.

A continuación se muestra un ejemplo que demuestra el uso del gancho SWR:

```
importar useSWR desde 'swr';

const fetcher = (url) => fetch(url).then((res) => res.
json());

constante MiComponente = () => {
    const { datos, error } = useSWR('/api/data', fetcher);
    si (error) devuelve <div>Error al cargar datos</div>;
    si (!data) devuelve <div>Cargando...</div>;
    devolver <div>{datos.mensaje}</div>;
};


```

En el ejemplo anterior, usamos `useSWR` para obtener datos del punto final `/api/data` mediante la función `fetcher`. Las variables `data` y `error` capturan el resultado de la llamada a la API, que se puede renderizar en consecuencia en el componente.

- `datos`: La propiedad `datos` contiene los datos obtenidos. Inicialmente, no estará definida o tendrá el valor predeterminado especificado en el gancho `useSWR`.
- `Error`: Si se produce un error durante la obtención de datos, la propiedad "error" contendrá los detalles del error. Puede activar esta propiedad para gestionar los casos de error adecuadamente.

Además, el gancho proporciona las siguientes funciones:

- `mutate(data?, shouldRevalidate?, options?)`: La función `mutate` permite mutar manualmente los datos. Puede usarla para actualizarlos o activar una revalidación. Acepta tres parámetros opcionales.
 - `datos`: Los nuevos datos que se configurarán. Si se proporciona, se actualizarán.
 - `propiedad`.
 - `shouldRevalidate`: Un valor booleano que indica si se debe activar una revalidación tras la mutación de datos. Por defecto, es verdadero.
 - `opciones`: Opciones adicionales para la mutación, como `revalidate` y `noRevalidate`.
- `revalidate()`: La función `revalidate` activa una revalidación de los datos, realizando una nueva llamada API para obtener los datos más recientes.

- `isValidating`: La propiedad `isValidating` es un valor booleano que indica si el gancho SWR está validando (obteniendo) los datos. Puede usarse para mostrar indicadores de carga u otros indicadores mientras se obtienen los datos.

Estas propiedades y funciones proporcionan un control poderoso sobre la obtención y actualización de datos en su aplicación Next.js.

Cliente Apollo: El Cliente Apollo es un potente cliente de raph que simplifica la obtención y gestión de datos en aplicaciones Next.js. Ofrece funciones como almacenamiento en caché, gestión de estado local y suscripciones en tiempo real. A continuación, se detallan las propiedades del Cliente Apollo y su uso:

1. Configuración del cliente Apollo en Next.js:

Para utilizar Apollo Client en Next.js, necesitará instalar los paquetes necesarios:

```
npm install @apollo/client graphql
```

Una vez instalado, puede crear una instancia de Apollo Client con la configuración requerida, normalmente en un archivo separado

```
importar { ApolloClient, InMemoryCache } desde '@apollo/client';

const cliente = nuevo ApolloClient({
  uri: 'https://localhost:3000.com/graphql', // reemplazar
  con su punto final de API GraphQL
  caché: nuevo InMemoryCache(),
});

exportar cliente predeterminado;
```

En el ejemplo anterior, creamos una instancia de Apollo Client con el punto de conexión de la API GraphQL (`'uri'`) adecuado y un `'InMemoryCache'` para el almacenamiento en caché de datos. Puede personalizar la configuración según sus necesidades.

2. Proveedor Apollo en Next.js:

Next.js proporciona el componente `'ApolloProvider'` para encapsular su aplicación y proporcionar la instancia de Apollo Client a todos los componentes de la jerarquía. Normalmente puedes configurarlo en tu archivo `pages/app.js`

```
importar { ApolloProvider } desde '@apollo/client';
importar cliente desde '../path/to/apolloClient';
```

```

función MyApp({ Componente, pageProps }) {
    devolver (
        <ApolloProvider cliente={cliente}>
            <Componente {...pageProps} />
        </ProveedorApolo>
    );
}

exportar predeterminado MyApp;

```

Al envolver su aplicación con `ApolloProvider` y pasar la instancia `client`, todos los componentes de su aplicación pueden acceder y utilizar Apollo Client para la obtención y gestión de datos.

3. Consulta de datos con Apollo Client:

Para obtener datos de una API GraphQL con Apollo Client, puedes usar el gancho `useQuery` proporcionado por `@apollo/client` . Aquí tienes un ejemplo:

```

importar { gql, useQuery } desde '@apollo/client';
constante GET_USERS = gql'
    consulta {
        usuarios {
            identificación
            nombre
        }
    }
';

función MiComponente() {
    const { cargando, error, datos } = useQuery(GET_USERS); if (cargando)
        return <div>Cargando...</div>; if (error) return <div>Error
        al cargar datos</div>;
    devolver (
        <ul>
            {datos.usuarios.map((usuario) => (
                <li key={usuario.id}>{usuario.nombre}</li>
            )))
    )
}

```

Implementación de API Restful y GraphQL

```
</ul>
);
}
```

En el ejemplo anterior, definimos una consulta raph usando la etiqueta gql de @apollo/client. El gancho `useQuery` se usa para obtener datos según la consulta `GET_USERS`. La propiedad `loading` indica si los datos aún se están obteniendo, la propiedad `error` captura cualquier error que ocurra durante el proceso y la propiedad `data` contiene los datos obtenidos.

4. Mutación con el cliente Apollo:

El cliente Apollo proporciona el gancho `useMutation` para manejar mutaciones (modificaciones de datos en raph). Aquí hay un ejemplo

```
importar { gql, useMutation } de '@apollo/client';
constante ADD_USER = gql'
mutación AddUser($input: UserInput!) {
    addUser(entrada: $entrada) {
        ...añadirUsuario
        nombre
    }
}
función MiComponente() {
    const [addUser, { cargando, error, datos }] = useMutation(ADD_USUARIO);
    constante handleAddUser = async () => {
        intentar {
            const {datos} = esperar agregarUsuario({
                variables: {
                    aporte: {
                        nombre: "John Doe",
                    },
                },
            },
        },
    },
}
```

```
});  
    consola.log(datos);  
} captura (error) {  
    consola.error(error);  
}  
};  
si (cargando) devuelve <div>Cargando...</div>;  
si (error) devuelve <div>Error al agregar usuario</div>;  
devolver (  
    <div>  
        <button onClick={handleAddUser}>Agregar usuario</button>  
    </div>  
);  
}
```

En el ejemplo anterior, definimos una mutación raph usando la etiqueta gql de @apollo/client. El gancho useMutation se usa para gestionar la mutación ADD_USER. Devuelve una función addUser que puede llamarse para activar la mutación. El resultado de la mutación, incluyendo la carga, el error y los datos, se captura en las variables correspondientes.

En la función handleAddUser , llamamos a la función addUser y proporcionamos las variables necesarias para la mutación. En este caso, añadimos un usuario llamado "John Doe". Se puede acceder a los datos de respuesta y utilizarlos según corresponda.

5. Caché del cliente Apollo:

El Cliente Apollo utiliza una caché para almacenar y gestionar los datos recuperados. Esta caché permite una recuperación y sincronización eficientes de datos. De forma predeterminada, el Cliente Apollo utiliza la implementación InMemoryCache , que ofrece potentes capacidades de almacenamiento en caché.

Puede personalizar el comportamiento de la caché según sus necesidades. Por ejemplo, puede modificar las opciones de caché, definir solucionadores de caché personalizados o incluso implementar la manipulación de datos del lado del cliente.

Estos son algunos de los aspectos clave del uso de Apollo Client en Next.js. Apollo Client ofrece muchas más funciones y capacidades, como la gestión de estados locales, suscripciones y gestión de errores. Proporciona una solución integral para

trabajar con GraphQL en sus aplicaciones Next.js, simplificando la obtención, el almacenamiento en caché y la manipulación de datos.

Para integrar los puntos finales de la API REST en una aplicación Next.js utilizando el gancho SWR (Stale-While-Revalidate), puede seguir estos pasos:

1. Instale las dependencias necesarias:

```
npm instalar swr
```

Cree un nuevo archivo, `api.js`, para gestionar las solicitudes de API mediante la función `fetch` de cualquier biblioteca P de su elección. Este archivo contendrá funciones para cada PI o punto final:

```
exportar const fetchTodos = async () => {
    constante respuesta = await fetch('/api/todos');
    const data = await respuesta.json();
    devolver datos;
};

exportar const createTodo = async (todo) => {
    respuesta constante = esperar fetch('/api/todos', {
        método: 'POST',
        encabezados: {
            'Tipo de contenido': 'application/json',
        },
        cuerpo: JSON.stringify(todo),
    });
    const data = await respuesta.json();
    devolver datos;
};

exportar const updateTodo = async (id, todo) => {
    respuesta constante = await fetch(`'/api/todos/${id}` , {
        método: 'PUT',
        encabezados: {
```

```
'Tipo de contenido': 'application/json',
},
cuerpo: JSON.stringify(todo),
});
const data = await respuesta.json();
devolver datos;
};

exportar const deleteTodo = async (id) => {
respuesta constante = await fetch(`/api/todos/${id}` , {
método: 'BORRAR',
});
const data = await respuesta.json();
devolver datos;
};
```

- En su componente Next.js, importe las dependencias necesarias y utilice el Gancho SWR para obtener y administrar los datos de los puntos finales de la API:

```
importar useSWR desde 'swr';
importar { fetchTodos } desde './api';
constante TodoList = () => {
const { datos: todos, error } = useSWR('/api/todos', fetchTodos);

si (error) {
devolver <p>Error: {error.message}</p>;
}
si (!todos) {
regresar <p>Cargando...</p>;
}
devolver (
<div>
    Lista de tareas pendientes
<ul>
```

```
{todos.map((todo) => (
    <li clave={todo.id}>{todo.title}</li>
))
</ul>
</div>
);
};

exportar TodoList predeterminado;
```

4. Actualice los puntos finales de su API en el archivo `pages/api/todos.js` para manejar las solicitudes respectivas:

```
exportar función predeterminada manejador(req, res) {
    si (req.metodo === 'GET') {
        // Manejar la solicitud GET para obtener todos los tareas pendientes
        constante todos = [
            { id: 1, título: 'Todo 1' },
            { id: 2, título: 'Todo 2' },
            // ...
        ];
        res.status(200).json(todos);
    } de lo contrario si (req.metodo === 'POST') {
        // Manejar la solicitud POST para crear una nueva tarea
        // ...
    } de lo contrario si (req.metodo === 'PUT') {
        // Manejar solicitud PUT para actualizar una tarea pendiente
        // ...
    } de lo contrario si (req.metodo === 'BORRAR') {
        // Manejar la solicitud DELETE para eliminar una tarea pendiente
        // ...
    }
}
```

5. Ahora, puedes usar el componente `TodoList` en tu aplicación Next.js, y automáticamente buscará y actualizará los TODO usando el gancho SWR.

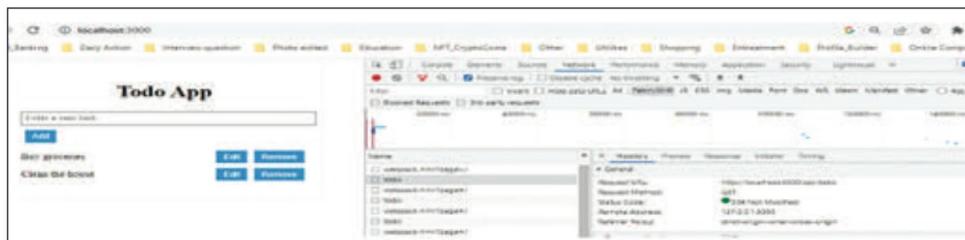


Figura 8.4: Ejemplo de interfaz de usuario que consume API RESTful

Recuerde iniciar su aplicación Next.js (`npm run dev`) para ver los cambios en acción. Asegúrese de tener las importaciones y los archivos necesarios en las ubicaciones adecuadas para que el código funcione correctamente. (ver Figura 8.4).

Nota: El repositorio de GitHub proporcionado contiene ejemplos de código relacionados con el libro "Desarrollo de aplicaciones web escalables con Next.js y React". En concreto, el código de la rama "chapter_8_API_DEMO" demuestra la integración de la API en una aplicación Next.js.

Para usar la API de GraphQL con Apollo Client en una aplicación Next.js, deberá instalar las dependencias necesarias y configurar Apollo Client. Siga estos pasos:

1. Instale las dependencias necesarias:

```
npm install apollo-boost graphql
```

Cree un nuevo archivo, `apolloClient.js`, para configurar el cliente pollo.

```
import { ApolloClient, InMemoryCache } from 'apollo-boost';
const cliente = new ApolloClient({
  uri: '/api/graphql',
  cache: new InMemoryCache(),
});
export default cliente;
```

Cree una nueva página en el directorio de páginas, por ejemplo, `books.js`, para obtener y mostrar los libros:

```
import { gql, useQuery } from '@apollo/client';
import cliente from '../apolloClient';
```

```
constante GET_BOOKS = gql`  
    consulta GetBooks {  
        libros {  
            ...libro  
            título  
            autor  
        }  
    };  
  
    exportar función predeterminada BooksPage() {  
        const { cargando, error, datos } = useQuery(GET_BOOKS, { cliente });  
        si (cargando) {  
            regresar <p>Cargando...</p>;  
        }  
        si (error) {  
            devolver <p>Error: {error.message}</p>;  
        }  
  
        const { libros } = datos;  
        devolver (  
            <div>  
                Libros  
                {libros.map((libro) => (  
                    <div key={libro.id}>  
                        <h2>{libro.título}</h2>  
                        <p>{libro.autor}</p>  
                    </div>  
                ))}  
            </div>  
        );  
    };
```

4. Actualice graphqlHandler en el archivo de ruta PI, apigraphql.js, para manejar los encabezados CORS y habilitar la introspección:

```

importar { ApolloServer, gql } desde 'apollo-server-micro';
constante typeDefs = gql` 
  tipo Libro {
    ¡hice!
    Título: ¡Cuerda!
    autor: ¡Cuerda!
  }

  tipo Consulta {
    libros:[¡Libro!]!
    libro(id: ID!): Libro
  }
  ...definiciones de interpretación...
}

const libros = [
  { id: '1', título: 'Libro 1', autor: 'Autor 1' },
  { id: '2', título: 'Libro 2', autor: 'Autor 2' },
];
resolvers constantes = {
  Consulta: {
    libros: () => libros,
    libro: (padre, argumentos) => libros.find((libro) => libro.id === argumentos.id),
  },
  // ...Resolutores de mutaciones...
};

constante apolloServer = nuevo ApolloServer({
  definiciones de tipos,
  resolutores,
}

```

```

    introspección: cierto,
});

exportar const config = {
    api: {
        bodyParser: falso,
    },
};

exportar predeterminado apolloServer.createHandler({ ruta: '/api/graphql' });

```

5. Ahora, puedes visitar la página "libros" en tu aplicación Next.js para ver la lista de libros obtenidos de la API.

Recuerde iniciar su aplicación Next.js (`npm run dev`) para ver los cambios en acción. Asegúrese de tener las importaciones y los archivos necesarios en las ubicaciones adecuadas para que el código funcione correctamente.

Manejo de errores y excepciones en llamadas API

En Next.js, la gestión de errores y excepciones en las llamadas a la API implica implementar la lógica de gestión de errores tanto en el servidor como en el cliente. Aquí tienes un enfoque general:

1. Manejo de errores del lado del servidor:

- En su controlador de ruta API (`pages/api/your-api-route.js`), envuelva su código dentro de un bloque try-catch.
- Detectar cualquier error potencial que pueda ocurrir durante la llamada API.
- Devuelve una respuesta de error apropiada utilizando los métodos `res.status()` y `res.json()`. Por ejemplo

```

    exportar función asíncrona predeterminada manejador(req, res) {

        intentar {
            // Código API aquí
            res.status(200).json({ mensaje: 'Éxito' });

        } captura (error) {

```

```

    res.status(500).json({ mensaje: 'Error interno del servidor'
  });
}

}

```

2. Manejo de errores del lado del

- cliente: • En el código del lado del cliente (componente React), gestione cualquier error o excepción que pueda ocurrir durante la llamada a la API.
- API. • Use un bloque `try-catch` o `Promise.catch()` para detectar cualquier error generado durante la llamada a la API.
- Puede mostrar mensajes de error al usuario o realizar acciones apropiadas.
- acciones en función del error recibido.
- Aquí hay un ejemplo de manejo de errores en un componente del lado del cliente de Next.js usando `fetch`:

```

importar { useState } de 'react';
exportar función predeterminada MyComponent() {
  const [error, setError] = useState(null); const
  fetchData = async () => {
    intentar {
      const respuesta = await fetch('/api/su-ruta-api');
      si (!respuesta.ok) {
        lanzar nuevo Error('La solicitud falló');
      }
      // Procesar los datos de respuesta
    } captura (error) {
      setError('Ocurrió un error');
    }
  };
  devolver (
    <div>
      {error && <p>{error}</p>}
      <button onClick={fetchData}>Obtener datos</button>
    </div>
  )
}

```

Implementación de API Restful y GraphQL

```
    );
}
```

. manejando errores con SWR

SWR es una popular biblioteca de obtención de datos para Next.js que proporciona capacidades integradas de manejo de errores.

```
importar useSWR desde 'swr';

constante MiComponente = () => {
  const { datos, error } = useSWR('/api/su-ruta-api', fetcher);

  si (error) {
    // Manejar el error
    devolver <p>Error: {error.message}</p>;
  }

  // Manejar estados de carga y éxito
  si (!datos) {
    regresar <p>Cargando...</p>;
  }

  // Representar los datos
  devolver <div>Datos: {datos}</div>;
};

};
```

En el ejemplo anterior, el gancho `useSWR` se utiliza para obtener datos de la ruta PI especificada `apiyour-api-route` mediante la función `fetcher` . Las variables `data` y `error` proporcionadas por SWR se utilizan para gestionar los diferentes estados de la llamada a la API.

4. Manejo de errores con el cliente Apollo:

Si está utilizando Apollo Client para llamadas de API GraphQL en Next.js, puede manejar errores utilizando los ganchos `useQuery` y `useMutation` proporcionados por Apollo.

```
importar { useQuery, gql } desde '@apollo/client';
constante GET_DATA = gql`  

  consulta ObtenerDatos {  

    // nuestra definición de uery aquí
```

```
}

constante MiComponente = () => {

    const { datos, cargando, error } = useQuery(GET_DATA);

    si (cargando) {

        regresar <p>Cargando...</p>;

    }

    si (error) {

        // Manejar el error

        devolver <p>Error: {error.message}</p>;

    }

    // Representar los datos

    devolver <div>Datos: {datos}</div>;

};

}
```

En el ejemplo anterior, se utiliza el gancho `useQuery` para obtener datos mediante `GET_DATA`. Las variables `data`, `loading` y `error` proporcionadas por Apollo se utilizan para gestionar los diferentes estados de la llamada a la API.

Estos ejemplos muestran cómo gestionar errores y excepciones en llamadas a la API mediante SWR y Apollo Client en Next.js. Puede personalizar la lógica de gestión de errores según sus requisitos específicos y las estructuras de respuesta.

Mejores prácticas para la seguridad y autenticación de API en aplicaciones Next.js

Proteger las API e implementar la autenticación en aplicaciones Next.js implica seguir las mejores prácticas para proteger datos confidenciales y evitar el acceso no autorizado. A continuación, se presentan algunas prácticas recomendadas para la seguridad y autenticación de las API en aplicaciones Next.js:

- Utilice HTTPS: Siempre sirva su aplicación Next.js a través de HTTPS para cifrar la comunicación entre el cliente y el servidor, garantizando la confidencialidad e integridad de los datos.
- Implementar una autenticación adecuada:
Utilice un mecanismo de autenticación seguro como JSON Web Tokens (JWT) o OAuth.

Implementar algoritmos de hash de contraseñas fuertes como bcrypt para almacenar de forma segura las contraseñas de los usuarios.

Utilizar técnicas de administración de sesiones para manejar las sesiones de usuario y almacenar datos de sesión de forma segura (por ejemplo, utilizando cookies seguras con indicadores `ttponly` y `Secure`).

Emplear mecanismos como limitación de velocidad y bloqueos de cuentas para evitar ataques de fuerza bruta.

- Proteger datos confidenciales:

Evite almacenar información confidencial, como contraseñas o claves API, en texto plano. En su lugar, utilice soluciones de almacenamiento seguro, como variables de entorno o un sistema seguro de gestión de claves.

Utilice cifrado al almacenar o transmitir datos confidenciales, especialmente en bases de datos o respuestas de API.

- Validar y desinfectar la entrada del usuario:

Aplicar validación y desinfección de entrada para evitar vulnerabilidades comunes como ataques de inyección SQL y secuencias de comandos entre sitios (XSS).

Utilice bibliotecas o marcos que ofrezcan validación integrada y mecanismos de sanitización.

- Implementar el control de acceso basado en roles (RBAC):

Definir roles y permisos para diferentes tipos de usuarios.

Restringir el acceso a determinados puntos finales o acciones de API según el usuario Roles y permisos.

- Protección contra falsificación de solicitudes entre sitios (CSRF)

Implementar mecanismos de protección CSRF, como el uso de tokens CSRF o cookies del mismo sitio, para evitar solicitudes no autorizadas de sitios web maliciosos.

- Claves API y secretos seguros:

Proteja las claves API, los tokens y otros secretos almacenándolos en soluciones de almacenamiento seguro (por ejemplo, variables de entorno) y evite enviarlos al control de versiones.

- Habilitar CORS (intercambio de recursos entre orígenes):

Configure los encabezados CRS de forma adecuada para restringir el acceso a su PI desde dominios no autorizados y evitar ataques de origen cruzado.

- Actualizar periódicamente las dependencias:

Mantenga su aplicación Next.js y sus dependencias actualizadas para beneficiarse de los parches de seguridad y las correcciones de errores.

- Realizar auditorías y pruebas de seguridad:

Realizar periódicamente auditorías de seguridad, revisiones de código y análisis de vulnerabilidades para identificar y abordar posibles vulnerabilidades de seguridad.

Realice pruebas de penetración y evaluaciones de seguridad exhaustivas para descubrir cualquier debilidad en la seguridad de su aplicación.

Recuerde que la seguridad es un proceso continuo y es esencial mantenerse actualizado sobre las últimas prácticas de seguridad, vulnerabilidades y mejores prácticas para garantizar la seguridad continua de su aplicación Next.js y sus API.

Conclusión

Aprender a implementar APIs en Next.js puede mejorar considerablemente tu capacidad para crear aplicaciones web potentes e interactivas. Al comprender los fundamentos del desarrollo de APIs e integrarlos en tus proyectos de Next.js, puedes liberar el potencial para una comunicación e intercambio de datos fluidos.

Para consolidar su comprensión, es beneficioso explorar ejemplos prácticos. Al trabajar con un ejemplo práctico, puede adquirir experiencia práctica en la configuración de PIs en Next.js. Este ejemplo podría implicar la creación de puntos finales de PI para gestionar operaciones CRUD relacionadas con partículas, como la creación de nuevas partículas, la recuperación de datos de partículas, la actualización de partículas existentes y la eliminación de partículas.

Siguiendo tutoriales y documentación específicos de Next.js y frameworks populares como Apollo Server, aprenderá a implementar PIs RESTful y Raph de forma eficiente. Descubrirá cómo definir los esquemas y resolvers necesarios, gestionar la renderización del lado del servidor y gestionar eficazmente errores y excepciones.

A lo largo de este proceso, obtendrá conocimientos sobre las mejores prácticas de seguridad y autenticación de API.

Al aplicar estos principios y adquirir experiencia práctica con ejemplos prácticos, dominará la implementación de PI en sus aplicaciones Next.js. Este conocimiento le permitirá crear aplicaciones web robustas y escalables que aprovechen el potencial de las API para brindar experiencias dinámicas e interactivas a sus usuarios.

En el siguiente capítulo, exploraremos cómo guardar datos en bases de datos SQL y NoSQL en el contexto de las aplicaciones Next.js. El almacenamiento de datos es un aspecto crucial de la mayoría de las aplicaciones web, y comprender las diferencias entre las bases de datos SQL y NoSQL puede ayudarle a tomar decisiones informadas sobre qué tipo de base de datos utilizar según los requisitos de su proyecto.

Preguntas de opción múltiple

1. ¿Cuál de los siguientes es un beneficio importante de utilizar IP en el desarrollo web?
 - A. Mejora del diseño de la interfaz de usuario
 - B. Permitir una comunicación fluida entre sistemas
 - C. Optimización del rendimiento de la base de datos
 - D. Aumentar la velocidad de procesamiento del servidor
2. El estilo arquitectónico PI proporciona una forma flexible y eficiente de
¿Consultar y mutar datos?
 - A. API RESTful
 - B. API SOAP
 - C. API de GraphQL
 - D. API de RPC
- ¿Qué framework se puede utilizar para configurar raph Next.js? IPs en
 - A. Express.js
 - B. Servidor Apollo
 - C. Django
 - D. Laravel
4. ¿Qué enfoque sigue la API RESTful para manejar operaciones CRUD?
 - A. Separación entre comandos y consultas
 - B. Arquitectura basada en eventos
 - C. Enfoque basado en recursos
 - D. Patrón de middleware

5. ¿Cuáles son algunas de las mejores prácticas para proteger y autenticar las API en aplicaciones Next.js?
- A. Usar HTTPS, implementar mecanismos de autenticación adecuados como JWT u OAuth, proteger datos confidenciales y realizar auditorías de seguridad periódicas.
 - B. Utilizar HTTP, almacenar datos confidenciales en texto sin formato y confiar en validación del lado del cliente
 - C. Habilitar la protección CORS, usar variables de entorno para claves API y evitar la validación de entrada
 - D. Usar OAuth solo para autenticación y almacenar secretos en la aplicación código y nunca actualizar las dependencias

Respuestas

1.b	
2.	do
. b	
4.	do
5.	a

Capítulo 9

Uso de diferentes tipos de bases de datos

Introducción

En este capítulo, profundizaremos en el mundo de las bases de datos y exploraremos cómo pueden utilizarse en aplicaciones Next.js. Las bases de datos desempeñan un papel crucial en el desarrollo web, actuando como la columna vertebral para el almacenamiento, la recuperación y la gestión de datos.

Para comenzar, ofreceremos una descripción general de las bases de datos y su importancia en el desarrollo web. Comprender los fundamentos de las bases de datos sentará las bases para explorar los distintos tipos de bases de datos disponibles. Le presentaremos las bases de datos relacionales, las bases de datos NoSQL e incluso profundizaremos en las bases de datos gráficas, destacando sus características únicas y casos de uso.

Seleccionar la base de datos adecuada para su aplicación es fundamental, ya que cada tipo tiene sus propias fortalezas y debilidades. Le guiaremos en el proceso de toma de decisiones, considerando factores como la estructura de datos, la escalabilidad y el rendimiento.

Una vez que hayas elegido, te ayudaremos a configurar una conexión de base de datos en Next.js. Te explicaremos los pasos para conectar bases de datos populares como MongoDB, MySQL y PostgreSQL a tu aplicación Next.js, permitiéndote aprovechar sus funcionalidades sin problemas.

A continuación, exploraremos las operaciones comunes que realizará con bases de datos, centrándonos en las operaciones CRUD (Crear, Leer, Actualizar y Eliminar). Aprenderá a interactuar con la base de datos seleccionada, manipular datos y recuperar información usando Next.js.

La gestión de errores de la base de datos y la implementación de técnicas de depuración eficaces es otro aspecto crucial que abordaremos. La resolución de problemas comunes y la adopción de las mejores prácticas para la gestión de errores garantizarán el correcto funcionamiento de su aplicación Next.js.

La seguridad de las bases de datos es fundamental, y le brindaremos las mejores prácticas para proteger sus datos. Analizaremos técnicas para proteger su base de datos, proteger información confidencial y prevenir vulnerabilidades comunes.

Además, nos sumergiremos en el ámbito del modelado de datos y el diseño de esquemas. Obtendrá conocimientos sobre el diseño de estructuras de bases de datos eficientes, la creación de relaciones entre entidades y la optimización de consultas para un mejor rendimiento.

Finalmente, abordaremos la escalabilidad de su base de datos para satisfacer las demandas de una base de usuarios en crecimiento. Analizaremos estrategias para mejorar el rendimiento y garantizar una alta disponibilidad, permitiendo que su aplicación gestione eficazmente el aumento del tráfico y las interacciones de los usuarios.

A lo largo de este capítulo, presentaremos ejemplos prácticos y casos de uso para ilustrar los conceptos analizados. Al finalizar, contará con los conocimientos y las herramientas necesarios para aprovechar diferentes tipos de bases de datos en sus aplicaciones Next.js, lo que le permitirá crear aplicaciones web modernas y robustas. ¡Embárquese juntos en este viaje hacia las bases de datos y descubra el poder de los datos en Next.js!

Estructura

En este capítulo cubriremos los siguientes temas:

- Configurar una conexión de base de datos en Next.js •
- Usar las bases de datos populares en Next.js como MongoDB, MySQL • Operaciones
- CRUD con la base de datos seleccionada • Manejo de
- errores de base de datos y técnicas de depuración • Mejores prácticas
- de seguridad de bases de datos • Modelado
- de datos y diseño de esquemas • Escalado de
- la base de datos para rendimiento y alta disponibilidad

Descripción rápida de la base de datos Sistema de gestión

Un Sistema de Gestión de Bases de Datos (SGBD) es un sistema de software que permite a los usuarios almacenar, organizar y gestionar grandes cantidades de datos de forma eficiente. Proporciona un enfoque estructurado y centralizado para gestionar los datos, facilitando el acceso, la recuperación, la modificación y la eliminación de información.

Componentes clave de un DBMS:

- Datos: El DBMS administra diferentes tipos de datos, como texto, números, imágenes, videos, etc., que se almacenan en varios formatos y estructuras.
- Base de datos: Una base de datos es una colección de datos relacionados, organizados y almacenados de forma estructurada. Consta de tablas, vistas, índices y otros objetos que definen la estructura y las relaciones de los datos.
- Esquema de base de datos. El esquema de base de datos define la estructura lógica y física de la base de datos. Describe las tablas, sus atributos (columnas), tipos de datos, restricciones y relaciones con otras tablas.

Lenguaje de Manipulación de Datos (DML): El DML permite a los usuarios recuperar, insertar, actualizar y eliminar datos en la base de datos. Las sentencias DML más comunes incluyen SELECT, INSERT, UPDATE y DELETE.

Lenguaje de definición de datos (DD). DD es un lenguaje utilizado para definir y gestionar la estructura de la base de datos. Incluye sentencias como CREATE, ALTER y DROP, que se utilizan para crear tablas, modificar su estructura o eliminarlas.

Optimización de consultas: El SGBD optimiza la ejecución de consultas para mejorar el rendimiento y la eficiencia. Analiza la consulta, evalúa diferentes planes de ejecución y selecciona el enfoque más eficiente para recuperar los datos.

- Gestión de Transacciones: El SGBD garantiza las propiedades ACID de las transacciones. ACID significa Atomicidad (las transacciones se tratan como una sola unidad de trabajo), Consistencia (garantiza la integridad de los datos), Aislamiento (las transacciones se ejecutan de forma aislada) y Durabilidad (una vez confirmada una transacción, sus cambios son permanentes).

Control de concurrencia: El SGBD gestiona el acceso concurrente a la base de datos por parte de múltiples usuarios o procesos. Garantiza la integridad de los datos cuando se ejecutan múltiples transacciones simultáneamente. Se utilizan técnicas como el bloqueo, el sellado de tiempo y el control de concurrencia multisésion para gestionar la concurrencia.

- Seguridad y autorización: El SGBD proporciona mecanismos para controlar el acceso a la base de datos y garantizar la seguridad de los datos. Incluye la autenticación de usuarios, la autorización y la gestión de privilegios para proteger los datos confidenciales del acceso no autorizado.
- Copia de seguridad y recuperación: El SGBD facilita la copia de seguridad y la recuperación de datos en caso de fallos del sistema, corrupción de datos o errores humanos. Permite a los usuarios crear copias de seguridad, restaurar datos a un estado anterior y realizar operaciones de recuperación para minimizar la pérdida de datos.

Tipos de DBMS:

- DBMS relacional (RDBMS): el tipo más utilizado, donde los datos se organizan en tablas con relaciones predefinidas entre ellas.
Los ejemplos incluyen MySQL, Oracle Database y Microsoft SQL Server.
- DBMS orientado a objetos (OODBMS): estos sistemas almacenan datos en forma de objetos, lo que permite la herencia, la encapsulación y el polimorfismo.
Son adecuados para estructuras de datos complejas. Algunos ejemplos son MongoDB y Apache Cassandra.
- SGBD jerárquico: Los datos se organizan en una estructura de árbol, donde cada registro tiene una relación padre-hijo. Fue popular en sus inicios, pero ha sido reemplazado en gran medida por otros modelos.
- Sistema de gestión de bases de datos (SGBD) en red: Similar a un SGB jerárquico, pero con una estructura más flexible. Permite que los registros tengan múltiples registros principales y secundarios, creando relaciones complejas.
- Sistemas de gestión de bases de datos NoSQL: Las bases de datos NoSQL (no solo SQL) están diseñadas para gestionar datos a gran escala, no estructurados y diversos. Ofrecen alta escalabilidad, rendimiento y flexibilidad. Algunos ejemplos sonongoDB, Cassandra y Redis.
- Sistemas de Gestión de Bases de Datos NewSQL: Las bases de datos NewSQL combinan la escalabilidad y el rendimiento de NoSQL con las propiedades ACID de las bases de datos relacionales tradicionales. Su objetivo es abordar las limitaciones de los sistemas de gestión de bases de datos relacionales tradicionales (RDBMS) para gestionar grandes volúmenes de datos y usuarios concurrentes. Los sistemas NewSQL ofrecen arquitecturas distribuidas y un procesamiento paralelo eficiente, manteniendo la consistencia transaccional. Algunos ejemplos son Google Spanner, Cockroach DB y TiDB.

beneficios de cantar una D

- Centralización de datos: Un SGBD permite a los usuarios almacenar datos en una ubicación centralizada, lo que proporciona una vista unificada de los datos para diferentes aplicaciones y usuarios. Esto elimina la redundancia y la inconsistencia de los datos.
- Intercambio de datos: El SGBD permite que varios usuarios accedan y comparten datos simultáneamente. Proporciona acceso controlado y garantiza la integridad de los datos mediante mecanismos de control de concurrencia.
- Seguridad de datos: DBMS incluye características como autenticación de usuarios, control de acceso y cifrado para proteger datos confidenciales contra accesos no autorizados, garantizando la seguridad de los datos y el cumplimiento de las regulaciones.
- Integridad de los datos: el DBMS aplica restricciones de integridad de datos, como restricciones de clave principal e integridad referencial, para mantener la precisión y la consistencia de los datos.

- Consistencia de datos: El SGBD garantiza que los datos se mantengan consistentes y válidos incluso durante accesos y actualizaciones concurrentes. Las propiedades ACID garantizan la ejecución fiable de las transacciones y mantienen la integridad de los datos.

Escalabilidad de datos: Un SGBD proporciona mecanismos para gestionar grandes cantidades de datos de forma eficiente. Permite el escalamiento horizontal mediante la adición de más servidores o nodos para distribuir los datos y la carga de trabajo .

- Recuperación de datos: El SGBD admite mecanismos de copia de seguridad y recuperación para evitar la pérdida de datos en caso de fallos del sistema, desastres o errores humanos. Permite restaurar la base de datos a un estado anterior o recuperarla en un momento dado.
- Independencia de datos: DBMS proporciona independencia de datos al separar la estructura lógica de la base de datos (esquema) de los detalles de almacenamiento físico. Esto permite realizar modificaciones al esquema de la base de datos sin afectar las aplicaciones que utilizan los datos.
- Consulta e informes de datos: DBMS ofrece lenguajes de consulta (por ejemplo, SQL) y herramientas para realizar consultas de datos complejas, generar informes y extraer información útil de los datos.
- Rendimiento mejorado: DBMS optimiza la ejecución de consultas, la indexación y las técnicas de almacenamiento en caché para mejorar el rendimiento y los tiempos de respuesta. Utiliza algoritmos de optimización de consultas para elegir los planes de ejecución más eficientes.

Mantenimiento de datos: El SGBD proporciona herramientas para la gestión y el mantenimiento de la base de datos, como copias de seguridad, recuperación, indexación y optimización del rendimiento. Simplifica las tareas administrativas y reduce el esfuerzo necesario para la gestión de la base de datos .

En resumen, un Sistema de Gestión de Bases de Datos (SGBD) es un sistema de software crucial que proporciona una gestión de datos eficiente y fiable. Ofrece un enfoque estructurado para almacenar, organizar, manipular y recuperar datos, garantizando al mismo tiempo su integridad, seguridad y escalabilidad. Un SGBD desempeña un papel vital en las aplicaciones y organizaciones modernas, permitiendo una gestión eficaz de los datos y respaldando las operaciones empresariales críticas.

Tenga en cuenta que este capítulo se centrará en dos tipos de sistemas de gestión de bases de datos (SGBD): SGBD relacional (SGBDR) con MySQL y SGBD NoSQL con MongoDB. Exploraremos las características, ventajas y casos de uso de estos dos populares sistemas para comprender a fondo sus capacidades. En este capítulo no se abordarán otros tipos de SGBD.

Gestión de bases de datos relacionales Sistemas

Un Sistema de Gestión de Bases de Datos Relacionales (SGBDR) es un sistema de software que gestiona bases de datos relacionales. Proporciona una forma estructurada de almacenar, organizar y recuperar datos según un esquema predefinido. yS es uno de los SGBDR más populares y se utiliza ampliamente en aplicaciones web.

Las bases de datos relacionales se basan en el modelo relacional, que organiza los datos en tablas compuestas por filas y columnas. Cada tabla representa una entidad o una relación entre entidades. En yS, las tablas se crean con un esquema específico que define la estructura y los tipos de datos de las columnas.

La normalización es el proceso de diseñar un esquema de base de datos para eliminar la redundancia y garantizar la integridad de los datos. Implica dividir tablas grandes en tablas más pequeñas relacionadas y establecer relaciones entre ellas mediante claves primarias y externas. La normalización ayuda a reducir la redundancia de datos, mejorar su consistencia y facilitar su recuperación eficiente.

Existen varias formas normales en la normalización de bases de datos, entre ellas:

- Primera forma normal (1NF): garantiza que cada columna de una tabla contenga valores atómicos (indivisibles y no se pueden dividir más).
- Segunda forma normal (2NF): se basa en la 1NF y requiere que cada columna que no sea clave en una tabla dependa funcionalmente de la clave principal completa.
- Tercera forma normal (3NF): se basa en la 2NF y requiere que cada columna no clave de una tabla dependa únicamente de la clave principal y no de otras columnas no clave.
- Forma normal de Boyce-Codd (BCNF): una forma más estricta de 3NF que elimina todas las dependencias funcionales entre columnas que no son clave.
- Cuarta forma normal (4NF): garantiza que no haya dependencias multivalor dentro de una tabla.

Ahora vamos a analizar OLAP (procesamiento analítico en línea) y OLTP (procesamiento analítico en línea). Procesamiento de transacciones):

OLAP:

OLAP es una categoría de software y tecnología que se utiliza para el análisis complejo y la generación de informes sobre grandes volúmenes de datos. Se centra en proporcionar inteligencia empresarial y capacidades de apoyo a la toma de decisiones. Las bases de datos OLAP están diseñadas para operaciones de lectura intensiva y consultas analíticas complejas. Normalmente almacenan datos agregados.

en una estructura multidimensional, conocida como cubo de datos, que permite realizar consultas rápidas y segmentar los datos desde diferentes perspectivas.

OLTP:

OLTP, por otro lado, es una categoría de software y tecnología utilizada para gestionar y procesar las operaciones transaccionales diarias de una organización.

Las bases de datos OLTP están diseñadas para el procesamiento de datos a alta velocidad, la consistencia transaccional y el acceso concurrente. Admiten operaciones como la inserción, actualización y eliminación de pequeñas cantidades de datos en tiempo real. MySQL, al ser un RDBMS, se utiliza comúnmente para aplicaciones OLTP donde la consistencia de los datos y el procesamiento de transacciones en tiempo real son cruciales.

Sistemas de gestión de bases de datos NoSQL

NoSQL (No Solo SQL) es un tipo de sistema de gestión de bases de datos que se aleja del modelo relacional tradicional utilizado en sistemas de gestión de bases de datos relacionales (RDBMS) como MySQL. MongoDB es un ejemplo popular de un DBMS NoSQL, y exploremos sus características y contexto en detalle:

- Modelo orientado a documentos:

Las bases de datos NoSQL, como MongoDB, utilizan un modelo orientado a documentos para el almacenamiento de datos. En lugar de tablas con filas y columnas, MongoDB almacena los datos en documentos flexibles y autodescriptivos. Estos documentos suelen representarse en formato similar a JSON (BSON), lo que permite el almacenamiento de estructuras anidadas y esquemas dinámicos.

- Flexibilidad del esquema:

A diferencia de la estructura de esquema fija de RDBS, las bases de datos NoS como OndoDB proporcionan flexibilidad de esquema. Cada documento puede tener su propia estructura, lo que le permite almacenar distintos tipos de datos en la misma colección. Esta flexibilidad simplifica el proceso de desarrollo, especialmente en escenarios donde los esquemas de datos evolucionan con el tiempo.

- Escalabilidad y alto rendimiento:

Las bases de datos NoSQL están diseñadas para escalar horizontalmente, lo que significa que pueden distribuir datos entre múltiples servidores o nodos para manejar grandes volúmenes de datos y tráfico. ongoDB, en particular, se destaca en escalabilidad, ofreciendo capacidades de fragmentación automática para distribuir datos y consultas entre un clúster de máquinas. Proporciona un alto rendimiento tanto para operaciones de lectura como de escritura, lo que lo hace adecuado para aplicaciones a gran escala.

- Replicación y alta disponibilidad:

MongoDB admite conjuntos de réplicas, que son clústeres de nodos de base de datos que mantienen copias de los mismos datos. Los conjuntos de réplicas ofrecen redundancia de datos y alta disponibilidad, lo que garantiza que el sistema permanezca operativo incluso en caso de fallo de un nodo. MongoDB utiliza mecanismos automáticos de comutación por error para seleccionar un nuevo nodo principal y mantener un servicio continuo.

- Consulta e indexación:

MongoDB ofrece un lenguaje de consulta flexible y potente para la recuperación de datos, conocido como Lenguaje de Consulta MongoDB (MQL). MQL admite una amplia gama de funciones de consulta, como filtrado, ordenación, agregación y consultas geoespaciales. MongoDB también admite la indexación, lo que permite optimizar el rendimiento de las consultas mediante la creación de índices en campos específicos.

- Casos de uso y contexto:

MongoDB es especialmente adecuado para casos de uso donde la flexibilidad, la escalabilidad y el procesamiento de datos en tiempo real son cruciales. Se utiliza comúnmente en escenarios como sistemas de gestión de contenido, análisis en tiempo real, sistemas de registro y monitorización, y gestión de perfiles de usuario. Además, el modelo de documentos de MongoDB facilita la integración con frameworks y lenguajes de desarrollo web modernos.

Diferencia entre DBMS SQL y NoSQL:

Aspecto	SQL (relacional) Base de datos)	NoSQL (base de datos no relacional)
Datos Estructura	Datos tabulares estructurados con esquema fijo.	Datos flexibles, semiestructurados o no estructurados con esquema dinámico.
Esquema	Requiere un esquema predefinido con estricta estructura.	Esquema sin esquema o esquema flexible que puede evolucionar con el tiempo.
Consulta Idioma	SQL (lenguaje de consulta estructurado) se utiliza para definir y manipular datos.	Lenguajes de consulta específicos de cada base de datos NoSQL (por ejemplo, MongoDB utiliza un lenguaje de consulta similar a JavaScript).

Escalabilidad	Escalamiento vertical (escalamiento de recursos de hardware de un solo servidor).	Escalamiento horizontal (agregar más servidores para distribuir la carga).
Datos Relaciones	Admite relaciones complejas entre tablas utilizando claves externas.	Diseñado principalmente para manejar datos desnormalizados con relaciones mínimas o nulas.
ÁCIDO Cumplimiento	Se mantienen las propiedades ACID (atomicidad, consistencia, aislamiento, durabilidad).	ay sacrificar algunas propiedades de CID por escalabilidad y rendimiento mejorados (por ejemplo, consistencia final).
Actas	Fuerte soporte para transacciones (agrupoando múltiples operaciones en una sola unidad).	Varía según las bases de datos NoSQL y algunas ofrecen soporte para transacciones limitado o nulo.
Integridad de los datos	Aplica restricciones estrictas de integridad de datos (por ejemplo, claves principales, claves externas).	Se basa en la lógica de la aplicación para mantener la integridad de los datos.
Almacenamiento	Normalmente utiliza un esquema fijo y almacena datos en tablas.	Varios formatos de almacenamiento, incluidos modelos clave-valor, de documento, columnas o basados en gráficos.
Casos de uso	Ideal para datos relacionales estructurados con consultas y transacciones complejas (por ejemplo, sistemas bancarios).	Ideal para manejar grandes volúmenes de datos semiestructurados o no estructurados, creación rápida de prototipos y escenarios con requisitos cambiantes (por ejemplo, análisis de redes sociales).

Tabla 9.1: DBMS SQL versus NoSQL

Configurar una conexión a una base de datos en Next.js

Para configurar una conexión de base de datos en Next.js con MongoDB y MySQL, necesitará instalar los paquetes necesarios y configurar los ajustes de conexión.

Aquí tienes una guía paso a paso para cada base de datos:

1. Instalación de los paquetes necesarios:

```
npm install mongodb mysql
```

Cree un archivo independiente para gestionar las conexiones a la base de datos. Para MongoDB, cree un archivo llamado `mongodb.js` y, para MySQL, un archivo llamado `mysql.js`.

En `mongodb.js`, importe el paquete `mongodb` y defina la configuración de conexión:

```
const MongoClient = require('mongodb');

const uri = 'mongodb://localhost:27017'; // Reemplace con su URI de conexión MongoDB

const dbName = 'your-database-name'; // Reemplace con el nombre de su base de datos
```

```
const cliente = nuevo MongoClient(uri, {
```

```
    useNewUrlParser: verdadero,
```

```
    useUnifiedTopology: cierto,
```

```
});
```

```
función asíncrona connectMongoDB() {
```

```
    intentar {
```

```
        esperar cliente.connect();
```

```
        console.log('Conectado a MongoDB');
```

```
        const db = cliente.db(dbName);
```

```
        retorna db;
```

```
    } captura (error) {
```

```
        console.error('Error al conectar a MongoDB', error);
```

```
        arrojar error;
```

```
}
```

```
}
```

```
módulo.exports = connectMongoDB;
```

n mysl.s importa el paquete mysl y define la conexión
ajustes:

```
constante mysql = require('mysql');

conexión constante = mysql.createConnection({
    host: 'localhost', // Reemplace con su host MySQL
    usuario: 'su-nombre-de-usuario', // Reemplace con su MySQL
    nombre de usuario

    contraseña: 'su-contraseña', // Reemplace con su contraseña de MySQL

    base de datos: 'su-nombre-de-base-de-datos', // Reemplace con su
    nombre de la base de datos

});
```

```
conexión.connect((error) => {
    si (error) {
        console.error('Error al conectar a MySQL', error);
        arrojar error;
    }
    console.log('Conectado a MySQL');
});
```

```
módulo.exports = conexión;
```

- En el código de Next.js, importe los archivos de base de datos correspondientes y utilice las conexiones según sea necesario. Por ejemplo, cree un archivo llamado pagesindex.js e incluya el siguiente código:

```
importar { useEffect } de react;
importar connectMongoDB desde '../mongodb';
importar conexiónMySQL desde '../mysql';
```

```
exportar función predeterminada HomePage() {  
    usarEfecto(() => {  
        función asíncrona fetchData() {  
            const db = await connectMongoDB();  
  
            // Realice operaciones de MongoDB aquí  
            // Ejemplo: const collection = db.collection ('tu-colección');  
  
            // Ejemplo: const data = await collection.find().  
            aArray();  
  
            connectionyS.uery(SEE * O yourtable, (error, resultados, campos)  
=> {  
    si (error) {  
        console.error('Error al consultar MySQL', error);  
        arrojar error;  
    }  
  
    // Procesa los resultados de la consulta MySQL aquí  
});  
}  
  
obtener datos();  
}, []);  
  
devolver (  
    <div>  
        {/*contenido de nuestra página*/}  
    </div>  
);  
}
```