

Recuerde reemplazar los valores de marcador de posición (como `your-database-name`, `your-username`, `your-password`, etc.) con los detalles de conexión de base de datos reales.

Con esta configuración, tendrá una página Next.js que se conecta a bases de datos MongoDB y MySQL. La función `fetchData` se ejecuta al montar el componente, lo que le permite realizar operaciones con la base de datos y procesar los resultados según sea necesario.

Hay varias bibliotecas de terceros disponibles para trabajar con bases de datos en JavaScript y Node.js, como Mongoose, Sequelize, TypeORM y PrismORM.

Estas bibliotecas proporcionan la funcionalidad de mapeo relacional de objetos (ORM) y simplifican las interacciones de bases de datos.

Mientras que ongoose está diseñado específicamente para ongoDB, Sequelize se centra principalmente en bases de datos SQL (MySQL, PostgreSQL, SQLite, MSSQL). TypeORM, por otro lado, es compatible con diversas bases de datos, como PostgreSQL, MySQL, SQLite, MSSQL y Oracle.

Sin embargo, si prefiere una biblioteca que gestione bases de datos SQL y NoSQL, TypeORM es una excelente opción. TypeORM es versátil y le permite trabajar con diferentes tipos de bases de datos mediante un PI unificado. Ofrece un enfoque consistente e intuitivo para trabajar con diversas bases de datos, lo que resulta práctico si necesita alternar entre bases de datos SQL y NoSQL dentro del mismo proyecto.

PrismORM no es tan conocido ni popular como todas las demás bibliotecas mencionadas. Sin embargo, si cumple con sus requisitos específicos y lo considera adecuado para su proyecto, puede explorar sus características y evaluar sus capacidades.

En última instancia, la elección de la biblioteca de bases de datos depende de las necesidades específicas de su proyecto, su familiaridad con la biblioteca y las bases de datos con las que trabaja. Considere factores como las bases de datos compatibles, la funcionalidad ORM, la facilidad de uso, el apoyo de la comunidad y el desarrollo activo al seleccionar la biblioteca adecuada para su proyecto. La siguiente tabla compara tres bibliotecas comunes que facilitan el trabajo con bases de datos.

Característica	Mangosta (MongoDB)	Secuela (MySQL, PostgreSQL, SQLite, MSSQL)	TiposORM (Varias bases de datos)
Bases de datos compatibles	MongoDB	MySQL, PostgreSQL, SQLite, MSSQL y más	PostgreSQL, MySQL, SQLite, MSSQL, Oracle

Funcionalidad ORM	Sí	Sí	Sí
Objeto-Relacional Mapeo (ORM)	Sí	Sí	Sí
Definición del esquema	Basado en esquemas (con Mongoose) Esquemas)	Basado en Modelos (con Sequelize) Modelos)	Basado en decorador Clases)
Lenguaje de consulta	Consulta de MongoDB Lenguaje (MQL)	SQL	SQL
Agrupación de conexiones	No (gestionado por Controlador MongoDB)	Sí	Sí
Apoyo para asociaciones Relaciones	Sí	Sí	Sí
Apoyo para Migraciones	No	Sí	Sí
Desarrollo Activo	Sí	Sí	Sí
Popularidad	Ampliamente utilizado	Ampliamente utilizado	Cada vez más popular

Tabla 9.2: Diferencia entre Mongoose, Sequelize y Type ORM

Operaciones CRUD con la base de datos seleccionada

Las operaciones CRUD se refieren a las operaciones básicas que se realizan en una base de datos: crear, leer, actualizar y eliminar. En Next.js, puede usar la biblioteca TypeORM para gestionar estas operaciones con bases de datos SQLite y MongoDB. Para establecer una conexión local, debe configurar la base de datos en su aplicación Next.js.

Una explicación paso a paso de cómo realizar operaciones CRUD usando TypeORM con SQLite y MongoDB en Next.js:

Paso 1: Configurar un proyecto Next.js

Crea un nuevo proyecto Next.js ejecutando el siguiente comando en tu terminal:

```
npx crear-próxima-aplicación mi-próxima-aplicación
```

Paso 2: Instalar dependencias

Navegue al directorio de su proyecto e instale las dependencias necesarias:

```
cd mi-próxima-aplicación
```

```
npm install typeorm sqlite3 mongodb
```

Paso 3: Crear una conexión a la base de datos

Cree una carpeta llamada onfig en el directorio raíz de su proyecto. Dentro de ella , cree un archivo llamado database.ts. Este archivo contendrá la configuración de conexión a la base de datos.

Para SQLite:

```
const { connectionOptions } = require('typeorm');
```

```
constante sliteonfig = {
    tipo: ranura,
    base de datos: './data/database.sqlite',
    sincronizar: verdadero,
    registro: verdadero,
    entidades: [
        // Agregue sus clases de entidad aquí
    ],
};
```

```
módulo.exports = sliteonfig;
```

```
exportar sliteonfig predeterminado;
```

```
const { connectionOptions } = require('typeorm');
```

```
constante mongoonfig = {
    tipo: mongodb,
```

```

anfitrión: localhost,
puerto: 27017,
base de datos: mibasededatos,
sincronizar: verdadero,
registro: verdadero,
entidades: [
    // Agregue sus clases de entidad aquí
],
};

módulo.exports = mongoonfig;

```

te De finir entidades

Cree una carpeta llamada `entities` en el directorio raíz de su proyecto. Dentro de ella, cree clases de entidad que representen su modelo de datos. Estas clases definirán la estructura de sus tablas o colecciones.

Por ejemplo, creamos una entidad "Usuario" :

```
const { Entidad, columna, columna primaria y secundaria } = require(-typeorm);
```

```

@Entidad()
clase Usuario {
    @PrimaryGeneratedColumn()

```

identificación;

```

    @Columna()
    nombre;

```

```

    @Columna()

```

correo electrónico;

```
// Agregue otras propiedades y relaciones según sea necesario  
}
```

```
módulo.exports = Usuario;
```

Paso 5: Implementar operaciones CRUD

En el directorio raíz de su proyecto, cree una carpeta llamada "pages". Dentro de esta carpeta, cree un archivo para cada operación CRD: create.tsx, read.tsx, update.tsx y delete.tsx. Estos archivos gestionarán las operaciones correspondientes.

Por ejemplo, en create.js, puedes implementar la lógica para crear un nuevo usuario

```
importar { useState } de react;  
importar { createonnection, getonnection } desde typeorm;  
importar sliteonfig desde ../config/database;  
  
exportar función predeterminada CreateUser() {  
    const [nombre, setame] = useState();  
    const [correo electrónico, establecerCorreo electrónico] = useState();  
  
    const handleSubmit = async (evento) => {  
        evento.preventDefault();  
  
        intentar {  
            const conexión = await createonnection(sliteonfig);  
            const userRepository = conexión.getRepository(Usuario);  
  
            const newUser = userRepository.create({ nombre, correo electrónico });  
            esperar userRepository.save(newUser);  
  
            esperar conexión.close();
```

```
        console.log(ser creado exitosamente);

    } captura (error) {
        console.error(Error al crear el usuario:, error);
    }
};

devolver (
    <formulario onSubmit={manejarEnvío}>
        <entrada
            tipo=texto
            valor={nombre}
            onChange={(evento) => setName(evento.objetivo.valor)}
            marcador de posición=nombre
        />
        <entrada
            tipo=correo
            electrónico
            valor={correo electrónico} onChange={(evento) => setEmail(evento.objetivo.valor)}
            marcador de posición=Correo electrónico
        />
        tipo de botón=enviar>crear ser/botón>
    </form>
);
}
```

De manera similar, puede implementar la lógica para las operaciones de lectura, actualización y eliminación conectándose a la base de datos, recuperando el repositorio apropiado y realizando las operaciones necesarias.

Paso 6: Ejecute la aplicación Next.js

Para iniciar su aplicación Next.js y probar las operaciones CRUD, ejecute el siguiente comando en su terminal:

```
npm ejecutar dev
```

Abra su navegador y acceda a `http://localhost:3001/create` para crear un nuevo usuario. Puede crear páginas adicionales para otras operaciones CRUD y probarlas también.

Nota: Asegúrese de tener un servidor SQLite o una instancia de MongoDB en ejecución en su equipo local, según la base de datos que utilice. Asegúrese de que las credenciales de la base de datos y los datos de conexión coincidan con los especificados en el archivo de configuración.

¡Listo! Ya has implementado operaciones CRUD con SQLite y MongoDB en Next.js usando la biblioteca TypeORM. Puedes ampliar este ejemplo para gestionar modelos de datos y operaciones más complejas según sea necesario.

Manejo de errores de base de datos y técnicas de depuración

En Next.js, la gestión de errores de base de datos y las técnicas de depuración implican garantizar una gestión adecuada de errores al interactuar con las bases de datos y emplear prácticas de depuración eficaces para identificar y resolver problemas. A continuación, se detalla cómo abordar estas tareas:

- Manejo de errores:

Conectarse a la base de datos: Para interactuar con una base de datos en Next.js, normalmente se usa una biblioteca como Prisma, Sequelize o Mongoose. Estas bibliotecas proporcionan mecanismos para conectarse a la base de datos y ejecutar consultas. Asegúrese de establecer una conexión con la base de datos antes de intentar cualquier operación.

Ejecución de consultas : Al ejecutar consultas de base de datos, es fundamental gestionar posibles errores. La mayoría de las bibliotecas de bases de datos proporcionan mecanismos para detectar y gestionar los errores devueltos por el servidor de bases de datos. Por ejemplo, con Prisma, se pueden usar bloques `try-catch` o gestión de errores basada en promesas para detectar y gestionar errores.

Mensajes de error: cuando ocurre un error durante las operaciones de la base de datos, es crucial proporcionar mensajes de error significativos para ayudar en la depuración. Incluya información relevante, como el tipo de error, la consulta en ejecución y cualquier dato relevante involucrado. Registrar los mensajes de error en un sistema central, como un archivo de registro o un servicio de monitoreo, puede ayudar a rastrear los problemas.

Manejo de errores eficiente : En Next.js, puede implementar límites de error mediante componentes de límite de error o el método `getDerivedStateFromError`. Estas técnicas permiten detectar errores dentro de los componentes y mostrar interfaces de usuario alternativas en lugar de bloquear toda la aplicación. Puede mostrar una página de error o un mensaje de error específico a los usuarios y registrar los detalles del error para su posterior investigación.

- **Técnicas de depuración:**

Registro : El registro es una técnica esencial para depurar problemas relacionados con la base de datos. Utilice bibliotecas de registro como Winston, Pino o las instrucciones `console.log` integradas para generar información relevante durante la ejecución. Registre detalles importantes, como los tiempos de ejecución de las consultas, los resultados y los mensajes de error, para detectar problemas.

Herramientas de depuración: Next.js ofrece potentes herramientas de desarrollo, como la extensión de navegador Next.js DevTools y las herramientas integradas para desarrolladores de React. Estas herramientas permiten inspeccionar componentes, examinar propiedades y supervisar las solicitudes de red. Úselas para identificar posibles problemas con las consultas a la base de datos o el flujo de datos.

Optimización de consultas : Si encuentra problemas de rendimiento con las consultas de la base de datos, es fundamental optimizarlas. Puede usar herramientas como el analizador de consultas o el generador de perfiles de la base de datos para identificar consultas lentas o inefficientes. Compruebe si faltan índices, uniones redundantes o patrones de consulta deficientes. Ajustar los índices o reescribir las consultas puede mejorar significativamente el rendimiento.

Revisión de código : Involucre a otros desarrolladores en la revisión de su código, especialmente al trabajar con operaciones complejas de bases de datos. Las revisiones de código entre pares pueden ayudar a identificar errores lógicos, posibles condiciones de carrera o patrones de código deficientes que podrían causar errores o afectar el rendimiento.

Pruebas unitarias : Escriba pruebas unitarias para su código relacionado con la base de datos utilizando frameworks como Jest o Mocha. Pruebe diferentes escenarios, incluyendo consultas exitosas y erróneas. Esta práctica ayuda a garantizar que su código se comporte como se espera y a detectar cualquier problema en las primeras etapas del desarrollo.

Exploraremos el manejo de errores de base de datos y las técnicas de depuración en Next.js con un ejemplo de código que utiliza TypeORM.

1. Manejo de errores de base de datos:

Suponiendo que está utilizando TypeORM como biblioteca ORM (Object-Relational Mapping) en su proyecto Next.js, puede manejar errores de base de datos de la siguiente manera:

```
```
const { getRepository } = require('typeorm');
const { Usuario } = require('../entidades/Usuario');

función asíncrona getUserById(id) {
 intentar {
 const userRepository = getRepository(Usuario);
 const usuario = await userepository.findOne(id);

 si (usuario) {
 lanzar nuevo Error('Usuario no encontrado');
 }

 usuario de retorno;
 } captura (error) {
 console.error('Error al recuperar el usuario:', error);
 lanzar nuevo Error('no se pudo recuperar el usuario');
 }
}
```

En este ejemplo, importamos la función `getRepository` de `typeorm` para recuperar el repositorio de la entidad "Usuario". Luego, usamos el repositorio para buscar un usuario por su ID. Si no se encuentra, se genera un error personalizado. Cualquier otro error de base de datos detectado se captura en el bloque `catch`, se registra y se vuelve a generar con un mensaje de error genérico.

## 2. Técnicas de depuración:

Ahora exploremos las técnicas de depuración en Next.js con TypeORM:Logging:

```
```
const { getRepository } = require('typeorm');
const { Usuario } = require('../entidades/Usuario');
```

```

función asíncrona getUserById(id) {
    intentar {
        const userRepository = getRepository(Usuario);
        const usuario = await userepository.findOne(id);

        si (usuario) {
            lanzar nuevo Error('Usuario no encontrado');
        }

        console.log('Usuario recuperado:', usuario);

        usuario de retorno;
    } captura (error) {
        console.error('Error al recuperar el usuario:', error);
        lanzar nuevo Error('no se pudo recuperar el usuario');
    }
}

```

En este ejemplo actualizado, añadimos una sentencia `console.log` para mostrar el usuario recuperado. Esto facilita la inspección de los datos y la verificación de si se está obteniendo el usuario correcto. Puede personalizar las sentencias de registro según sus necesidades de depuración.

b. Herramientas de depuración:

Next.js ofrece potentes herramientas de desarrollo y extensiones de navegador para la depuración, incluyendo React Developer Tools y Next.js DevTools. Estas herramientas permiten inspeccionar componentes, examinar propiedades y supervisar las solicitudes de red. Úselas para identificar posibles problemas con los componentes o el flujo de datos.

c. Optimización de consultas:

```

```
const { getRepository } = require('typeorm');
const { Usuario } = require('../entidades/Usuario');

```

```

función asíncrona getUsersWithPosts() {

```

```

 intentar {
 const userRepository = getRepository(Usuario);
 const users = await userepository.find({ relaciones: [publicaciones]
 });
 usuarios que regresan;
 } captura (error) {
 console.error('Error al recuperar usuarios:', error);
 lanzar nuevo Error(no se pudo recuperar usuarios);
 }
}

```

En este ejemplo, obtenemos usuarios junto con sus publicaciones asociadas usando la opción "relaciones" de TypeORM . Esto simplifica la recuperación de datos relacionados, pero si la consulta se vuelve lenta debido a una gran cantidad de usuarios o publicaciones, puede optimizarla aún más mediante el uso de paginación, carga ansiosa o la aplicación de filtros adicionales.

#### d. Pruebas unitarias:

Las pruebas unitarias son cruciales para detectar errores y garantizar la corrección del código. Aquí tienes un ejemplo con el framework de pruebas Jest:

```

const { getRepository } = require('typeorm');
const { Usuario } = require('../entidades/Usuario');
const { getUserId } = require('../usuario');
broma.mock('typeorm');
describe('getUserById', () => {
 test('devuelve un usuario cuando la consulta a la base de datos es exitosa',
 async()
 => {
 const usuario simulado = { id: 1, nombre: 'John Doe' };
 const findOneock = est.fn().mockesolvedvalue(mockser);
 obtener repositorio.mocketurnvalueOnce(
 findOne: findOneock,
);
 constante usuario = esperar getUserById(1);
 });
}

```

```
esperar(usuario).toEqual(mockUser);
esperar(findOneock).toaveeenalledimes(1);
esperar(findOneock).toaveeenalledWith(1);
});

test('lanza un error cuando no se encuentra el usuario', async () =>
{
 const findOneock = est.fn().mockesolvedvalue(undefined);
 obtener repositorio.mocketurnvalueOnce({
 findOne: findOneock,
 });

 esperar esperar(getseryd(1)).recta.tohrow('ser no
encontró');

 esperar(findOneock).toaveeenalledimes(1);
 esperar(findOneock).toaveeenalledWith(1);
});
});
```

En este ejemplo de prueba unitaria, simulamos la función `getRepository` y el método `findOne` de TypeORM utilizando las capacidades de simulación de Jest. Simulamos tanto una consulta exitosa como una consulta que devuelve `undefined` (usuario no encontrado). Esto garantiza que la función `getUserById` se comporte como se espera y gestione los errores de la base de datos correctamente.

Al emplear estas técnicas, puede manejar errores de base de datos de manera efectiva y depurar problemas en su aplicación Next.js usando TypeORM, lo que garantiza un funcionamiento fluido y una mejor experiencia de usuario.

## Mejores prácticas de seguridad de bases de datos en Next.js

En cuanto a las mejores prácticas de seguridad de bases de datos en Next.js, hay varias consideraciones clave a tener en cuenta. Analicémoslas en detalle:

**1. Protección de las credenciales de la base de datos:**

- Almacenar las credenciales de la base de datos de forma segura, como mediante el uso de variables de entorno o una solución de gestión de secretos seguros.
- Anule la codificación rígida de credenciales en su código fuente, archivos de configuración o repositorios de control de versiones.
- Restrinja el acceso a archivos confidenciales y asegúrese de que se establezcan los permisos adecuados.

**2. Implementación de la autenticación y autorización de usuarios:**

- Utilice mecanismos de autenticación seguros, como hash y sal de contraseñas, para proteger las credenciales de los usuarios.
- Implementar controles de acceso y autorización adecuados para garantizar que los usuarios sólo puedan acceder a los datos que están autorizados a ver o modificar.
- Validar y desinfectar las entradas de los usuarios para evitar errores de seguridad comunes. vulnerabilidades como ataques de inyección SQL.

**3. Cifrado y protección de datos:**

- Implemente Seguridad de la capa de transporte (TLS) para cifrar los datos transmitidos entre su aplicación y la base de datos.
- Considere implementar el cifrado en reposo para proteger datos confidenciales almacenados en la base de datos.
- Utilice bibliotecas de cifrado o funciones de cifrado integradas proporcionadas por su sistema de base de datos.

**4. Actualizaciones y parches periódicos:**

- Mantenga su sistema de base de datos y las bibliotecas asociadas actualizados con los últimos parches y actualizaciones de seguridad.
- Monitorear los avisos de seguridad y aplicar rápidamente parches para solucionar cualquier problema. vulnerabilidades identificadas.

**5. Validación de entrada y parametrización de consultas:**

- Validar y desinfectar las entradas de los usuarios para evitar que se acceda a datos maliciosos. ejecutado como parte de consultas de base de datos.
- Utilice consultas parametrizadas o sentencias preparadas para evitar la inyección de SQL ataques.
- Aproveche los generadores de consultas de bases de datos u ORM (mapeadores relacionales de objetos) que manejan la parametrización de consultas automáticamente.

**6. Auditoría y registro:**

- Habilite las funciones de registro y auditoría proporcionadas por su sistema de base de datos para rastrear y monitorear la actividad de la base de datos.

- Revisar y analizar periódicamente los registros de la base de datos para identificar cualquier información sospechosa, actividades o posibles violaciones de seguridad.

7. Limitar el acceso a la base de datos:

- Otorgar únicamente los privilegios necesarios a los usuarios de la base de datos, siguiendo las principios del principio del mínimo privilegio.
- Evite utilizar cuentas de base de datos privilegiadas para aplicaciones regulares.
- Implementar políticas de contraseñas sólidas y hacer cumplir la complejidad de las contraseñas normas.

8. Copias de seguridad periódicas y recuperación ante desastres:

- Realice copias de seguridad periódicas de su base de datos para garantizar la integridad de los datos y disponibilidad.
- Almacene copias de seguridad de forma segura y pruebe el proceso de restauración periódicamente.
- Tenga un plan de recuperación ante desastres para recuperarse rápidamente de cualquier pérdida de datos, infracciones o pérdidas.

9. Implementación y alojamiento seguros:

- Utilice plataformas o infraestructura de alojamiento seguras que ofrezcan sólidas medidas de seguridad y estándares de cumplimiento.
- Siga prácticas de implementación segura, como el uso de protocolos seguros (por ejemplo, HTTPS) y la implementación en servidores reforzados.

10. Evaluaciones y pruebas de seguridad periódicas:

- Realice evaluaciones de seguridad periódicas y pruebas de penetración para identificar vulnerabilidades en su base de datos y aplicación.
- Utilice herramientas de pruebas de seguridad para buscar vulnerabilidades comunes y configuraciones incorrectas.
- Involucre a expertos en seguridad para realizar auditorías de seguridad integrales de su infraestructura de aplicaciones y bases de datos.

Recuerde, la seguridad de la base de datos es un proceso continuo y es importante mantenerse actualizado con las últimas prácticas de seguridad, vulnerabilidades y recomendaciones del sistema de base de datos que está utilizando.

## Modelado de datos y diseño de esquemas

El modelado de datos y el diseño de esquemas implican el diseño de la estructura y las relaciones de los datos que se almacenarán en una base de datos. TypeORM es un

Biblioteca R de mapeo objeto-relacional que simplifica el trabajo con bases de datos en TypeScript y JavaScript.

Aquí están los términos y conceptos clave que debes comprender:

- Entidad:

Una entidad representa una tabla o colección de base de datos en TypeORM. Se define como una clase de TypeScript decorada con el decorador `@Entity()`. Cada entidad corresponde a un documento de fila en la tabla de base de datos. recopilación.

- Columna:

Una columna representa un campo o atributo de una entidad. Se define mediante el decorador `@Column()`. Las columnas almacenan valores de datos, como cadenas, números, valores booleanos o fechas.

- Clave principal:

La clave primaria identifica de forma única cada documento de fila de una entidad. Garantiza que cada entrada sea única. En ypeR, las claves primarias se definen mediante el decorador `'@PrimaryGeneratedColumn()'` y suelen usar números autoincrementales.

- Relaciones:

Las relaciones definen cómo se relacionan las entidades entre sí. ypeR admite varios tipos de relaciones, entre ellas:

Uno a uno: una entidad está asociada con otra entidad en una relación uno a uno.

Uno a muchos: una entidad está asociada con varias entidades en una relación de uno a muchos.

Muchos a uno: varias entidades se asocian con una entidad en una relación de muchos a uno.

Muchos a muchos: varias entidades se asocian con varias entidades en una relación de muchos a muchos.

Estas relaciones se establecen utilizando decoradores como `eoe`, `@eoay`, `ayoe` y `ayoay`.

- Migración:

La migración es una forma de gestionar los cambios en el esquema de la base de datos a lo largo del tiempo. Cuando su aplicación evoluciona y es necesario actualizar el esquema de la base de datos, las migraciones le ayudan a realizar dichos cambios conservando los datos existentes. TypeORM proporciona herramientas para crear, ejecutar y administrar migraciones.

- Consultas:

Consultar es el proceso de recuperar datos de la base de datos según criterios específicos. TypeORM ofrece un generador de consultas que permite construir consultas mediante una cadena de métodos y condiciones. Esto facilita la creación de consultas complejas sin necesidad de escribir sentencias SQL sin formato.

- Transacciones:

Las transacciones garantizan la atomicidad y consistencia de las operaciones de la base de datos. Una transacción agrupa un conjunto de operaciones de la base de datos en una sola unidad, que puede tener éxito en su totalidad o fallar. Garantiza que, si alguna parte de la transacción falla, se reviertan todos los cambios, manteniendo así la integridad de los datos.

- ORM (Mapeo Objeto-Relacional):

ORM es una técnica de programación que asigna objetos de una aplicación a tablas de una base de datos relacional. Simplifica las operaciones de la base de datos al permitir a los desarrolladores trabajar con objetos y realizar operaciones CRUD sin escribir consultas SQL explícitamente.

TypeORM es una biblioteca ORM que proporciona un conjunto de herramientas y características para mapear entidades, definir relaciones, realizar operaciones de base de datos y manejar migraciones en una aplicación.

En resumen, el modelado de datos y el diseño de esquemas con TypeORM implican la creación de entidades, la definición de columnas y relaciones, la gestión de migraciones para cambios de esquema, la consulta de datos mediante el generador de consultas y la garantía de la integridad transaccional. Estos conceptos le ayudan a diseñar estructuras de bases de datos eficientes, escalables y mantenibles para sus aplicaciones.

Profundicemos en el modelado de datos y el diseño de esquemas con TypeORM, cubriendo todos los tipos de operaciones, incluidas operaciones CRUD (Crear, Leer, Actualizar, Eliminar), consultas, transacciones y migraciones.

1. Configuración de TypeORM:

Comience instalando las dependencias necesarias usando npm o yarn:

```
npm install typeorm reactmetadata
```

...

- . Configuración

Cree un archivo typeorm.config.js en el directorio raíz del proyecto para definir los parámetros de conexión a la base de datos. Por ejemplo, aquí hay un

configuración para una base de datos PostgreSQL

```
módulo.exportaciones = {
 tipo: postgres,
 anfitrón: localhost,
 puerto: 5432,
 nombre de usuario: su nombre de usuario,
 contraseña: tucontraseña,
 base de datos: subbase de datos,
 sincronizar: verdadero,
 registro: verdadero,
 entidades: [src/entidades/**/*.ts],
 migraciones: [src/migrations/**/*.ts],
 suscriptores: [src/subscribers/**/*.ts],
 cli: {
 entidadesDir: src/entidades,
 migracionesDir: src/migraciones,
 suscriptoresDir: src/suscriptores,
 },
};

```

```

3. Creación de entidades:

Cree clases de entidad con TypeScript y decórelas con decoradores TypeORM. Consideremos el ejemplo de una aplicación de blog con dos entidades: Usuario y Publicación. Cada usuario puede tener varias publicaciones, y cada publicación pertenece a un usuario.

```
importar { Entidad, ColumnaGeneradaPrimaria, Columna,
    Unooany,
    cualquieraUno, oinolumn } de typeorm;
@Entidad()
clase de exportación Usuario {
    @PrimaryGeneratedColumn()
    id: numero;
```

```

    @Columna()
    nombre: cadena;
    @Oneoany(() => ost, post => post.usuario)
    publicaciones: Post[];
}

@Entidad()
clase de exportación Post {
    @PrimaryGeneratedColumn()
    id: numero;
    @Columna()
    título: cadena;
    @Column({ tipo: 'texto' })
    contenido: cadena;
    @anyoOne(() => ser, usuario => usuario.posts)
    @oinolumn({ nombre: id de usuario })
    usuario: Usuario;
}

```

En este ejemplo, el decorador `@Entity()` marca la clase como una entidad TypeORM. El decorador `@PrimaryGeneratedColumn()` define la clave principal de la entidad. El decorador `@Column()` especifica las columnas de la tabla, y los decoradores `eoay` y `@ayoe` establecen relaciones entre las entidades `ser` y `Post`.

4. Conexión a la base de datos:

Cree una conexión a la base de datos utilizando la configuración de TypeORM y las clases de entidad. Aquí hay un ejemplo.

```

```importar { createConnection } desde 'typeorm';
importar { Usuario } desde './entidades/Usuario';
importar { Post } desde './entities/Post';
```

```

```

función asíncrona connectoDatabase() {

    intentar {

        const conexión = await createConnection();
        console.log('Conectado a la base de datos');

        // La lógica de aplicación adicional va aquí

        esperar conexión.close();
        console.log('Conexión cerrada');

    } captura (error) {
        console.error('Error al conectar con la base de datos:', error);

    }

}

conectarBaseDeDatos();
```

```

## 5. Operaciones CRUD:

typeorm simplifica las operaciones de CRD utilizando su EntityManager.

A continuación se muestran ejemplos de cada

operación:

- Crear un nuevo usuario y guardarlo en la base de datos:

```

```
const usuario = nuevo Usuario();
```

```
usuario.nombre = 'John Doe';
```

```
await connection.manager.save(usuario);
```

```
console.log('Usuario guardado:', usuario);
```

```

- Recuperar todos los usuarios de la base de datos:

```
const usuarios = await conexión.manager.find(ser);
```

```
console.log('Todos los usuarios:', usuarios);
````
```

- Actualizar un usuario:

```
````
```

```
const usuario = await conexión.manager.findOne(ser, { id: 1 });
user.name = 'Nombre actualizado';
```

```
await connection.manager.save(usuario);
console.log('Usuario actualizado:', usuario);
````
```

- Eliminar un usuario:

```
const usuario = await conexión.manager.findOne(ser, { id: 1 });
```

```
esperar conexión.manager.remove(usuario);
console.log('Usuario eliminado');
````
```

## 6. Consulta:

TypeORM proporciona un potente generador de consultas para consultas complejas.

He aquí un ejemplo:

```
const users = await conexión.manager
 .createQueryBuilder(Usuario,
 'usuario') .leftJoinAndSelect('usuario.publicaciones',
 'publicación') .where('usuario.id = :userId', { userId: 1 })
 .getMany();
```

```
console.log('Usuario y sus publicaciones:', users);
````
```

Esta consulta obtiene un usuario con sus publicaciones asociadas mediante una unión izquierda.

7. Transacciones:

TypeORM admite transacciones para garantizar la atomicidad y la consistencia. A continuación, un ejemplo:

```
...
    constante queryRunner =
conexión.createQueryRunner();
esperar queryRunner.connect();
esperar ueryunner.startransaction();

intentar {
    const usuario = nuevo Usuario();
    usuario.nombre = 'Jane Smith';

    esperar queryRunner.manager.save(usuario);

    const post = nueva publicación();
    post.title = 'Nueva publicación';
    post.content = 'Lorem ipsum dolor sit amet';

    post.usuario = usuario;
    esperar queryRunner.manager.save(publicar);

    esperar ueryunner.committransaction();
    console.log(transacción confirmada);
} captura (error) {
    esperar ueryunner.rollbacktransaction();
    console.log(transacción revertida);
} finalmente {
    esperar queryRunner.release();
}
```

Este ejemplo muestra una transacción que guarda un nuevo usuario y una publicación asociada a él. Si alguna operación falla, la transacción se revierte.

8. Migraciones:

Ye simplifica las migraciones de esquemas de base de datos. Aquí hay un ejemplo de creación y ejecución de una migración:

Migración de tipo: crear un reutilizable

Este comando genera un nuevo archivo de migración. Dit el archivo generado para definir los cambios necesarios.

```
importar {igrationinterface, ueryrunner} desde typeorm;
clase de exportación reateserable161234561 implementa
Interfaz de migración {
```

```
    público asíncrono up(queryRunner: QueryRunner): Promise<void> {
```

```
        esperar queryRunner.query(`
```

Usuario de EE E (

id SE ,

nombre personaje variable O ,

3ddbfce4ac151b

DCB E (identificación)

)

`);

}

```
    público asíncrono inactivo (queryRunner: QueryRunner): Promise<void> {
```

espera ueryrunner.uery(DO E usuario);

}

}

Para ejecutar la migración:

migración de typeorm:run

Estos ejemplos proporcionan una descripción general del modelado de datos y el diseño de esquemas con TypeORM, que cubre varias operaciones como CRUD, consultas, transacciones,

y migraciones. La documentación de TypeORM ofrece más detalles sobre funciones avanzadas y opciones de personalización.

Escalar la base de datos para lograr rendimiento y alta disponibilidad

Escalar una base de datos para lograr rendimiento y alta disponibilidad implica optimizar la infraestructura de la base de datos para gestionar una mayor carga de trabajo y garantizar que permanezca accesible incluso en caso de fallos. A continuación, se ofrece una explicación detallada para usuarios principiantes:

- Escalado de bases de datos:

El escalado de bases de datos se refiere al proceso de aumentar la capacidad y las capacidades de un sistema de bases de datos para gestionar volúmenes de datos y solicitudes de usuarios crecientes.

Existen dos tipos principales de escalado:

Escalado vertical (escalamiento vertical): En el escalado vertical, el servidor de base de datos se actualiza con recursos de hardware más potentes, como CPU, memoria o almacenamiento, para gestionar el aumento de carga. Esto implica actualizar el servidor existente o migrar a uno más potente.

Escalado horizontal (escalado horizontal): El escalado horizontal implica añadir más servidores de bases de datos para distribuir la carga de trabajo entre varias máquinas. Esto se logra mediante técnicas como la fragmentación o la replicación de bases de datos.

- Optimización del rendimiento:

Para mejorar el rendimiento de la base de datos, considere las siguientes técnicas:

Indexación: cree índices apropiados en columnas consultadas con frecuencia para acelerar las operaciones de recuperación de datos.

Optimización de consultas : Analizar y optimizar las consultas de la base de datos para minimizar su tiempo de ejecución. Esto puede implicar la reescritura de consultas, el uso de estrategias de unión adecuadas o la optimización del modelo de datos.

Almacenamiento en caché: implemente una capa de almacenamiento en caché para almacenar en la memoria datos a los que se accede con frecuencia, lo que reduce la necesidad de consultas a la base de datos.

Desnormalización: en algunos casos, desnormalizar el esquema de la base de datos (reduciendo la cantidad de uniones) puede mejorar el rendimiento al reducir la complejidad de las consultas.

Particionado: divide tablas grandes en fragmentos más pequeños y manejables según criterios específicos, por ejemplo, rango, lista o hash. Esto permite una mejor distribución de datos y un acceso más rápido.

- Alta disponibilidad:

La alta disponibilidad garantiza que la base de datos permanezca accesible incluso ante fallos.

Las técnicas clave para lograrla incluyen:

Replicación : Mantener múltiples copias de la base de datos en diferentes servidores. La replicación puede ser síncrona (en tiempo real) o asíncrona (con retardo), según el nivel deseado de consistencia y rendimiento de los datos.

Comutación por error: Implementar mecanismos para cambiar automáticamente a un servidor en espera si el servidor principal falla. Esto requiere la monitorización continua del estado de la base de datos y la capacidad de redirigir rápidamente el tráfico entrante.

Balanceo de carga: Distribuya las solicitudes entrantes entre varios servidores de bases de datos para distribuir la carga de trabajo de forma uniforme. Los平衡adores de carga supervisan el estado del servidor y dirigen el tráfico a los recursos disponibles.

Recuperación ante desastres : Cree copias de seguridad periódicas de la base de datos y almacénelas en ubicaciones externas. En caso de una falla grave o un desastre, estas copias de seguridad pueden utilizarse para restaurar la base de datos a un estado anterior.

Redundancia: utilice componentes redundantes como fuentes de alimentación, conexiones de red y dispositivos de almacenamiento para minimizar los puntos únicos de falla.

- Monitoreo y Optimización:

Supervisar continuamente el rendimiento y la salud del sistema de base de datos.

Esto implica el seguimiento de métricas como el uso de CPU, la utilización de memoria, el disco duro y el tráfico de red. Analice estas métricas para identificar cuellos de botella y áreas de optimización. Revise y ajuste periódicamente la configuración de la base de datos según los patrones de rendimiento observados.

En conclusión, escalar una base de datos para lograr rendimiento y alta disponibilidad implica optimizar la infraestructura de la base de datos, implementar técnicas de mejora del rendimiento, garantizar la redundancia de datos y mantener mecanismos de comutación por error para mantener la base de datos accesible y confiable incluso durante períodos de alta demanda o fallas.

Conclusión

Este capítulo ofreció una exploración exhaustiva de las bases de datos y su función en las aplicaciones Next.js. Comenzamos por comprender los fundamentos de las bases de datos y su importancia en el desarrollo web. Posteriormente, profundizamos en diferentes tipos de bases de datos, incluyendo las relacionales, NoSQL y gráficas.

Se destacaron sus características únicas y casos de uso. Se analizó el proceso de toma de decisiones para seleccionar la base de datos adecuada, considerando factores como la estructura de datos, la escalabilidad y el rendimiento.

Una vez seleccionado el tipo de base de datos, el capítulo guió a los lectores a través del proceso de configuración de una conexión de base de datos en Next.js, centrándose específicamente en bases de datos populares como MongoDB, MySQL y PostgreSQL. Se explicaron las operaciones comunes que se realizan con bases de datos, como las operaciones CRUD, mostrando cómo interactuar con la base de datos seleccionada, manipular datos y recuperar información usando Next.js.

Además, el capítulo abordó aspectos cruciales como la gestión de errores en las bases de datos, la implementación de técnicas de depuración eficaces y la garantía de la seguridad de las bases de datos. Se proporcionaron las mejores prácticas para la resolución de problemas comunes, la protección de la información confidencial y la prevención de vulnerabilidades.

Se exploraron el modelado de datos y el diseño de esquemas para ayudar a los lectores a diseñar estructuras de bases de datos eficientes, crear relaciones entre entidades y optimizar consultas para un mejor rendimiento.

Por último, se discutieron estrategias para escalar bases de datos para satisfacer las demandas de una creciente base de usuarios, garantizando alta disponibilidad y un mejor rendimiento.

Se presentaron ejemplos prácticos y casos de uso a lo largo del capítulo para ilustrar los conceptos tratados.

Al adquirir los conocimientos y las herramientas que se presentan en este capítulo, los lectores podrán aprovechar diferentes tipos de bases de datos en sus aplicaciones Next.js, lo que les permitirá crear aplicaciones web modernas y robustas que aprovechen el poder de los datos. ¡Embárquese juntos en este viaje hacia las bases de datos y descubra todo el potencial de Next.js!

En el próximo capítulo, nos sumergiremos en el fascinante mundo del renderizado en Next.js. Exploraremos las diferentes técnicas de renderizado, como el renderizado del lado del servidor (SSR), el renderizado del lado del cliente (CSR) y la generación de sitios web estáticos (SS), comprendiendo sus beneficios y casos de uso.

Preguntas de opción múltiple

1. ¿Cuál de los siguientes no es un tipo de base de datos analizado en este capítulo?
 - A. Base de datos relacional
 - B. Base de datos NoSQL

C. Base de datos de gráficos

D. Base de datos de blockchain

2. ¿Qué factores deben tenerse en cuenta al seleccionar la base de datos adecuada para una aplicación?

A. Estructura de datos, escalabilidad y rendimiento

B. Diseño de interfaz de usuario y esquemas de color

C. Estrategias de marketing y público objetivo

D. Capacidad de respuesta del sitio web y velocidad de carga

3. ¿Qué bases de datos se mencionan como opciones populares para conectarse con

¿Siguiente.js?

A. MongoDB, MySQL y PostgreSQL

B. SQLite, Redis y Firebase

C. Oracle, DB2 e Informix

D. Cassandra, Couchbase y DynamoDB

4. ¿Qué significan las operaciones CRUD?

A. Crear, renderizar, actualizar, eliminar

B. Copiar, renombrar, cargar, descargar

C. Crear, Leer, Actualizar, Eliminar

D. Compilar, ejecutar, cargar, depurar

5. ¿Qué aspecto se aborda en este capítulo respecto a la seguridad de las bases de datos?

A. Construcción de sistemas de autenticación de usuarios

B. Implementación de la validación de formularios en el frontend

C. Prevención de ataques de inyección SQL

D. Optimización de consultas de bases de datos

Respuestas

| | |
|----|----|
| 1. | d |
| 2 | a |
| 3 | a |
| 4 | do |
| 5 | do |

Al explorar términos como operaciones CRUD, SQL, NoSQL, ACID y API, los lectores pudieron reforzar su comprensión de estos conceptos y comprender los principios fundamentales de la gestión de bases de datos. Estas preguntas interactivas les permitieron consolidar sus conocimientos y mejorar su comprensión.

Las preguntas de opción múltiple sirvieron como una herramienta valiosa para evaluar la comprensión y retención de conceptos clave por parte de los lectores, preparándolos para aplicar sus conocimientos en escenarios del mundo real.

Capítulo 10

Comprensión de la renderización en aplicaciones Next.js

Introducción

En el mundo del desarrollo web, el renderizado juega un papel crucial para ofrecer contenido dinámico e interactivo a los usuarios. Dos técnicas de renderizado populares, el renderizado del lado del cliente y el renderizado del lado del servidor, se han convertido en enfoques esenciales para la creación de aplicaciones web. En este capítulo, exploraremos estos métodos de renderizado y su importancia en el contexto de las aplicaciones Next.js.

Para comenzar, ofreceremos una descripción general completa del renderizado del lado del cliente y del lado del servidor, profundizando en sus conceptos fundamentales y destacando sus diferencias clave. Al comprender los principios básicos de cada enfoque, obtendrá una base sólida para tomar decisiones informadas sobre estrategias de renderizado.

A continuación, exploraremos por qué la renderización del lado del servidor es un aspecto fundamental de Next.js. Examinaremos cómo la renderización del lado del servidor puede mejorar el rendimiento, optimizar la optimización para motores de búsqueda (SEO) y optimizar la experiencia general del usuario. Mediante ejemplos reales y perspectivas prácticas, ilustraremos los beneficios de aprovechar la renderización del lado del servidor en el framework Next.js.

Sin embargo, la renderización del lado del cliente también tiene sus ventajas, especialmente en escenarios donde el contenido dinámico y la interactividad son primordiales. Exploraremos los diferentes escenarios donde la renderización del lado del cliente destaca y demostraremos cómo se puede implementar eficazmente en aplicaciones Next.js. Desde la dinámica...

Renderizado del lado del cliente para un uso óptimo: este capítulo le brindará el conocimiento necesario para aprovechar las capacidades de renderizado del lado del cliente en Next.js.

Para ayudarlo a tomar decisiones informadas, analizaremos las mejores prácticas para elegir entre la representación del lado del cliente y del lado del servidor en aplicaciones Next.js. Exploraremos los factores a considerar al seleccionar la representación adecuada.

Estrategia y proporcionar pautas para optimizar el rendimiento de la representación.

Los temas tratados en este capítulo incluyen la comprensión de los conceptos básicos de la representación del lado del cliente y la representación del lado del servidor, los beneficios y las desventajas de cada enfoque, el enfoque de Next.js para la representación, la representación del lado del servidor con Next.js, la representación del lado del cliente con Next.js, la representación dinámica del lado del cliente con Next.js, cuándo usar la representación del lado del cliente versus la representación del lado del servidor y las mejores prácticas para usar estas técnicas de representación en aplicaciones Next.js.

Al finalizar este capítulo, comprenderá a fondo el renderizado del lado del cliente y del lado del servidor, y cómo se pueden utilizar eficazmente en aplicaciones Next.js. Con este conocimiento, podrá tomar decisiones informadas y optimizar el rendimiento del renderizado de sus proyectos Next.js. Así que, profundicemos en el fascinante mundo del renderizado en Next.

js

Estructura

En este capítulo cubriremos los siguientes temas:

- Comprender los conceptos básicos de la renderización del lado del cliente y del lado del servidor.
representación
- Beneficios y desventajas de la renderización del lado del cliente y del lado del servidor •
Enfoque de Next.js para la renderización del lado del cliente y del lado del servidor
- Renderización del lado del servidor con Next.js •
Renderización del lado del cliente con Next.js •
- Renderización dinámica del lado del cliente con Next.js
• Cuándo usar la renderización del lado del cliente y cuándo usar la renderización del lado del servidor • Mejores prácticas para usar la renderización del lado del cliente y del lado del servidor en Next.js
aplicaciones

Comprender la representación en Next.js

Next.js es un popular framework de React que simplifica el proceso de creación de aplicaciones web SSR renderizadas del lado del servidor y estáticas. Ofrece una forma intuitiva de gestionar el renderizado mediante un enfoque híbrido.

Con Next.js, tiene la flexibilidad de elegir entre diferentes métodos de renderizado según los requisitos de su proyecto. Puede optar por el renderizado del lado del cliente (CSR), donde la página se renderiza inicialmente en el lado del cliente mediante avaScript. Este enfoque proporciona interactividad, pero puede resultar en una carga inicial de la página más lenta.

Next.js también admite la renderización SSR del lado del servidor, donde la página se renderiza en el servidor y se envía al cliente completamente renderizada. SSR ofrece un mejor rendimiento y ventajas de S, ya que el contenido está disponible para los motores de búsqueda.

Otra opción es la generación de sitios estáticos (SS), donde las páginas se prerenderizan como archivos estáticos durante el proceso de compilación. Este enfoque ofrece un rendimiento excelente, ya que no hay procesamiento del lado del servidor durante la ejecución. SS es ideal para sitios web con contenido que no cambia con frecuencia, como blogs o sitios de documentación.

Next.js combina lo mejor de SSR y SS, lo que le permite elegir la estrategia de renderizado por página. Esta flexibilidad permite a los desarrolladores crear aplicaciones web altamente optimizadas, rápidas y compatibles con S.

Comprendamos los conceptos básicos de la renderización del lado del cliente, la renderización del lado del servidor y la generación de sitios estáticos:

- Representación del lado del cliente CSR

La renderización del lado del cliente implica cargar una página HTML básica desde el servidor y luego usar avaScript para obtener y renderizar dinámicamente los datos en el navegador del cliente. El proceso de renderización del lado del cliente generalmente implica realizar solicitudes PI para recuperar datos y actualizar el modelo de objeto D Document con los datos recibidos. Este enfoque permite una experiencia de usuario más interactiva y dinámica, ya que el contenido se puede actualizar sin tener que refrescar toda la página. Sin embargo, supone una mayor carga para el dispositivo del cliente, ya que el navegador debe gestionar tanto la renderización como la obtención de datos.

- Representación del lado del servidor SSR

Por otro lado, la renderización del lado del servidor implica generar el contenido en el propio servidor y enviar el HTML pre-renderizado al navegador del cliente. En SSR, el servidor procesa la solicitud, recupera los datos necesarios y genera una página HTML completa que incluye el contenido solicitado. Este HTML pre-renderizado se envía al cliente, lo que reduce la carga de su dispositivo. SSR es beneficioso para los motores de búsqueda.

optimización de ya que el contenido está disponible para los rastreadores de motores de búsqueda (SEO). Sin embargo, la carga inicial de la página puede ser más lenta en comparación con CSR, ya que el servidor necesita procesar la solicitud y generar la .

- Generación de sitios estáticos SS

La generación de sitios estáticos (SS) es una técnica de renderizado de sitios web donde las páginas se prerenderizan durante la compilación y se entregan como archivos estáticos a los clientes. Con SS, el contenido se genera una sola vez durante el proceso de compilación y no requiere procesamiento del servidor durante la ejecución. Este enfoque ofrece varias ventajas, como un mejor rendimiento, menor carga del servidor y mayor seguridad. También permite una mejor optimización para motores de búsqueda (S), ya que estos pueden rastrear e indexar fácilmente las páginas estáticas. SS se utiliza comúnmente en sitios web con contenido que no cambia con frecuencia, como blogs, sitios de documentación y páginas de marketing.

Diferencias clave entre RSE y RSE

- Carga de página inicial: SSR proporciona una página completamente renderizada desde el servidor, lo que permite una renderización inicial más rápida, mientras que CSR requiere solicitudes y procesamiento adicionales en el lado del cliente.
- S SSR es más amigable con los S ya que los motores de búsqueda pueden rastrearlo fácilmente y, indexar el contenido pre- , mientras que CSR requiere esfuerzos adicionales. renderizado para que los motores de búsqueda puedan descubrirlo.
- La CSR de rendimiento puede proporcionar una experiencia más rápida e interactiva una vez cargada la página inicial, ya que las actualizaciones posteriores pueden gestionarse en el lado del cliente sin necesidad de recargar la página completa. La SSR puede tener un tiempo de carga inicial más largo, pero ofrece una mejor experiencia de usuario para dispositivos más lentos o con malas condiciones de red.
- Complejidad del código: CSR requiere más código JavaScript del lado del cliente para obtener y renderizar datos, mientras que SSR traslada esta responsabilidad al lado del servidor, lo que reduce la complejidad en el cliente.

La Tabla 10.1 presenta la diferencia entre SSR y CSR

| Característica | Representación del lado del servidor (SSR) | Representación del lado del cliente (CSR) |
|------------------------|--|---|
| Representación Proceso | Pre-renderizado de servidor en el | Representación y obtención de datos en el lado del cliente mediante S |

| | | |
|---------------------------------|---|---|
| Carga inicial
Tiempo | Renderizado inicial más rápido | Representación inicial más lenta debido a solicitudes adicionales |
| S | Compatible con S, ya que el contenido está fácilmente disponible para indexar | Requiere esfuerzos adicionales para que los motores de búsqueda puedan descubrirlo |
| Actuación | Mejor experiencia de usuario en dispositivos más lentos o con malas condiciones de red | Experiencia más rápida e interactiva una vez cargada la página inicial |
| Código
Complejidad | código avaScript del lado del cliente de ess | código avaScript del lado del cliente |
| Obtención de datos | No se requieren solicitudes de PI adicionales o limitadas | Solicitudes de PI adicionales para la obtención y representación de datos |
| Desarrollo
Complejidad | El servidor gestiona la representación, lo que reduce la complejidad del lado del cliente | El cliente gestiona la representación, lo que aumenta la complejidad del lado del cliente |
| Almacenamiento en caché | Es más fácil implementar el almacenamiento en caché del lado del servidor | El almacenamiento en caché requiere lógica adicional del lado del cliente |
| Casos de uso | Sitios web con mucho contenido, S- aplicaciones enfocadas | Aplicaciones interactivas y dinámicas, aplicaciones de página única SP |
| Marcos Next.js, Nuxt.js y ue.js | | React, ngular, ue.js, mber.js, etc. |

Tabla 10.1: RSE vs. RSE

Tenga en cuenta que el uso de SSR o CSR depende de diversos factores y requisitos de su aplicación específica. Es habitual combinar ambas técnicas en una misma aplicación, aprovechando las ventajas de cada enfoque donde resulte más adecuado. Frameworks como Next.js ofrecen la flexibilidad de elegir entre SSR y CSR según componentes o rutas específicas dentro de la aplicación.

beneficios y materias primas de Can

CSR y SSR tienen sus respectivos beneficios y desventajas, y comprenderlos puede ayudar a los desarrolladores a tomar decisiones informadas en función de sus casos de uso específicos. Exploraremos las ventajas y desventajas de cada enfoque de renderizado en detalle:

beneficios de C

- La experiencia de usuario mejorada (CSR) permite una experiencia de usuario fluida e interactiva al permitir actualizaciones dinámicas de contenido sin tener que actualizar toda la página. Ofrece una experiencia más similar a la de una aplicación, ya que los usuarios pueden interactuar con ella en tiempo real.
- Rendimiento mejorado después de la carga inicial una vez que la carga inicial cargan CSS y avaScript, y las actualizaciones posteriores y la obtención de datos pueden realizarse de forma asíncrona sin necesidad de recargar la página completa. Esto se traduce en tiempos de respuesta más rápidos y una experiencia de usuario fluida.
- Interacciones front-end enriquecidas CSR es ideal para aplicaciones que dependen en gran medida de la interactividad del lado del cliente y del contenido dinámico, como paneles de control en tiempo real, herramientas colaborativas o feeds de redes sociales.

Dibujadas o C:

- Tiempo de carga inicial más lento: CSR requiere descargar el paquete avaScript inicial y realizar solicitudes PI adicionales para obtener datos. Esto puede generar tiempos de carga de página inicial más lentos en comparación con la representación del lado del servidor.
- Desafíos de los motores de búsqueda Los rastreadores de motores de búsqueda pueden enfrentar dificultades para indexar y comprender el contenido representado por el cliente. Es posible que sean necesarias medidas adicionales, como la representación del lado del servidor de páginas críticas o el uso de técnicas como la representación previa, para garantizar un rendimiento adecuado de los motores de búsqueda.

Mayor complejidad del lado del cliente. La CSR requiere más código del lado del cliente para gestionar la obtención de datos, la renderización y la gestión del estado. Esta complejidad puede dificultar el desarrollo y la depuración.

beneficios de

- Página inicial más rápida carga SSR envía pre-renderizados desde el servidor,

Esto resulta en tiempos de carga de página inicial más rápidos. Esto es especialmente ventajoso para sitios web o aplicaciones con mucho contenido, donde la carga rápida es crucial.

- S-Friendly Los motores de búsqueda pueden rastrear e indexar fácilmente el contenido pre-renderizado, lo que hace que SSR sea beneficioso para un mejor SEO. Garantiza que el contenido esté fácilmente disponible para que los motores de búsqueda lo comprendan y clasifiquen.
- Graceful Degradation SSR maneja con elegancia los escenarios en los que JavaScript está deshabilitado o no es compatible con el dispositivo o navegador del cliente, lo que garantiza que los usuarios aún puedan acceder y ver el contenido.

Dibujadas o

- El aumento de la carga del servidor (SSR) supone una mayor carga para el servidor, ya que necesita generar el HTML para cada solicitud. Esto puede afectar el rendimiento y la escalabilidad del servidor, especialmente en aplicaciones con alto tráfico o que consumen muchos recursos.
- Interactividad limitada del lado del cliente SSR se basa en información generada por el servidor, lo que significa que ciertos tipos de interactividades que requieren procesamiento del lado del cliente pueden ser limitadas en comparación con CSR.
- Complejidad de desarrollo y lógica de renderizado del lado del servidor: SSR requiere que los desarrolladores manejen la lógica de renderizado en el lado del servidor, lo que puede requerir codificación y configuración adicionales del lado del servidor en comparación con CSR.

Es importante destacar que se puede emplear un enfoque híbrido, que utiliza tanto CSR como SSR selectivamente según componentes o rutas específicas, para aprovechar las ventajas de cada método de renderizado y mitigar sus inconvenientes. Frameworks como Next.js ofrecen opciones de renderizado híbrido, lo que ofrece flexibilidad y posibilidades de optimización.

El enfoque de Next.js hacia la RSE y la RSC

Next.js es un popular framework de React que ofrece un enfoque basado en principios para la CSR y la SSR. Simplifica la implementación de ambas estrategias de renderizado, ofreciendo a los desarrolladores un potente conjunto de herramientas para crear aplicaciones web eficientes.

Exploraremos en detalle el enfoque de Next.js para la CSR y la SSR.

- energía misteriosa con Next.js:

Next.js simplifica la renderización del lado del servidor al permitir a los desarrolladores escribir componentes que se puedan renderizar en el servidor. Cuando se realiza una solicitud a una aplicación Next.js, el servidor obtiene los datos necesarios y renderiza el componente. El HTML resultante se envía al cliente para su visualización. Este enfoque ofrece varias ventajas.

Optimización SSR de Next.js permite que los rastreadores de motores de búsqueda descubran e indexen contenido fácilmente, mejorando la optimización de motores de búsqueda.

Carga inicial más rápida Al pre-renderizar las páginas en el servidor, Next.js entrega una página HTML completa al cliente, lo que da como resultado tiempos de carga inicial más rápidos.

La degradación SSR de raceful garantiza que las páginas sean accesibles incluso cuando avaScript está deshabilitado, lo que proporciona una mejor experiencia de usuario para una gama más amplia de dispositivos.

- Cliente en C con Next.js:

Next.js también es compatible con CSR, lo que permite renderizar componentes en el lado del cliente mediante avaScript. Este enfoque es adecuado para escenarios donde el contenido dinámico y la interactividad son primordiales. Next.js aprovecha las capacidades de React para la renderización en el lado del cliente, lo que permite la actualización eficiente de componentes sin recargar toda la página. Algunas ventajas de CSR con Next.js incluyen:

Experiencia de servicio mejorada CSR permite actualizaciones e interactividad en tiempo real, lo que proporciona una experiencia de usuario más atractiva.

Contenido dinámico Next.js permite que los componentes obtengan datos en el lado del cliente, lo que facilita la representación de contenido dinámico y el manejo de las interacciones del usuario.

División de código y carga diferida Next.js ofrece capacidades integradas de división de código y carga diferida, optimizando el rendimiento al cargar solo los componentes necesarios cuando es necesario.

renderizing con Next.js

Next.js ofrece un enfoque de renderizado híbrido que permite a los desarrolladores elegir entre SSR y CSR por página o por componente. Esta flexibilidad permite a los desarrolladores lograr un equilibrio entre el rendimiento, la interactividad y los requisitos de S.

De forma predeterminada, Next.js utiliza SSR para la carga inicial de la página, lo que permite una representación más rápida y un S mejorado. La navegación posterior dentro de la aplicación puede usar CSR, lo que garantiza una experiencia de usuario fluida e interactiva.

Next.js también optimiza el proceso de renderizado mediante técnicas como la división automática de código, el almacenamiento en caché inteligente y la regeneración estática incremental. Estas funciones ayudan a mejorar el rendimiento y garantizan un renderizado eficiente del contenido.

En resumen, Next.js simplifica la implementación de SSR y CSR al proporcionar un marco que integra perfectamente el lado del servidor y el lado del cliente.

Renderizado. Ofrece flexibilidad para elegir el enfoque de renderizado adecuado según los requisitos específicos, lo que lo convierte en una herramienta potente para crear aplicaciones web eficientes y de alto rendimiento.

con Next.js

Aquí hay una explicación detallada con ejemplos de código de las funciones utilizadas para lograr SSR con

Next.js .getererieros

La función `getServerSideProps` permite obtener datos del servidor y pasarlo como propiedades a los componentes. Se ejecuta en cada solicitud y obtiene los datos dinámicamente. Aquí tienes un ejemplo.

```
importar React desde 'react';
función Página de inicio({ datos }) {
    devolver (
        <div>
            ¡Bienvenido a mi aplicación Next.js!
            Datos de la API: {data}
        </div>
    );
}

exportar función asíncrona getServerSideProps() { const
    res = await fetch('https://api.example.com/data'); const data = await
    res.json();

    devolver {
        accesorios: {
            datos,
        },
    };
}
```

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

En el ejemplo anterior, la función `getServerSideProps` obtiene datos de un punto final de PI `httpsapi.example.comdata`. Los datos obtenidos se pasan como datos de propiedad al componente `omePage`, que luego se puede representar con los datos representados por el servidor.

### .gettatiros

La función `getStaticProps` permite la renderización del lado del servidor con generación estática. Obtiene datos durante la compilación y genera páginas HTML estáticas. El HTML generado se reutiliza luego en cada solicitud, lo que proporciona un mejor rendimiento. Aquí hay un ejemplo.

```
```importar React desde 'react';  
función Página de inicio({ datos }) {  
    devolver (  
        <div>  
            ¡Bienvenido a mi aplicación Next.js!  
            Datos de la API: {data}  
        </div>  
    );  
}  
  
exportar función asíncrona getStaticProps() {  
    const res = await fetch('https://api.example.com/data');  
    const datos = await res.json();  
  
    devolver {  
        accesorios: {  
            datos,  
        },  
    };  
}
```

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

```
```
```

En el ejemplo anterior, la función `getStaticProps` obtiene los datos del punto final de PI en el momento de la compilación. Luego, los datos se pasan como datos de propiedad al componente omePage y Next.js genera una página estática que incluye los datos obtenidos.

### 3. `getStaticPaths`

La función `getStaticPaths` se utiliza para el enrutamiento dinámico y la renderización del lado del servidor con generación estática. Define las rutas posibles para las rutas dinámicas y especifica los datos que se pre-renderizarán. A continuación, se muestra un ejemplo.

```
importar React desde 'react';
función Post({ post }) {
 devolver (
 <div>
 <h1>{título de la publicación}</h1>
 <p>{post.content}</p>
 </div>
);
}

exportar función asíncrona getStaticPaths() {
 const res = await fetch('https://api.example.com/posts');
 const posts = await res.json();

 const rutas = posts.map((post) => ({
 parámetros: { slug: post.slug },
 }));
}

devolver {
```

```
 caminos,
```

```
 reserva: falso,
```

```
};

}

exportar función asíncrona getStaticProps({ parámetros }) {
 const res = await fetch(`https://api.example.com/
publicaciones/${params.slug}`);
 constante post = esperar res.json();

 devolver {
 accesorios: {
 correo,
 },
 };
}

exportar publicación predeterminada;
```

```

En este ejemplo, la función `getStaticPaths` obtiene una lista de publicaciones de un punto de conexión de PI httpsapi.example.composts y asigna cada publicación a su parámetro `slug` correspondiente . El objeto `paths` se devuelve con la lista de rutas.

La función `getStaticProps` obtiene los datos de la publicación según el parámetro dinámico `slug` y los devuelve como propiedades. El componente `Post` recibe la propiedad `post` y renderiza el título y el contenido de la publicación.

Estas funciones (`getServerSideProps`, `getStaticProps` y `getStaticPaths) habilitan la renderización del lado del servidor en Next.js. Le permiten obtener datos de forma dinámica, generar páginas estáticas y pre-renderizar rutas dinámicas, lo que proporciona opciones de renderización eficientes y flexibles para sus aplicaciones Next.js.

C con Next.js

CSR en Next.js le permite renderizar componentes en el lado del cliente, lo que significa que el HTML inicial es mínimo y el contenido se genera y renderiza dinámicamente en el navegador. Aquí hay una explicación detallada con un ejemplo de código de renderizado del lado del cliente con Next.js.

. Energía de componente

En Next.js, puedes crear componentes React en el directorio de páginas y se renderizarán automáticamente en el lado del cliente. Aquí hay un ejemplo de un componente simple renderizado del lado del cliente:

```
importar React desde 'react';
función Página de inicio() {
    devolver (
        <div>
            ¡Bienvenido a mi aplicación Next.js!
            <p>Esta página se renderiza del lado del cliente.</p>
        </div>
    );
}

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

En el ejemplo anterior, tenemos un componente funcional `HomePage` que renderiza contenido estático. Este componente se renderizará en el lado del cliente al acceder a la página.

### . Registro de datos de ADN

La renderización del lado del cliente en Next.js permite obtener datos de PI o fuentes externas de forma dinámica. Se pueden usar diversos enfoques, como `useEffect`, `useState`, o bibliotecas externas como `axios` o `fetch`, para obtener datos del lado del cliente. A continuación, se muestra un ejemplo de obtención de datos del lado del cliente.

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

función Página de inicio() {
 const [datos, setData] = useState(nulo);
```

```
usarEfecto(() => {
 constante fetchData = async () => {
 const res = await fetch('https://api.example.com/data');
 const datos = await res.json();
 setData(datos);
 };

 obtener datos();
}, []);

devolver (
<div>
 ¡Bienvenido a mi aplicación Next.js!
 {data && <p>Datos de la API: {data}</p>}
</div>
);

}

exportar página de inicio predeterminada;
```

```

En el ejemplo actualizado, utilizamos el gancho `useState` para administrar el estado `data` y el gancho `useEffect` para obtener datos de un PI httpsapi.example.comdata. Los datos obtenidos se almacenan en el estado y se representan en el componente.

. nteratiit un Clientesie ates

La renderización del lado del cliente en Next.js permite que los componentes interactivos se actualicen y rendericen dinámicamente según las interacciones o eventos del usuario. Se pueden usar diversos controladores de eventos y bibliotecas de gestión de estados como «Redux», o `React Context`, o incluso el componente `Link` integrado de Next.js para la navegación del lado del cliente. Aquí hay un ejemplo de un componente renderizado interactivo del lado del cliente:

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

función Página de inicio() {
    const [count, setCount] = useState(0);

    const handleIncrement = () => {
        setCount(count + 1);
    };

    devolver (
        <div>
            ¡Bienvenido a mi aplicación Next.js!
            <p>Conde: {count}</p>
            <button onClick={handleIncrement}>Incrementar</button>
        </div>
    );
}

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

En el ejemplo anterior, utilizamos el gancho useState para administrar el estado del conteo. Cuando se hace clic en el botón "Incrementar" , se llama a la función 'handleIncrement' , que actualiza el estado `count` y vuelve a renderizar el componente con el valor actualizado.

La renderización del lado del cliente en Next.js ofrece flexibilidad e interactividad, lo que permite obtener datos dinámicamente y actualizar componentes según las acciones del usuario. Es ideal para escenarios donde el contenido dinámico y la interactividad son fundamentales, como actualizaciones en tiempo real o interfaces de usuario complejas.

## Representación dinámica del lado del cliente

Para lograr una CSR y SSR dinámicas juntas en Next.js, puede usar una combinación de ambos enfoques según su caso de uso específico. Aquí hay una explicación de cómo puede lograr una representación dinámica del lado del cliente con Next.js, utilizando tanto SSR como CSR.

- Representación del lado del servidor SSR

Para la carga inicial de la página o cuando S es crucial, puede aprovechar la representación del lado del servidor. Con SSR, el servidor genera el con los datos iniciales, lo que garantiza que los motores de búsqueda puedan indexar el contenido y los usuarios puedan ver la página completamente representada incluso sin avaScript habilitado.

- Representación del lado del cliente CSR

Para actualizaciones dinámicas o interactividad, puede utilizar la renderización del lado del cliente. La CSR permite obtener y actualizar datos en el lado del cliente sin tener que recargar toda la página, lo que resulta en una experiencia de usuario más ágil. Este enfoque es ideal para escenarios donde se requieren actualizaciones en tiempo real, interacciones del usuario o renderización dinámica de contenido.

Para lograr tanto la RSC como la RSE, puede seguir estos pasos

#### Implementación de SSR por pasos

Usa la función `getServerSideProps` en tus páginas de Next.js para obtener datos del servidor y pasarlo como propiedades a tus componentes. Esto te permite renderizar la página con los datos iniciales antes de enviarla al cliente.

```
```función de exportación asíncrona getServerSideProps() {  
    // Obtener datos de una API o realizar cualquier operación del lado del servidor  
    const datos = await fetchData();  
  
    // Devuelve los datos como propiedades  
    devolver {  
        accesorios: {  
            datos,  
        },  
    };  
}```
```

Paso Implementar la RSE

Para actualizaciones dinámicas o interactividad, puede usar técnicas de renderizado del lado del cliente. Puede obtener datos usando `useState` u otras bibliotecas de obtención de datos del lado del cliente como `'axios'` o `'fetch'`.

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

función MiComponente() {
    const [datos, setData] = useState(nulo);

    usarEfecto(() => {
        constante fetchData = async () => {
            const res = await fetch('https://api.example.com/data');
            const datos = await res.json();
            setData(datos);
        };
        obtener datos();
    }, []);
}

// Representa tu componente con los datos obtenidos
devolver <div>{datos && <p>Datos: {datos}</p>}</div>;
}
```

Al combinar SSR y CSR, se puede lograr un enfoque de renderizado híbrido con Next.js. Esto permite proporcionar contenido inicial renderizado en servidor para S y rendimiento, a la vez que incorpora actualizaciones dinámicas del lado del cliente para la interactividad y los datos en tiempo real. Esta combinación permite una experiencia de usuario flexible y óptima.

La elección entre CSR y SSR depende de varios factores y de los requisitos específicos de su aplicación. Aquí hay una explicación detallada de cuándo utilizar cada enfoque de renderizado:

CSR de renderizado del lado del cliente

Utilice la representación del lado del cliente cuando:

- Contenido dinámico: Si su aplicación depende en gran medida de contenido dinámico que debe obtenerse y actualizarse con frecuencia, CSR es una opción ideal. Le permite obtener datos del lado del cliente y actualizar la interfaz de usuario sin tener que recargar toda la página.
- Interactividad cuando necesita proporcionar una experiencia de usuario altamente interactiva con actualizaciones en tiempo real, como chats en vivo, edición colaborativa o visualización de datos interactiva, CSR es una opción preferida. Permite interacciones fluidas y retroalimentación inmediata de las acciones del usuario.
- Aplicaciones de página única (SP). Si crea una aplicación de página única donde toda la lógica reside en el lado del cliente, CSR suele ser el enfoque predeterminado. Permite una navegación fluida y transiciones entre diferentes vistas sin recargas completas de la página.
- PPS móviles y PPS progresivos Al desarrollar aplicaciones móviles o PPS, CSR puede brindar una experiencia similar a la nativa con transiciones suaves, capacidades sin conexión y uso eficiente de los recursos.

Representación del lado del servidor SSR

Utilice la representación del lado del servidor cuando:

- S y carga de página inicial Si la optimización del motor de búsqueda S es crucial para su aplicación o si desea proporcionar contenido completamente renderizado a los usuarios, incluso con avaScript deshabilitado, se recomienda SSR. Esto garantiza que los motores de búsqueda puedan rastrear e indexar sus páginas eficazmente y que los usuarios puedan ver el contenido de inmediato.
- Rendimiento y tiempo de interacción: SSR puede acelerar el tiempo de interacción, ya que el servidor envía HTML pre-renderizado al cliente. Esto reduce el tiempo de carga inicial de la página y mejora el rendimiento percibido, especialmente en páginas con mucho contenido.

Seguridad y protección de datos: al gestionar datos sensibles, SSR ofrece mayor seguridad al mantener los datos en el servidor. Esto reduce el riesgo de exponer información sensible al código avaScript del cliente.

- Compatibilidad SSR garantiza que su aplicación funcione en una amplia gama de dispositivos, navegadores y configuraciones de usuario, ya que depende menos de la ejecución de avaScript del lado del cliente.

- Intercambio de redes sociales Si compartir en redes sociales es un aspecto importante de su aplicación, SSR garantiza que los enlaces compartidos muestren contenido y metadatos significativos, lo que mejora la experiencia general de intercambio.

Es importante tener en cuenta que también se pueden combinar SSR y CSR en enfoques híbridos, donde algunas páginas se renderizan en el servidor mientras que otras se renderizan en el cliente. Next.js, por ejemplo, ofrece flexibilidad para elegir la estrategia de renderizado adecuada para cada página.

En última instancia, la decisión entre CSR y SSR depende de los requisitos específicos, las compensaciones y los objetivos de su aplicación. Analizar factores como la dinámica del contenido, la interactividad, S, el rendimiento, la seguridad y la compatibilidad lo ayudarán a determinar el enfoque de renderizado más adecuado.

En el dinámico entorno de Next.js, los desarrolladores aprovechan el poder de las directivas para optimizar la ejecución del código tanto en el servidor como en el cliente. Este capítulo profundiza en el uso de dos directivas fundamentales, `use server` y `use client`, y explora su papel en la representación SSR del lado del servidor y la representación CSR del lado del cliente para mejorar el rendimiento y optimizar la organización del código. A continuación, se ofrece una explicación detallada:

- Presentación de las directivas de Next.js

En el contexto de Next.js, directivas como `use server` y `use client` actúan como marcadores para el código, indicando si debe ejecutarse exclusivamente en el servidor o en el cliente. Este uso estratégico de directivas optimiza la funcionalidad general y la capacidad de respuesta de las aplicaciones Next.js.

- Propósito del `servidor de uso`

La directiva `use server` desempeña un papel fundamental en la optimización de la ejecución del código del lado del servidor. Al emplearla, los desarrolladores pueden excluir código designado del paquete del lado del cliente. Esta exclusión contribuye a un tamaño de paquete más reducido, lo que mejora el rendimiento de carga inicial de la página.

ejemplo

```
``javascript
//server-side.js
```

```
'utilizar servidor';

función asíncrona fetchData() {
    respuesta constante = esperar fetch('https://jsonplaceholder.
typicode.com/posts/1');

    const data = await respuesta.json();

    devolver datos;

}

exportar fetchData predeterminado;
'''
```

Los desarrolladores deben tener cuidado al utilizar "use server" para garantizar que el código marcado se ejecute exclusivamente en el servidor y que los resultados se integren perfectamente en la respuesta del cliente.

- Rol de «cliente de uso»

Por el contrario, la directiva `use client` permite identificar código destinado exclusivamente a la ejecución del lado del cliente. Este código se incluye en el paquete del lado del cliente, lo que permite a los desarrolladores interactuar con el objeto Document modelo D y responder a los eventos del usuario.

ejemplo

```
``javascript
// cliente-lado.js

'utilizar cliente';

const handleClick = () => {
    constante           elemento           =           documento.

getElementById('botón');
```

```
elemento.textContent = '¡Hizo clic!';

};

const botón = document.getElementById('botón');

botón.addEventListener('clic', handleClick);
...
```

Con `use client`, las funciones y los escuchas de eventos se ejecutan exclusivamente en el cliente, después de la carga de la página, lo que mejora los elementos interactivos de la aplicación.

- Particionado estratégico de código

Al implementar correctamente las directivas `use server` y `use client` , los desarrolladores pueden dividir eficazmente su código en componentes del lado del servidor y del lado del cliente. Esto no solo mejora la organización del código, sino que también optimiza el rendimiento, contribuyendo a una experiencia de usuario fluida en las aplicaciones Next.js.

- Configuración para el comportamiento predeterminado

Para simplificar el uso de directivas en un proyecto Next.js, los desarrolladores pueden configurar el archivo next.config.json. El siguiente ejemplo demuestra cómo habilitar acciones del lado del servidor de forma predeterminada.

```
``javascript
/** @type {import(next).extonfig} */
constante nextonfig = {

    experimental: {

        serverActions: verdadero,
    },
};
```

```
módulo.exports = nextonfig;  
...
```

Esta configuración garantiza que las acciones del lado del servidor estén habilitadas de forma predeterminada, lo que ofrece una experiencia de desarrollo cohesiva y eficiente con Next.js.

est ratios o sing C una aplicación en Next.js

Al trabajar con CSR y SSR en aplicaciones Next.js, es importante seguir ciertas prácticas recomendadas para garantizar un rendimiento, una capacidad de mantenimiento y una experiencia de usuario óptimos. Estas son algunas prácticas recomendadas clave para usar CSR y SSR en Next.js.

- Separación de código

Mantenga una clara separación entre el código del lado del cliente y del servidor. Utilice las funciones apropiadas de Next.js, como `getServerSideProps`, `getStaticProps` y `useEffect`, para gestionar la lógica de renderizado del lado del servidor y del cliente por separado. Esto facilita la comprensión y la gestión del flujo de datos y el renderizado tanto en el cliente como en el servidor.

- Elija la estrategia de renderizado adecuada

Evalué los requisitos de su aplicación y elija la estrategia de renderizado adecuada. Utilice SSR cuando necesite optimizar el rendimiento de carga inicial de la página o al gestionar datos confidenciales. Utilice CSR cuando necesite contenido dinámico, interactividad o actualizaciones en tiempo real. Considera usar un enfoque híbrido que combine SSR y CSR para diferentes partes de su aplicación.

- Carga lenta y división de código

Implementa técnicas de carga diferida y división de código para optimizar el tiempo de carga y mejorar el rendimiento. Divide tu código en fragmentos más pequeños y cárgalos según demanda, reduciendo el tamaño inicial del paquete. Usa herramientas como importaciones dinámicas, React.lazy y enrutamiento dinámico en Next.js para dividir el código y cargar componentes solo cuando sea necesario.

- Almacenamiento en caché y memorización:

Utilice mecanismos de almacenamiento en caché para mejorar el rendimiento y reducir las solicitudes innecesarias al servidor. Implemente técnicas de almacenamiento en caché del lado del servidor, como el almacenamiento en caché de respuestas de API, consultas a bases de datos o cálculos costosos. Además, utilice mecanismos de almacenamiento en caché del lado del cliente, como la memorización o bibliotecas de gestión de estado del lado del cliente, para evitar rerenderizados innecesarios y optimizar el proceso de renderizado del lado del cliente.

- Estados de carga y error de handling

Considere los estados de carga y error de su aplicación. Muestre indicadores de carga o marcadores de posición al obtener datos en el lado del cliente o del servidor. Gestioné los casos de error con precisión mostrando mensajes de error o contenido de respaldo para brindar una mejor experiencia de usuario y gestionar casos extremos de forma eficaz.

- Monitoreo y optimización del rendimiento

Supervise el rendimiento de su aplicación utilizando herramientas como Lighthouse, PageSpeed Insights o eb itals. Optimice su código, reduzca las repeticiones de renderizado innecesarias y minimice el tamaño de la carga útil para mejorar el rendimiento de CSR y SSR. Utilice técnicas como la optimización de imágenes, la carga diferida de activos y la minimización del número de dependencias externas.

- Pruebas y depuración:

Realice pruebas para cubrir diferentes escenarios y casos extremos para los componentes CSR y SSR. Utilice herramientas como React testing library o EST para realizar pruebas. Además, utilice las herramientas para desarrolladores del navegador, el modo de depuración Next.js y los registros del servidor para depurar problemas relacionados con la representación, la obtención de datos o el rendimiento.

- Documentación y colaboración en equipo:

Documente sus estrategias de renderizado, el flujo de datos y el comportamiento de los componentes para facilitar la colaboración entre los miembros del equipo. Defina claramente las responsabilidades de los componentes CSR y SSR y proporcione directrices sobre cuándo usar cada enfoque. Documente cualquier configuración personalizada o lógica del lado del servidor que pueda afectar el renderizado.

Si sigue estas prácticas recomendadas, podrá optimizar el rendimiento de la renderización, la capacidad de mantenimiento y la experiencia del usuario de su aplicación Next.js cuando utilice la renderización tanto del lado del cliente como del lado del servidor.

Conclusión

La renderización del lado del cliente y del lado del servidor son dos técnicas cruciales en el desarrollo web para ofrecer contenido dinámico e interactivo. Este capítulo exploró los conceptos fundamentales y las diferencias clave entre estos métodos de renderización. La renderización del lado del servidor se destacó como un aspecto crucial de las aplicaciones Next.js debido a su capacidad para mejorar el rendimiento, optimizar la optimización para motores de búsqueda y optimizar la experiencia del usuario. Sin embargo, la renderización del lado del cliente también tiene sus ventajas, especialmente en escenarios que requieren contenido dinámico e interactividad. El capítulo analizó los diferentes escenarios donde la renderización del lado del cliente destaca y brindó información sobre su implementación efectiva en Next.

Aplicaciones Next.js. También se analizaron las mejores prácticas para elegir entre renderizado del lado del cliente y del lado del servidor en aplicaciones Next.js, junto con pautas para optimizar el rendimiento del renderizado. Al comprender estas técnicas de renderizado y su aplicación en Next.js, los desarrolladores pueden tomar decisiones informadas y optimizar el rendimiento del renderizado en sus proyectos.

En el siguiente capítulo, "Seguridad de aplicaciones con Next Auth", profundizaremos en el ámbito de la autenticación y la seguridad en aplicaciones Next.js. Este capítulo te guiará en la implementación de Next Auth, una potente biblioteca de autenticación de terceros. Al aprovechar Next Auth, podrás optimizar el proceso de autenticación e integrar a la perfección diversos proveedores de autenticación, incluyendo algunos populares como Google, Facebook, Twitter y más. Únete a nosotros para explorar el mundo de la autenticación segura en Next.js y aprender a mejorar la seguridad de tus aplicaciones con Next Auth.

Preguntas de opción múltiple

1. ¿Cuáles son las dos técnicas de renderizado populares analizadas en el capítulo?

A. Renderizado front-end y renderizado back-end

B. Representación del lado del cliente y representación del lado del servidor

C. Renderizado estático y renderizado dinámico

D. Renderizado local y renderizado en la nube

2. ¿Qué técnica de renderizado es crítica para las aplicaciones Next.js?

A. Representación del lado del cliente

B. Renderizado del lado del servidor

- C. Representación estática
D. Renderizado dinámico
3. ¿Cuáles son los beneficios de SSR en las aplicaciones Next.js?
Rendimiento mejorado y experiencia de usuario optimizada.
B. Mayor interactividad y contenido dinámico
C. Mejor optimización de motores de búsqueda y tiempos de carga reducidos
D. II de los anteriores
4. ¿En qué escenarios destaca la representación del lado del cliente en las aplicaciones Next.js?
. cuando el rendimiento es una prioridad máxima
B. cuando el contenido dinámico y la interactividad son primordiales
C. La optimización de motores de búsqueda es crucial
D. cuando no se admite la representación del lado del servidor
5. ¿Cuáles son las mejores prácticas para elegir entre el lado del cliente y el lado del servidor?
¿Renderizado del lado del servidor en aplicaciones Next.js?
Elija siempre la representación del lado del cliente para un mejor rendimiento.
B. Elija siempre la representación del lado del servidor para una mejor S
C. Considere factores como los requisitos de desempeño y el contenido.
interactividad
D. Confíe en la configuración de renderizado predeterminada sin personalización

respuestas

1.	b
2.	b
3.	d
4.	b
5.	do

Capítulo 11

Asegurar la aplicación con

Siguiente autorización

En el panorama en constante evolución de las aplicaciones web, garantizar una autenticación y seguridad robustas es fundamental. En este capítulo, profundizaremos en el ámbito de la autenticación y la seguridad en aplicaciones Next.js, centrándonos en la implementación de Next Auth, una potente biblioteca de autenticación de terceros que simplifica el proceso de autenticación entre diversos proveedores.

Para iniciar nuestra exploración, comenzaremos con una introducción a la autenticación y la seguridad. Analizaremos los conceptos fundamentales, destacando la importancia de la autenticación segura de usuarios y la protección de datos confidenciales. Al comprender los principios básicos de la autenticación y la seguridad, obtendrá información valiosa sobre la importancia de incorporar estas prácticas en sus aplicaciones Next.js.

A continuación, analizaremos Next Auth, una herramienta invaluable para simplificar el proceso de autenticación. Ofreceremos una descripción general de Next Auth, explorando sus características, beneficios y cómo se integra a la perfección con las aplicaciones Next.js. Ya sea que trabaje con proveedores de autenticación populares como Google, Facebook, Twitter o proveedores personalizados, NextAuth ofrece un enfoque unificado y optimizado.

La implementación práctica es clave, por lo que le guiaremos en el proceso de configuración de Next Auth en su aplicación Next.js. Aprenderá los pasos y las configuraciones necesarias para comenzar a usar Next Auth, garantizando una integración fluida con su aplicación.

Ampliando la versatilidad de Next Auth, exploraremos la implementación de diferentes proveedores de autenticación. Descubrirá cómo aprovechar las capacidades de Next Auth para autenticar usuarios mediante diversos proveedores, permitiendo que su aplicación se adapte a una amplia gama de preferencias de usuario.

Sin embargo, la autenticación es solo un aspecto; proteger las páginas y las rutas API de su aplicación es igualmente crucial. Le mostraremos cómo proteger sus páginas y rutas API mediante los mecanismos de autenticación proporcionados por Next Auth. Aprenderá a restringir el acceso a áreas sensibles de su aplicación, otorgando solo a los usuarios autenticados el privilegio de interactuar con estos recursos.

Finalmente, profundizaremos en las mejores prácticas de autenticación y seguridad en aplicaciones Next.js. Analizaremos aspectos esenciales, como la gestión de sesiones, la mitigación de vulnerabilidades de seguridad comunes y el mantenimiento de la confianza del usuario. Al seguir estas mejores prácticas, reforzará las medidas de autenticación y seguridad de su aplicación Next.js, garantizando una experiencia de usuario segura y confiable.

A lo largo del capítulo, cubriremos temas que incluyen una introducción a la autenticación y la seguridad, una descripción general de Next Auth, la configuración de Next Auth en una aplicación Next.js, la implementación de diferentes proveedores de autenticación, la protección de páginas y rutas API con autenticación, y las mejores prácticas para la autenticación y la seguridad en aplicaciones Next.js.

Al finalizar este capítulo, tendrá los conocimientos y la experiencia práctica necesarios para integrar Next Auth en sus aplicaciones Next.js, protegiendo eficazmente su aplicación y brindando una experiencia de autenticación fluida a sus usuarios. ¡Profundicemos en el mundo de la autenticación y la seguridad con Next Auth!

Estructura

En este capítulo cubriremos los siguientes temas:

- Introducción a la autenticación y seguridad • Descripción general de Next Auth
- Configuración de Next Auth en una aplicación Next.js • Implementación de diferentes proveedores de autenticación • Protección de páginas y rutas API con autenticación • Mejores prácticas para autenticación y seguridad en aplicaciones Next.js

Introducción a la autenticación y la seguridad

La autenticación, la autorización y la seguridad son componentes fundamentales del desarrollo web que trabajan juntos para garantizar la integridad, la confidencialidad,

y la disponibilidad de datos y recursos. Exploraremos cada concepto en detalle y comprendamos su importancia en el contexto del desarrollo web.

- Autenticación

La autenticación es el proceso de verificar la identidad de un usuario o sistema.

Implica proporcionar credenciales, como un nombre de usuario y una contraseña, para demostrar que uno es quien dice ser. El objetivo principal de la autenticación es garantizar que solo las personas o sistemas autorizados puedan acceder a recursos específicos o realizar ciertas acciones.

En el desarrollo web, la autenticación normalmente implica los siguientes pasos:

Registro de usuario: Los usuarios crean una cuenta proporcionando la información necesaria, como nombre de usuario, correo electrónico y contraseña. Esta información suele almacenarse de forma segura en una base de datos.

Inicio de sesión de usuario: los usuarios se autentican ingresando sus credenciales, generalmente una combinación de nombre de usuario, correo electrónico y contraseña. Las credenciales ingresadas se validan con la información almacenada en la base de datos. Si coinciden, se le otorga acceso al usuario.

Autenticación multifactor (MFA): MFA agrega una capa adicional de seguridad al requerir que los usuarios proporcionen factores de autenticación adicionales, como una contraseña de un solo uso (OTP) enviada a su dispositivo móvil, además de su nombre de usuario y contraseña.

- Autorización

La autorización es el proceso que determina a qué acciones o recursos puede acceder un usuario tras una autenticación exitosa. Una vez verificada la identidad del usuario, los mecanismos de autorización definen sus permisos y restricciones dentro de la aplicación.

En el desarrollo web, la autorización implica:

Control de acceso basado en roles (RBAC). RBAC asigna roles a los usuarios según sus responsabilidades o privilegios. Cada rol tiene un conjunto de permisos asociados. Al asignar roles específicos a los usuarios, se puede controlar su acceso a diferentes funciones o secciones de la aplicación.

Listas de control de acceso (ACL) C proporciona un control más granular sobre recursos o acciones individuales al definir permisos específicos para cada usuario o rol. Con C, puede determinar derechos de acceso granulares, permitiendo o denegando el acceso a recursos específicos.

- Seguridad

La seguridad abarca las medidas adoptadas para proteger las aplicaciones y sistemas web de diversas amenazas, vulnerabilidades y ataques. Implica la implementación de medidas de seguridad para mantener la confidencialidad, integridad y disponibilidad de los datos y recursos.

En el desarrollo web, las consideraciones de seguridad incluyen:

Validación y desinfección de entradas: Validar y desinfectar las entradas del usuario ayuda a prevenir vulnerabilidades de seguridad comunes, como ataques de inyección SS y Cross-Site Scripting. Al garantizar que las entradas del usuario cumplan con los formatos esperados y eliminar datos maliciosos o inesperados, se pueden mitigar estos riesgos.

Comunicación segura: Los protocolos de comunicación segura como HTTPS (HTTP sobre SSL/TLS) cifran los datos transmitidos entre el cliente y el servidor, lo que evita escuchas y manipulaciones.

Configurar correctamente los certificados SSS es crucial para establecer una conexión segura.

Protección de datos: Los datos confidenciales, como contraseñas o información personal, deben almacenarse de forma segura mediante técnicas como el hash y el cifrado. El hash unidireccional cifra las contraseñas, haciéndolas irreversibles, mientras que el cifrado garantiza que solo se pueda acceder a los datos con las claves de descifrado adecuadas.

Encabezados de seguridad: Los encabezados de respuesta P, como Content Security Policy CSP, Strict-transport-Security SS y X-XSS-Protection, ayudan a mitigar los riesgos de seguridad y a proteger contra vulnerabilidades web comunes al definir el comportamiento del navegador y aplicar políticas de seguridad.

Actualizaciones de seguridad periódicas: Es fundamental mantener las aplicaciones web y sus dependencias actualizadas con los últimos parches y correcciones de seguridad. Las actualizaciones suelen corregir vulnerabilidades y fallos de seguridad conocidos, garantizando así la protección de la aplicación.

Pruebas de seguridad: Las evaluaciones de seguridad periódicas, como las pruebas de penetración y el análisis de vulnerabilidades, pueden identificar posibles debilidades en la aplicación. Al identificar y abordar proactivamente los problemas de seguridad, puede mejorar la seguridad general del sistema.

En resumen, la autenticación, la autorización y la seguridad son pilares esenciales del desarrollo web. La correcta implementación de la autenticación garantiza...

La verificación de la identidad de los usuarios, los controles de autorización para el acceso de los usuarios según roles o permisos, y las medidas de seguridad protegen contra diversas amenazas y vulnerabilidades. Al incorporar estos elementos eficazmente, los desarrolladores web pueden crear aplicaciones robustas y seguras.

Descripción general de Next Auth

Next Auth es una biblioteca de autenticación de código abierto para aplicaciones JavaScript, utilizada principalmente en el contexto del desarrollo web. Ofrece una forma sencilla y flexible de implementar diversos métodos de autenticación, como el correo electrónico, contraseña, inicios de sesión sociales (Google, Facebook, Twitter, etc.) y proveedores de identidad externos (OpenID Connect).



El objetivo de Next Auth es simplificar el proceso de autenticación y proporcionar una solución estandarizada para gestionar la autenticación de usuarios en proyectos basados en JavaScript, en particular aquellos desarrollados con frameworks como Next.js. Simplifica las complejidades de implementar la autenticación desde cero, permitiendo a los desarrolladores centrarse en la funcionalidad principal de sus aplicaciones.

Las características principales de Next Auth incluyen las siguientes:

- Proveedores de autenticación

NextAuth.js es compatible con varios proveedores de autenticación, lo que permite a los usuarios autenticarse con sus cuentas existentes en plataformas como GitHub, Google, Facebook, Twitter y más. Ofrece una API unificada para trabajar con diferentes proveedores, lo que facilita el cambio entre ellos o la combinación de varios.

- Gestión de sesiones

NextAuth.js gestiona la gestión de sesiones de usuarios autenticados. Gestiona automáticamente los tokens de sesión, configura cookies seguras solo para HTTP y proporciona enlaces y utilidades para acceder a la información de sesión en sus componentes y páginas de Next.js.

- Rutas API sin servidor

NextAuth.js se integra con las rutas de la API de Next.js para gestionar las solicitudes de autenticación. Proporciona una ruta especial (`/api/auth/[...nextauth].js`) que gestiona los flujos de autenticación, incluido el proveedor de autenticación.

devoluciones de llamadas, actualización de token y más. Esta ruta PI es donde configura las opciones de autenticación y define el comportamiento personalizado.

- Personalización y extensibilidad

NextAuth.js te permite personalizar y ampliar su funcionalidad según tus necesidades específicas. Puedes personalizar los componentes I, añadir flujos de autenticación personalizados, implementar comprobaciones de autorización e incluso ampliar los proveedores de autenticación para que sean compatibles con tus sistemas de autenticación.

- Middleware y devoluciones de llamadas

Nextuth.js ofrece un sistema de middleware flexible y ganchos de devolución de llamada que le permiten injectar lógica personalizada en varias etapas del flujo de autenticación. Esto le permite agregar verificaciones personalizadas, modificar datos del usuario, aplicar medidas de seguridad adicionales e integrarse con servicios externos.

- Ganchos de autenticación

NextAuth.js proporciona ganchos React y funciones del lado del servidor para acceder a datos relacionados con la autenticación en sus componentes Next.js y rutas API.

Estos ganchos incluyen useSession para recuperar información de la sesión, signIn para iniciar flujos de autenticación y getSession para el uso del lado del servidor.

- Seguridad

NextAuth.js sigue las mejores prácticas de seguridad e incluye protecciones integradas contra vectores de ataque comunes, como la falsificación de solicitudes entre sitios (CSRF) y los ataques de secuencias de comandos entre sitios (SS). También admite el almacenamiento seguro de tokens, el cifrado de cookies y otras funciones de seguridad.

Nextuth.js es muy valorado por su simplicidad, flexibilidad y compatibilidad con Next.js. Simplifica gran parte de la complejidad de la implementación de la autenticación, lo que permite a los desarrolladores centrarse en crear sistemas de autenticación seguros e intuitivos para sus aplicaciones Next.js. Cuenta con una comunidad en crecimiento y una extensa documentación, lo que facilita su inicio y la búsqueda de soporte para diversos escenarios de autenticación.

En general, Nextuth simplifica la implementación de la autenticación de usuarios en aplicaciones JavaScript al proporcionar una solución bien documentada y personalizable. Reduce el tiempo y el esfuerzo de desarrollo necesarios para crear sistemas de autenticación seguros y fiables, lo que permite a los desarrolladores centrarse en otros aspectos de sus proyectos.

Exploraremos una descripción general de NextAuth.js y cómo se puede usar para lograr la autenticación y autorización en una aplicación Next.js:

1. Instalación y configuración

Para empezar a usar NextAuth.js, debes instalarlo como dependencia en tu proyecto Next.js. Puedes hacerlo con npm o yarn:

```
npm install next-auth  
...
```

Después de la instalación, debe crear un archivo ...nextauth.js en la raíz de su proyecto Next.js. Este archivo sirve como archivo de configuración de Nextuth.js, donde define proveedores de autenticación, devoluciones de llamadas y otras configuraciones.

2. Proveedores de autenticación

NextAuth.js admite una amplia gama de proveedores de autenticación listos para usar, incluidos:

- Proveedores de OAuth: Google, Facebook, GitHub, Twitter, etc.
- Proveedores de bases de datos ongoDB, yS, PostgreS, etc.
- Proveedores de JWT (JSON Web Token) Autenticación personalizada.

Puede configurar uno o más proveedores de autenticación especificando sus credenciales y opciones en el archivo de configuración. Por ejemplo, para habilitar la autenticación con Google, debe agregar un proveedor de Google con su ID y secreto de cliente.

3. Devoluciones de llamadas y funciones de autenticación

NextAuth.js proporciona funciones de devolución de llamada que le permiten personalizar el flujo de autenticación y manejar los datos del usuario. Estas incluyen:

- signIn: se ejecuta cuando un usuario inicia sesión. Puede utilizar esta devolución de llamada para obtener o crear datos de usuario y devolverlos para que se almacenen en la sesión.
- Cerrar sesión: Se ejecuta cuando un usuario cierra sesión. Aquí puede realizar cualquier limpieza necesaria.
- getSession: Obtiene los datos de la sesión del usuario y los proporciona a páginas o rutas API. Puede personalizar los datos de la sesión e incluir información adicional si es necesario.

Puede definir estas devoluciones de llamadas en el archivo de configuración para adaptar el flujo de autenticación a los requisitos de su aplicación.

4. Autorización y control de acceso

NextAuth.js ofrece compatibilidad integrada con RBC y la definición de reglas de autorización. En la llamada `get Session`, puede enriquecer los datos de la sesión del usuario con información de autorización, como roles o permisos. Esto le permite implementar un control de acceso preciso en su aplicación Next.js, comprobando los datos de la sesión del usuario y determinando si tiene los permisos necesarios para acceder a ciertas páginas o realizar acciones específicas.

5. Gestión y almacenamiento de sesiones

NextAuth.js gestiona las sesiones de usuario almacenando los datos de la sesión en cookies o en una base de datos. De forma predeterminada, utiliza cookies seguras de solo P para almacenar la información de la sesión, lo que garantiza que los datos de la sesión estén cifrados y sean a prueba de manipulaciones. Sin embargo, también puede configurarlo para que utilice una base de datos para el almacenamiento de sesiones, lo que ofrece mayor flexibilidad y escalabilidad.

6. Personalización y temática

NextAuth.js ofrece varias opciones de personalización que te permiten adaptar la experiencia de autenticación al diseño y la imagen de marca de tu aplicación. Puedes personalizar la página de inicio de sesión, añadir campos de formulario adicionales y definir páginas de error personalizadas.

7. Características adicionales

NextAuth.js proporciona varias funciones adicionales para mejorar la autenticación y la autorización, entre ellas:

- Autenticación sin contraseña Admite flujos de inicio de sesión basados en correo electrónico, donde los usuarios reciben enlaces mágicos o contraseñas de un solo uso (OTP) para autenticarse.
- Webhooks: le permite activar eventos o realizar acciones personalizadas en función de eventos de autenticación, como la creación de usuarios o el inicio de sesión.
- Integraciones: NextAuth.js se integra perfectamente con Next.js y marcos populares como React y TypeScript.

En conclusión, NextAuth.js simplifica la implementación de la autenticación y la autorización en aplicaciones Next.js. Con su amplia gama de proveedores de autenticación, devoluciones de llamadas flexibles y funciones integradas de control de acceso, proporciona una base sólida para proteger su Next.

Aplicación js.

Configuración de Next Auth en una aplicación Next.js

Configurar Next Auth en una aplicación Next.js implica varios pasos, como instalar los paquetes necesarios, configurar la biblioteca Nextuth.js e integrarla en la aplicación. Nextuth.js es una biblioteca de autenticación flexible compatible con diversos proveedores de autenticación, como OAuth, JWT, correo electrónico/contraseña, entre otros.

Aquí tienes una guía paso a paso sobre cómo configurar NextAuth en una aplicación Next.js:

Paso 1: Instalar los paquetes necesarios

Comience creando una nueva aplicación Next.js o abriendo una existente.

Abre tu terminal y navega hasta el directorio del proyecto. Instala los siguientes paquetes:

```
npm instalar next-auth
```

```
npm instalar axios
```

```
npm instala react-hook-form
```

te Crea un archivo de configuración de Nextt.js



A continuación, cree un nuevo archivo en el directorio raíz de su proyecto llamado next-auth.config.js. Este archivo contendrá la configuración para Nextuth.js.

Aquí hay un ejemplo de configuración

```
importar NextAuth desde 'next-auth'  
importar proveedores desde 'next-auth/providers'  
opciones constantes = {  
    // Configure uno o más proveedores de autenticación
```

```
proveedores: [  
  Proveedores.GitHub({  
    clientId: 'SU_ID_DE_CLIENTE_DE_GITHUB',  
    clientSecret: 'SU_SECRETO_DE_CLIENTE_DE_GITHUB',  
  }),  
  proveedores.google({  
    clientId: 'SU_ID_DE_CLIENTE_DE_GOOGLE',  
    clientSecret: 'SU_SECRETO_DE_CLIENTE_DE_GOOGLE',  
  }),  
  // Agregue más proveedores de autenticación aquí  
],  
  
// Opciones de configuración opcionales  
sesión: {  
  jwt: cierto,  
},  
devoluciones de llamada: {  
  async signn(usuario, cuenta, perfil) {  
    // lógica de inicio de sesión personalizada  
    devuelve verdadero  
  },  
  redirección asíncrona (url, baseUrl) {  
    // lógica de redirección personalizada  
    devolver baseUrl  
  },  
  sesión asíncrona(sesión, usuario) {  
    // Datos de sesión personalizados  
    sesión.usuario.id = usuario.id  
    sesión de regreso  
  },  
  async wt(token, usuario, cuenta, perfil,  
  esNuevoUsuario) {
```

```

    // lógica de token W personalizada
    token de retorno

},
},
// más opciones de configuración...
}

exportar predeterminado (req, res) => NextAuth(req, res, op-
ciones)

```

En este ejemplo, utilizamos GitHub y Google como proveedores de autenticación. Deberás reemplazar "YOUR_GITHUB_CLIENT_ID", "YOUR_GOOGLE_CLIENT_ID" y "YOUR_GITHUB_CLIENT_SECRET" con tus credenciales de cliente de GitHub y Google. También puedes agregar más proveedores de autenticación, como LinkedIn o Facebook, incluyendo sus objetos de configuración en la matriz `providers`.

Repasemos los aspectos clave de este archivo de configuración.

Proveedores de autenticación

La propiedad "proveedores" es una matriz donde se pueden configurar uno o más proveedores de autenticación. En este ejemplo, hemos configurado GitHub y Google como proveedores, cada uno con su propio objeto de configuración, como Providers.GitHub y Providers.Google. Deberá reemplazar "YOUR_
ID_DE_CLIENTE_DE_GITHUB", 'SU_SECRETO_DE_CLIENTE_DE_GITHUB', 'SU_GOOGLE_
'CLIENTE_ID' y 'SU_SECRETO_DE_CLIENTE_DE_GOOGLE' con sus credenciales de cliente reales para cada proveedor.

Configuración de la sesión

La propiedad de sesión configura el comportamiento de la sesión. En este ejemplo, habilitamos JWT estableciendo jwt: true. Esto permite que NextAuth.js emita y administre JWT para usuarios autenticados. Puedes personalizar otras opciones de sesión, como los adaptadores de base de datos de sesión, la edad máxima de la sesión, etc.

Devoluciones de llamadas

La propiedad de devoluciones de llamada permite definir lógica personalizada en varias etapas del flujo de autenticación. En este ejemplo, hemos definido varias funciones de devolución de llamada como se indica a continuación:

- iniciar sesión

Esta función se llama cuando un usuario inicia sesión correctamente. Puede usarla para realizar acciones adicionales, como guardar los datos del usuario en su base de datos.

- redirigir

Esta función determina la URL a la que se redirigirá al usuario tras la autenticación.

Puede personalizar la lógica de redirección según el rol del usuario u otras condiciones.

- sesión

Esta función permite personalizar el objeto de sesión antes de serializarlo y enviarlo al cliente. Se pueden agregar o modificar propiedades del objeto de sesión.

- jwt:

Esta función permite personalizar la carga útil del token JWT. Se pueden añadir reclamaciones adicionales o modificar el token según la información del usuario.

Configuración internacional

El objeto de opciones proporciona más opciones de configuración que puede explorar en la documentación de NextAuth.js. Estas opciones incluyen especificar el adaptador de base de datos y páginas personalizadas para el inicio de sesión.

cerrar sesión, anular páginas predeterminadas, configurar proveedores de correo electrónico y más.

Finalmente, el archivo de configuración exporta una función que recibe los objetos req y res. NextAuth.js utiliza esta función para gestionar las solicitudes de autenticación.

Paso 3: Crear una ruta API para NextAuth

A continuación, cree un archivo llamado pagesapiauth...nextauth.js. Este archivo gestionará las solicitudes de autenticación. Pegue el siguiente código en el archivo.

```
importar NextAuth desde 'next-auth'  
importar proveedores desde 'next-auth/providers'  
  
opciones constantes = {
```

```
proveedores: [  
    Proveedores.GitHub({  
        clientId: 'SU_ID_DE_CLIENTE_DE_GITHUB',  
        clientSecret: 'SU_SECRETO_DE_CLIENTE_DE_GITHUB',  
    }),  
    // Agregue más proveedores de autenticación aquí  
],  
}  
  
exportar predeterminado (req, res) => NextAuth(req, res,  
opciones)
```

Asegúrese de reemplazar las credenciales del cliente de GitHub con las suyas.

Paso 4: Configurar páginas de autenticación

A continuación, cree un nuevo directorio llamado `pages/auth` y cree dos archivos dentro de él: `signin.js` y `signout.js`. Estos archivos manejarán las páginas relacionadas con la autenticación.

Aquí hay un código de ejemplo para `signin.js`:

```
importar { signn } desde nextauth/client  
exportar función predeterminada Signn() {  
    devolver (  
        <>  
        h1>Signo n/h1>  
        botón al hacer clic={() => signn(github)}>  
        Iniciar sesión con itub  
        </botón>  
        </>  
    )  
}
```

En este ejemplo, usamos la función `signIn` de `next-auth/client` para iniciar el flujo de autenticación con el proveedor de OAuth que deseas. Puedes agregar más proveedores y botones de autenticación según sea necesario.

Paso 5: Utilice NextAuth en su aplicación

Finalmente, puedes empezar a usar NextAuth en tu aplicación Next.js. Abre la página o el componente que deseas e importa el gancho `useSession` del paquete `next-auth/client` .

He aquí un ejemplo:

```
importar { useSession } desde 'next-auth/client'

exportar función predeterminada yage() {
    const [sesión, cargando] = useSession()

    si (cargando) {
        volver cargando.../p>
    }

    si (!sesión) {
        regresar p>arrendamiento iniciar sesión/p>
    }

    regresar <p>Bienvenido, {session.user.name

}</p>
}
```

En este ejemplo, usamos el gancho `useSession` para comprobar si un usuario está autenticado. Si el objeto `session` no está disponible, significa que el usuario no está autenticado y mostramos el mensaje "Inicie sesión".

De lo contrario, mostramos un mensaje de bienvenida con el nombre del usuario.

¡Listo! Has configurado NextAuth correctamente en tu aplicación Next.js. Ahora puedes personalizar el flujo de autenticación, añadir más proveedores y gestionar las sesiones de usuario según sea necesario. Consulta la documentación de NextAuth.js para obtener opciones de configuración más avanzadas y ejemplos de uso.

Implementación de diferentes proveedores de autenticación

La implementación de diferentes proveedores de autenticación con NextAuth.js le permite ofrecer múltiples opciones para que los usuarios se autentiquen e inicien sesión en su aplicación.

NextAuth.js proporciona soporte integrado para varios proveedores de autenticación, incluidos proveedores de OAuth como GitHub, Google, Facebook, Twitter y más, así como autenticación de correo electrónico/contraseña y autenticación JWT.

Aquí hay una explicación detallada sobre cómo implementar diferentes proveedores de autenticación con NextAuth.js:

1. Proveedores de OAuth

Los proveedores de OAuth permiten a los usuarios iniciar sesión en su aplicación con sus cuentas existentes en plataformas populares. Para implementar un proveedor de OAuth:

a. Instalar los paquetes necesarios

Por ejemplo, para implementar la autenticación de GitHub, necesitarías instalar el paquete `next-auth/providers`:

```
npm install next-auth/providers
```

. morir te roier y a él para te Nextt.js configuración

En el archivo de configuración nextuth.js, next-auth.config.js, importe el proveedor del paquete `next-auth/providers` e inclúyalo en la matriz `providers`. Proporcione el ID y el secreto de cliente necesarios para el proveedor:

```
importar proveedores desde 'next-auth/providers'  
opciones constantes = {  
    proveedores: [
```

```

    Proveedores.GitHub({
      clientId: 'SU_ID_DE_CLIENTE_DE_GITHUB',
      clientSecret: 'SU_CLIENTE_GITHUB_
      SECRETO',
    }),
    // Agregue más proveedores de OAuth aquí
  ],
}

// est de la configuración de extuth.s

```

. Con igre te roier on te roiers lathom

Cada proveedor de autenticación requiere cierta configuración en su plataforma. Por ejemplo, para implementar la autenticación de GitHub, debe configurar una aplicación OAuth de GitHub y proporcionar el ID y el secreto de cliente generados en la configuración de Nextuth.js.

2. Autenticación de correo electrónico/contraseña:

NextAuth.js admite la autenticación de correo electrónico y contraseña de forma predeterminada. Los usuarios pueden iniciar sesión con su correo electrónico y contraseña. Para implementar la autenticación por correo electrónico y contraseña:

a. Habilite la autenticación de correo electrónico y contraseña en la configuración de NextAuth.js.

En el archivo de configuración Nextuth.js, habilite el proveedor emailpassword agregándolo a la matriz `providers`:

```

importar proveedores desde 'next-auth/providers'
opciones constantes = {
  proveedores: [
    Proveedores.Correo electrónico({
      servidor: {
        host: 'SU_HOST_SMTP',
        puerto: 'SU_PUERTO_SMTP',
      }
    })
  ]
}

```

```

    aut: {
      usuario: 'SU_NOMBRE_DE_USUARIO_SMTP',
      contraseña: 'SU_CONTRASEÑA_SMTP',
    },
  },
  de: 'SU_DIRECCIÓN_DE_CORREO_ELECTRÓNICO',
),
],
}

// est de la configuración de extuth.s

```

. Con igre te serer etails

Especifique los detalles de su servidor SMTP, como el host, el puerto, las credenciales de autenticación y la dirección de correo electrónico para enviar correos electrónicos de verificación y restablecimiento de contraseña.

3. Autenticación JWT

NextAuth.js también admite la autenticación JWT, lo que permite emitir y validar JWT para los usuarios. Esto permite implementar sistemas de autenticación personalizados o integrarse con otros servicios que utilizan JWT. Para implementar la autenticación JWT:

a. nale Atención en la configuración de Nextt.js

En el archivo de configuración Nextuth.js, habilite la autenticación por provisión de las opciones requeridas:

```

opciones constantes = {
  proveedores: [],
  // est de la configuración de extuth.s
  jwt: {
    secreto: 'TU_SECRETO_JWT',
  },
}

```

. De finir o poseer Firma de un registro de certificación

Necesita definir funciones personalizadas para firmar y verificar tokens. Estas funciones reciben el objeto de usuario y deben devolver un token JWT o validar el token recibido, respectivamente. Puede usar bibliotecas como `jsonwebtoken` para este propósito.

4. Proveedores personalizados

NextAuth.js permite crear proveedores de autenticación personalizados para integrarlos con sistemas de autenticación propios o servicios de terceros que no son compatibles de fábrica. Para implementar un proveedor personalizado:

a. Crear un archivo stom roier

Cree un archivo avaScript para su proveedor personalizado, por ejemplo, myCustomProvider.js. En este archivo, defina las funciones necesarias para la autenticación y la recuperación de datos de usuario. Por ejemplo:

```
importar { Cuenta, Proveedor } desde 'next-auth/
proveedores

const MyCustomProvider = (opciones) => {
    devolver {
        id: 'mi-proveedor-personalizado',
        nombre: 'Mi proveedor personalizado',
        tipo: 'oauth',
        versión: '1.0',
        alcance: leer:perfil,
        URL del token de acceso:
        'https://mycustomprovider.com/oauth2/token',
        URL de autorización:
        'https://mycustomprovider.com/oauth2/auth',
        perfilador: https://mycustomprovider.com/api/
        perfil,
        clientId: opciones.clientId,
        clientSecret: opciones.clientSecret,
```

```

        perfil asíncrono (perfil, tokens) {
            // recuperar datos del usuario del perfil y
            fichas

            devolver {
                id: perfil.id,
                nombre: perfil.nombre,
                correo electrónico: perfil.email,
                imagen: perfil.imagen,
            }
        },
    },

    // Métodos de proveedor adicionales...
}

}

exportar predeterminado MyCustomProvider
b. Importe y agregue el proveedor personalizado a la configuración de Nextuth.js

En el archivo de configuración de Nextuth.js, importe el proveedor personalizado
e inclúyalo en la matriz de proveedores , pasando las opciones requeridas:

importar MyCustomProvider desde './myCustomProvider'
opciones constantes = {
    proveedores: [
        MiProveedorPersonalizado({
            clientId: 'SU_ID_DE_CLIENTE_DE_PROVEEDOR_PERSONALIZADO',
            clientSecret: 'SU_CLIENTE_PROVEEDOR_PERSONALIZADO_
SECRETO',
        }),
        // Otros proveedores...
    ],
}

```

```
// est de la configuración de extuth.s
```

c. Implementar los métodos de proveedor necesarios

En el archivo del proveedor personalizado, deberá implementar los métodos necesarios, como accessToken, refreshToken y oe. Estos métodos gestionan el flujo de autenticación, la recuperación de tokens de acceso, la actualización de tokens y la extracción de datos de usuario de la API del proveedor.

Nota: Los proveedores personalizados requieren una configuración adicional específica del proveedor con el que se integra. Consulte la documentación del proveedor para obtener instrucciones detalladas sobre cómo configurar el proveedor personalizado.

Al implementar diferentes proveedores de autenticación, puede brindarles a sus usuarios la flexibilidad de elegir su método de autenticación preferido, ya sea a través de plataformas populares, correo electrónico/contraseña o sistemas de autenticación personalizados.

NextAuth.js elimina gran parte de la complejidad de trabajar con diferentes proveedores, lo que hace que sea más fácil implementar y administrar la autenticación en su aplicación Next.js.

Protección de páginas y rutas API con autenticación

En Next.js, puedes proteger páginas y rutas de API implementando mecanismos de autenticación. La autenticación garantiza que solo los usuarios autenticados y autorizados puedan acceder a ciertas páginas o puntos finales de API. Si bien existen varios métodos para lograr esto en Next.js, describiremos un método común que utiliza sesiones y cookies.

Estos son los pasos:

1. Configurar la gestión de sesiones del lado del servidor: Next.js proporciona una API integrada para la gestión de sesiones del lado del servidor mediante un paquete llamado `next-iron-session`. Puede instalarlo ejecutando `npm install next-iron-session`. Este paquete permite almacenar datos de sesión de forma segura y eficiente.
2. Cree un punto final de autenticación: en su proyecto Next.js, cree una ruta API específicamente para manejar solicitudes de autenticación, como `api/auth``. Este punto final gestionará el inicio de sesión, el cierre de sesión y la creación de sesiones del usuario.

A continuación se muestra un ejemplo de cómo puede implementar el punto final de autenticación:

```
// api/auth.js
```

```
importar { ironSession } de 'next-iron-session';

manejador de función asíncrona (req, res) {
    si (req.metodo === 'POST') {
        // Validar las credenciales del usuario
        const { nombre de usuario, contraseña } = req.body;
        const isValid = await validateCredentials(nombre de usuario,
            contraseña);

        si (es Válido) {
            // Crear sesión y almacenar información del usuario
            req.session.set('usuario', { nombredeusuario });
            esperar req.session.save();
            res.status(200).json({ éxito: verdadero });
        } demás {
            res.status(401).json({ error: 'Credencia no válida-
iales' });
        }
    } de lo contrario si (req.metodo === 'BORRAR') {
        // Destruir sesión al cerrar sesión
        esperar req.session.destroy();
        res.status(200).json({ éxito: verdadero });
    } demás {
        res.status(405).end(); // Método no permitido
    }
}

exportar predeterminado ironSession(controlador, {
    contraseña: process.env.SESSION_PASSWORD,
    cookieName: 'sesión',
    Opciones de cookies: {
        seguro: proceso.env.NODE_ENV === 'producción',
```

```
    },  
});
```

En este ejemplo, el punto final /api/auth gestiona las solicitudes POST para iniciar sesión y las solicitudes DELETE para cerrar sesión. Valida las credenciales del usuario y crea o elimina la sesión según corresponda mediante next-iron-session.

3. Implementar la lógica de autenticación de usuarios: En el punto final de autenticación, deberá implementar la lógica para validar las credenciales de usuario, generalmente mediante una base de datos de usuarios o un servicio de autenticación externo (por ejemplo, OAuth). Si las credenciales son válidas, se crea una sesión y se almacena en ella la información relevante del usuario.
4. Cree un middleware para comprobar la autenticación: Implemente una función de middleware que compruebe si el usuario está autenticado antes de acceder a páginas protegidas o rutas PI. Puede crear un archivo llamado authMiddleware.js` con el siguiente código:

```
// authMiddleware.js  
  
importar { ironSession } de 'next-iron-session';  
  
const withAuth = (controlador) => {  
  devolver ironSession(async (req, res) => {  
    const usuario = re.session.get(usuario);  
  
    si (!usuario) {  
      res.status(401).json({ error: 'No autorizado' });  
    } demás {  
      req.usuario = usuario;  
      manejador de retorno (req, res);  
    }  
  }, {  
    contraseña: process.env.SESSION_PASSWORD,  
    cookieName: 'sesión',  
    Opciones de cookies: {  
      seguro: proceso.env.NODE_ENV === 'producción',  
    },  
  });
```

```
};

exportar predeterminado conAuth;
```

Esta función de middleware comprueba si el usuario está autenticado verificando si la sesión contiene la información del usuario. Si el usuario está autenticado, permite el acceso a la página protegida o a la ruta API; de lo contrario, devuelve un error 401 "No autorizado".

5. Páginas seguras y rutas API para proteger una página específica o una ruta PI, envuelva el componente o controlador correspondiente con `withAuth` software intermedio.

A continuación se muestran ejemplos de una página protegida y una ruta API:

```
// páginas/tablero.s

importar conAuth desde './ruta/a/authMiddleware';

const Dashboardage = ({ usuario }) => {

    // Contenido de página protegido

};

exportar predeterminado withuth(Dashboardage);
```

```
// páginas/api/protected.s

importar conAuth desde '../path/to/authMiddleware';

constante protectedHandler = (req, res) => {

    // protegido           lógica de ruta

};

exportar predeterminado conAuth(protectedHandler);
```

Al encapsular el componente `DashboardPage` y el componente `'protectedHandler'` con `'withAuth'`, estos quedan protegidos y solo los usuarios autenticados pueden acceder a ellos. Si un usuario no está autenticado, recibirá un error 401 "No autorizado".

6. Implemente la función de inicio y cierre de sesión: Utilice la función de inicio y cierre de sesión mencionada anteriormente para autenticar a los usuarios y administrar sus sesiones. Puede redirigirlos a la página de inicio de sesión si no están autenticados.

Siguiendo estos pasos, puede proteger páginas y rutas PI en Next.js implementando middleware de autenticación mediante sesiones del lado del servidor. El middleware verifica la autenticación antes de otorgar acceso a las páginas protegidas.

recursos.

Mejores prácticas para la autenticación y seguridad en aplicaciones Next.js

Implementar prácticas sólidas de autenticación y seguridad es crucial para garantizar la integridad y confidencialidad de los datos en las aplicaciones Next.js. Estas son algunas prácticas recomendadas a seguir cuando se trata de autenticación y seguridad en aplicaciones Next.js:

1. Utilice mecanismos de autenticación fuertes

- Implementar protocolos de autenticación seguros como OAuth 2.0 o JWT para la autenticación de usuarios.
- Aplicar políticas de contraseñas seguras, incluidos requisitos de complejidad, hash de contraseñas y salado.
- Considere implementar F para agregar una capa adicional de seguridad.

2. Implementar la autorización y el control de acceso

- Utilice RBC o C para controlar los permisos de usuario y restringir el acceso a recursos o acciones sensibles.
- Validar los roles o permisos de los usuarios en el lado del servidor para evitar el acceso no autorizado.

3. Almacene de forma segura las credenciales de usuario

- Contraseñas en hash y sal utilizando algoritmos criptográficos fuertes como bcrypt o Argon2.
- Evite almacenar información confidencial como contraseñas o claves API en texto sin formato.
- Considere utilizar sistemas seguros de gestión de credenciales como Administrador de secretos o variables de entorno.

S

4. Protéjase contra ataques de secuencias de comandos entre sitios (XSS)

- Implementar la validación y desinfección de entradas para evitar el uso malicioso.
- Entrada del usuario que se ejecuta como código.
- Utilice bibliotecas de seguridad como `DOMPurify` para desinfectar los datos generados por el usuario contenido antes de renderizarlo en el lado del cliente.

5. Prevenir ataques CSRF

- Utilice tokens CSRF y validelos en el lado del servidor para evitar solicitudes maliciosas de sitios externos.
- Incluir mecanismos de protección anti-CSRF como cookies SameSite y encabezados CSRF.

6. Implementar la seguridad de la capa de transporte (TLS)

- Utilice HTTPS para garantizar la comunicación cifrada entre el cliente y el servidor, protegiendo los datos confidenciales contra interceptación o manipulación.
- Obtener y configurar certificados SSS para establecer una conexión segura

conexiones.

7. Sanitizar y validar la entrada del usuario

- Implementar la validación de entrada del lado del servidor para garantizar que la entrada del usuario Se adhiere a los formatos y límites esperados.
- Utilice bibliotecas como oi o validator.js para validar la entrada del usuario para tipos de datos o patrones específicos.

8. Actualice periódicamente las dependencias y los parches

- Manténgase actualizado con las últimas versiones de Next.js y sus dependencias para beneficiarse de las correcciones y parches de seguridad.
- Monitorear los avisos de seguridad y aplicar parches rápidamente para solucionar las vulnerabilidades conocidas.

9. Implementar límites de velocidad y encabezados de seguridad

- Aplicar la limitación de velocidad para evitar ataques de fuerza bruta y tráfico excesivo. consumo de recursos.
- Configure encabezados de seguridad como CSP, SS y -SS-Protection para mitigar vulnerabilidades web comunes.

10. Realizar auditorías y pruebas de seguridad periódicas

- Realizar auditorías de seguridad y pruebas de penetración para identificar vulnerabilidades en su aplicación.
- Realizar revisiones periódicas del código y evaluaciones de seguridad para garantizar que se sigan las mejores prácticas.

11. Educar a los usuarios y mantener registros de auditoría

- Educar a los usuarios sobre las mejores prácticas de seguridad, como usar contraseñas seguras y ser cautelosos ante los intentos de phishing.
- Mantener registros de eventos relacionados con la autenticación y la seguridad para auditoría y análisis.

Si sigue estas prácticas recomendadas, podrá mejorar la autenticación y la seguridad de sus aplicaciones Next.js y protegerse contra vulnerabilidades y amenazas comunes.

Conclusión

Este capítulo ha proporcionado una exploración exhaustiva de la autenticación y la seguridad en aplicaciones Next.js, con especial atención a la autenticación de Next. Empezamos enfatizando la importancia de una autenticación robusta y medidas de seguridad en aplicaciones web. Comprender los conceptos fundamentales de autenticación y seguridad es crucial para implementar estas prácticas eficazmente.

Next Auth se presentó como una potente biblioteca de autenticación de terceros que simplifica el proceso de autenticación entre diversos proveedores. Se exploraron sus características, beneficios y su integración fluida con las aplicaciones Next.js. Se hizo hincapié en la implementación práctica y se proporcionó una guía paso a paso para configurar Next Auth en una aplicación Next.js.

El capítulo también analizó la implementación de diferentes proveedores de autenticación y demostró cómo proteger páginas y rutas API mediante los mecanismos de autenticación de NextAuth. También se destacaron las mejores prácticas de autenticación y seguridad en aplicaciones Next.js, abarcando temas como la gestión de sesiones, la mitigación de vulnerabilidades de seguridad y el mantenimiento de la confianza del usuario.

En resumen, este capítulo proporcionó a los lectores los conocimientos y las habilidades prácticas necesarias para incorporar Next Auth en sus aplicaciones Next.js, garantizando una autenticación segura y fluida para los usuarios. Siguiendo las mejores prácticas descritas, los desarrolladores pueden fortalecer sus aplicaciones y proporcionar un entorno seguro para que los usuarios interactúen con sus recursos.

En el próximo capítulo, profundizaremos en la instalación y configuración de un Next.js para desarrollar una aplicación CRD. Exploraremos cómo manejar el enrutamiento, crear páginas e implementar la estructura básica de la aplicación. Además, aprenderemos sobre los métodos de obtención de datos de Next.js y cómo integrar una base de datos para realizar operaciones CRD de manera eficiente.

Preguntas de opción múltiple

1. ¿Cuál es el enfoque de este capítulo?
 - A. Gestión de bases de datos en aplicaciones Next.js
 - B. Optimización del rendimiento en aplicaciones Next.js

C. Autenticación y seguridad en aplicaciones Next.js D.

Principios de diseño de UI en aplicaciones Next.js

2. ¿Qué biblioteca de autenticación se analiza en este capítulo?

A. Passport.js

B. Autenticación de Firebase

C. Siguiente autorización

D. Autenticación de Okta

3. ¿Cuál es el beneficio de utilizar Nextuth?

A. Simplifica el proceso de autenticación entre varios proveedores.

B. Proporciona componentes de interfaz de usuario avanzados para aplicaciones Next.js

C. Ofrece capacidades de gestión de bases de datos integradas.

D. Permite la sincronización de datos en tiempo real.

4. ¿Cuál es uno de los temas tratados en este capítulo?

A. Algoritmos de aprendizaje automático en aplicaciones Next.js

B. Localización e internacionalización en aplicaciones Next.js C. Diseño

web responsive en aplicaciones Next.js

D. Protección de páginas y rutas API con autenticación

5. ¿Cuál es el propósito de discutir las mejores prácticas en este capítulo?

A. Para destacar las desventajas de usar Next Auth

B. Para mostrar bibliotecas de autenticación alternativas

C. Brindar orientación para garantizar la autenticación segura y la confianza del

usuario. D. Presentar las próximas funciones de Next.js.

Respuestas

1 C	
2 C	
3 A	
4D	
5 °C	

Capítulo 12

Desarrollo de una aplicación CRUD con Next.js

Introducción

En este capítulo, nos adentraremos en el apasionante mundo de la creación de aplicaciones CRUD (Crear, Leer, Actualizar, Eliminar) con Next.js, un popular y potente framework de React para renderizado del lado del servidor y generación de sitios web estáticos. Aprovecharemos las capacidades de Next.js para desarrollar una aplicación de tareas completamente funcional, que permita a los usuarios crear, gestionar, actualizar y eliminar sus tareas fácilmente.

Para impulsar el backend de nuestra aplicación, integraremos Supabase, una alternativa moderna y de código abierto a las bases de datos tradicionales. Supabase combina la simplicidad y escalabilidad de PostgreSQL con una API de datos reactiva y en tiempo real.

Al utilizar Supabase, podemos gestionar fácilmente el almacenamiento y la recuperación de datos, así como gestionar actualizaciones en tiempo real sin problemas.

A lo largo de este capítulo, exploraremos los conceptos y técnicas clave necesarios para crear una aplicación CRUD robusta. Comenzaremos configurando nuestro entorno de desarrollo, asegurándonos de tener todas las herramientas y dependencias necesarias instaladas. A continuación, crearemos un nuevo proyecto Next.js y lo configuraremos para que funcione a la perfección con Supabase.

Adoptaremos un enfoque práctico y te guiarémos en el proceso de diseño de la interfaz de usuario de la aplicación utilizando componentes de React y las funciones de estilo integradas de Next.js. Aprenderás a crear formularios para la entrada de datos del usuario, mostrar datos dinámicos e implementar la navegación entre las diferentes páginas de la aplicación.

A continuación, nos sumergiremos en la funcionalidad principal de nuestra aplicación To-do. Implementaremos la lógica necesaria para crear nuevas tareas, recuperar tareas existentes de la base de datos, actualizar su estado y eliminarlas. También exploraremos cómo gestionar la validación y la gestión de errores para garantizar una experiencia de usuario fluida.

Al final de este capítulo, tendrá una comprensión profunda de cómo aprovechar el poder de Next.js y Supabase para desarrollar una aplicación CRUD rica en funciones.

Tendrás las habilidades para crear aplicaciones similares con diferentes modelos de datos y ampliar la funcionalidad para adaptarla a tus necesidades específicas. ¡Comencemos este emocionante proceso de creación de aplicaciones CRUD con Next.js y Supabase!

Estructura

En este capítulo se tratarán los siguientes temas:

- Introducción a las aplicaciones CRUD y cómo funcionan
- Configuración de un entorno de desarrollo Next.js y creación de un nuevo proyecto
- Creación de un esquema de base de datos para la aplicación CRUD y configuración de una conexión de base de datos
- Creación de la interfaz de usuario para la aplicación CRUD utilizando React y Next.js, incluida la creación de componentes y su estilo con CSS
- Implementar la aplicación en un servidor de producción

Configuración de su entorno de desarrollo

Prerrequisitos:

- Nodo 16.8 o posterior
- Sistema operativo macOS, Windows (incluido WSL) o Linux
- npm (un gestor de paquetes)
- Código de Visual Studio
- Cuenta de GitHub

Ahora, descargaremos la última versión de Next.js. Abra su terminal e introduzca el siguiente comando:

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

Recibirás las siguientes indicaciones:

¿Cómo se llama tu proyecto? todo-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

Asigne a su proyecto el nombre "todo-app", como en el ejemplo anterior, o elija el nombre que prefiera. Puede seleccionar "No" para las preguntas restantes, pero responda "Sí" a la pregunta sobre el uso de App Router, una nueva función introducida en Next.js 13.

Ahora, abra Visual Studio Code y navegue hasta la carpeta recién creada. Debería ver la siguiente estructura de archivos.

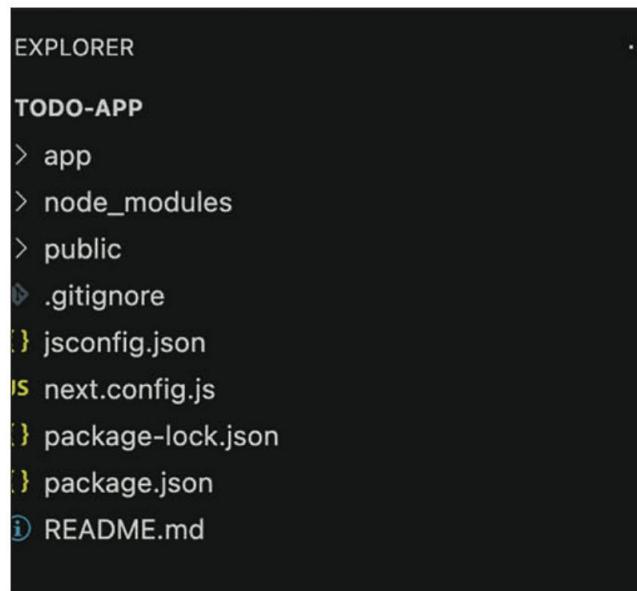


Figura 12.1: Estructura de carpetas de la aplicación

Ahora, ingrese el siguiente comando en la terminal:

```
npm ejecutar dev
```

Visita <http://localhost:3000> en el navegador. Deberías ver lo siguiente.
pantalla:

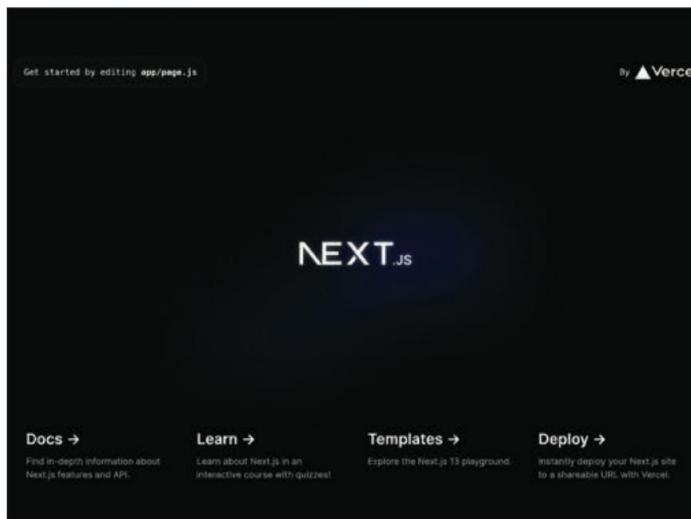


Figura 12.2: Pantalla predeterminada de la aplicación Next.js

Como iniciamos nuestra aplicación Next usando el comando `create-next-app` , borremos parte del código repetitivo de algunos archivos para obtener algo como esto

aplicación/globales.css

```
* {  
    tamaño de caja: caja de borde;  
    relleno: 0;  
    margen: 0;  
}
```

```
html,  
cuerpo {  
    ancho máximo: 100vw;  
    overowx: oculto;  
}
```

```
cuerpo {  
    fondo: negro;  
    color: blanco;  
}
```

aplicación/layout.js

```
importar './globals.css'  
importar { Inter } desde 'next/font/google'  
  
const inter = Inter({ subconjuntos: ['latin'] })
```

```
exportar const metadatos = {  
    Título: 'Aplicación de tareas pendientes'  
    descripción: 'Creado usando Next.js 13',  
}
```

```
exportar función predeterminada RootLayout({ children }) {  
    devolver (  
        <html lang="es">  
            <body className={inter.className}>{hijos}</body>  
        </html>  
    )  
}
```

aplicación/página.módulo.css

```
.principal {  
    pantalla: ej;  
    exdirección: columna;  
    espacio: 40px;  
    alinear-elementos: centro;  
    relleno: 6rem;  
    altura mínima: 100vh;  
}  
  
.
```

```
.botón de ir {  
    fondo: verde;  
    relleno: 10px;  
}
```

aplicación/página.js

```
importar estilos desde “./page.module.css”;  
importar enlace desde “siguiente/enlace”;
```

```
exportar función predeterminada Inicio()  
{ return (  
    <nombreClase principal={styles.main}>  
        Aplicación de tareas pendientes  
        <p>Esta es una aplicación creada con Next.js 13</p>  
        ¡Vamos!  
    </principal>  
,  
);  
}
```

¡Felicitaciones! Next.js se ha configurado correctamente en su computadora para el desarrollo local. Debería verse como se muestra en la Figura 12.3:

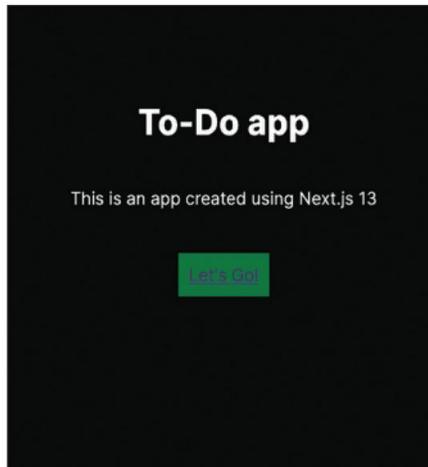


Figura 12.3: Aplicación To-Do

Visualización de elementos de tareas pendientes (Leer)

Hemos creado la página de inicio de nuestra aplicación, que los usuarios verán al visitar la página principal. Al hacer clic en el botón "Comenzar" , los redirigiremos a la página principal de la aplicación, donde podrán realizar operaciones CRUD.

Ahora crearemos una nueva carpeta crud dentro de la carpeta de la aplicación y crearemos un page.js para renderizar la interfaz de usuario para la ruta crud y un archivo page.module.css para escribir su CSS.

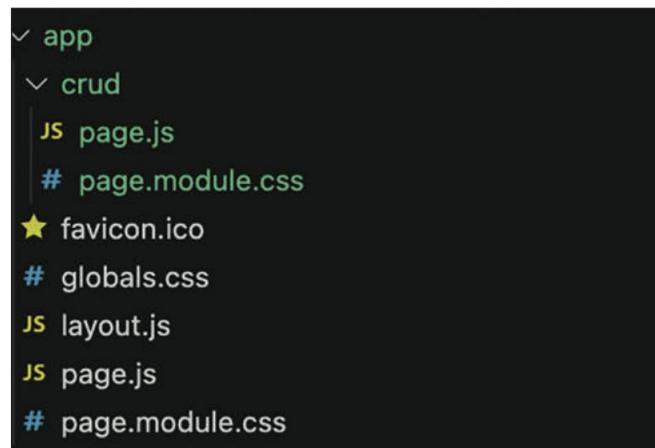


Figura 12.4 Nuevas carpetas y archivos agregados

El contenido de page.js en la carpeta crud será el siguiente:

```
importar estilos desde "./page.module.css";
```

```
exportar función predeterminada Crud() {  
    devolver (  
        <div className={styles.main}>  
            ¡Aquí realizaremos operaciones CRUD!  
        </div>  
    );  
}
```

página.módulo.css

```
.principal {  
    pantalla: ej;  
    exdirección: columna;  
    espacio: 40px;  
    alinear-elementos: centro;  
    relleno: 6rem;  
    altura mínima: 100vh;  
}
```

```
.botón de ir {  
    fondo: verde;  
    relleno: 10px;  
}
```

Si navegamos a <http://localhost:3000/crud>, ahora veremos lo siguiente pantalla:

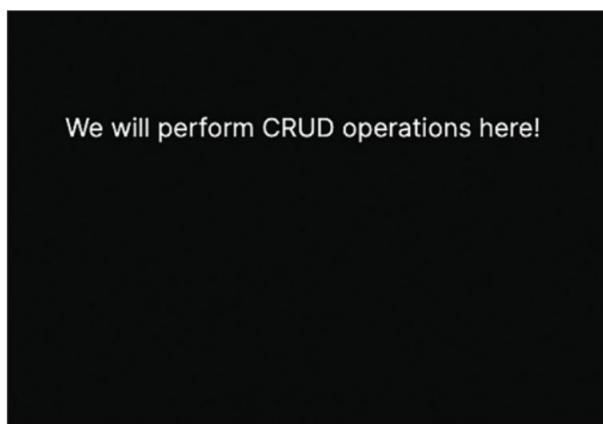


Figura 12.5: Salida en pantalla

Cambiaremos el href del componente de tinta en el archivo apppage.js a
¡Vamos!

Entonces, cuando un usuario hace clic en este enlace, será dirigido a la ruta crud anterior que acabamos de crear.

Ahora, editaremos el page.js en la ruta crud para construir nuestra aplicación. Como primer paso, crearemos una matriz estática de elementos (que contiene elementos de tareas pendientes) y los representaremos en la pantalla:

```
const items = ["Ir al gimnasio", "Comprar alimentos", "Recoger el correo"];
```

La matriz de elementos anterior contiene una lista de elementos por hacer, que intentaremos representar en la pantalla.

Crearemos una nueva carpeta de componentes en el nivel raíz y crearemos un ToDoltem Componente dentro de él. Recibirá una propiedad de elemento que contiene el nombre de la tarea pendiente:

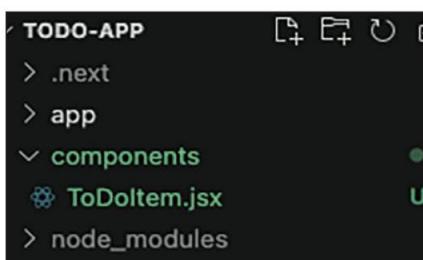


Figura 12.6 Archivo ToDoItem.jsx agregado

```
const ToDoItem = ({ elemento }) => {
  devolver <p>{item}</p>;
};
```

exportar predeterminado ToDoItem;

Cambie crud/page.js a:

```
importar estilos desde "./page.module.css";
importar ToDoItem desde "@/components/ToDoItem";

exportar función predeterminada Crud() {
  const items = ["Ir al gimnasio", "Comprar alimentos", "Recoger el correo"];

  devolver (
    <div className={styles.main}>
      <h2>Elementos por hacer</h2>
      <ul>
        {items.map((item) => (
          <ToDoItem item={item} />
        ))}
      </ul>
    </div>
  );
}
```

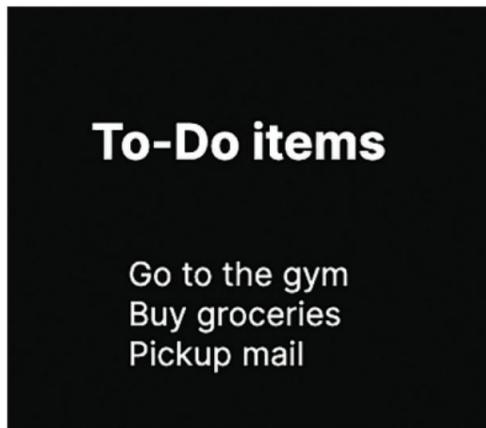


Figura 12.7: Elementos de tareas pendientes representados

Ahora puedes ver que estamos renderizando los elementos ToDo desde la matriz.

A continuación, puliremos la interfaz de usuario de la lista de tareas pendientes. Para ello, instalaremos una biblioteca de interfaz de usuario llamada Material UI. Esta biblioteca incluye numerosos componentes y utilidades de React preconfigurados. Ejecuta el comando `npm install @mui/material @emotion/react @emotion/styled @mui/icons-material` para instalar Material UI y su paquete de iconos. Añadiremos una casilla de verificación para marcar los elementos como completados y un botón de eliminación para cada elemento:

componentes/ToDoItem.jsx

“utilizar cliente”;

```
importar React desde “react”;
importar estilos desde “./Todoltem.module.css”;
importar ListItem desde “@mui/material/ListItem”;
importar ListItemButton desde “@mui/material/ListItemButton”;
importar ListItemIcon desde “@mui/material/ListItemIcon”;
importar Casilla de verificación desde “@mui/material/Checkbox”;
importar ListItemText desde “@mui/material/ListItemText”;
importar DeleteIcon desde “@mui/icons-material/Delete”;
```

```
const ToDoltem = ({ elemento, clave }) => {  
  const [marcado, establecidoMarcado] = React.useState([0]);
```

```
  const handleToggle = (valor) => () => {  
    const currentIndex = check.indexOf(valor);  
    const newChecked = [...comprobado];
```

```
    si (índiceActual === -1) {  
      newChecked.push(valor);  
    } demás {  
      newChecked.splice(índiceActual, 1);  
    }
```

```
    establecerComprobado(nuevoComprobado);  
  };  
  devolver (  
    <ListItemIcon className={styles.item} clave={clave}>  
      <ListItemIconButton onClick={handleToggle(tecla)}>  
        <íconoDeElementoDeLista>  
        <Casilla de verificación  
          comprobado={comprobado.indexOf(clave) !== -1}  
          índice de tabulación={-1}  
          nombreDeClase={styles.checkbox}  
        />  
        </íconoDeElementoDeLista>  
      <ListItemIconText primary={item} />  
    </BotónDeElementoDeLista>  
    <íconoDeElementoDeLista>
```

```
<DeleteIcon className={styles.delete} />
</IconoDeElementoDeLista>
</ListIItem>
);
};
```

exportar predeterminado ToDoltem;

componentes/ToDoItem.module.css

```
// Estilos
CSS .item {
fondo: negro;
margen inferior: 20px;
}
```

```
.casilla de verificación {
color: blanco;
}
```

```
.borrar {
color: rgb(235, 81, 81);
}
```

crud/page.js

```
// Componente que renderiza los elementos de Todo
```

“utilizar cliente”;

```
importar estilos desde "./page.module.css";
importar Todoltem desde "@/components/Todoltem";
importar Lista desde "@mui/material/List";
```

```
exportar función predeterminada Crud() {
  const items = ["Ir al gimnasio", "Comprar alimentos", "Recoger el correo"];
```

```
devolver (
  <div className={styles.main}>
    <h2>Elementos por hacer</h2>
    <Lista>
      {items.map((item, index) =>
        ( <Todoltem item={item} key={index} />
      )));
    </Lista>
  </div>
);
```

Notarás que escribimos una directiva de uso de cliente en la parte superior de los archivos. Esto se usa para especificar que es un componente de cliente, ya que estamos usando la biblioteca de cliente externa, Material UI.

Los cambios anteriores generarán la siguiente interfaz de usuario:

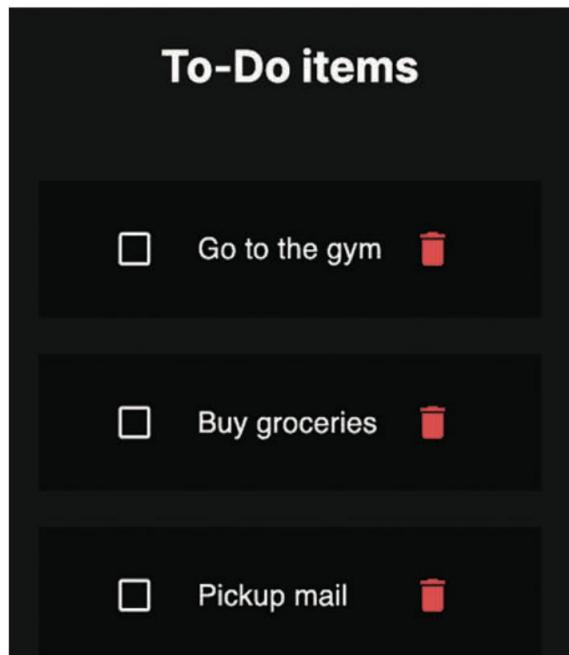


Figura 12.8: Elementos de tareas pendientes con casilla de verificación y botón de eliminar

No hemos agregado ninguna funcionalidad al botón Eliminar ni a la casilla de verificación hasta el momento, ya que los cubriremos en las próximas secciones.

Configuración de la base de datos utilizando Supabase

Nuestra aplicación ahora renderiza la interfaz de usuario a partir de los elementos de la matriz estática que contienen contenido ["Ir al gimnasio", "Comprar comida", "Recoger correo"]. En una aplicación real, estos datos se almacenarían en una base de datos. Realizaríamos una llamada a la API para obtener estos datos de un servidor y luego renderizarlos en el frontend. ¡Eso es exactamente lo que haremos ahora!

Para ello, ve a [Supabase.com](https://supabase.com) e inicia sesión con tus credenciales de GitHub. Verás el siguiente panel:

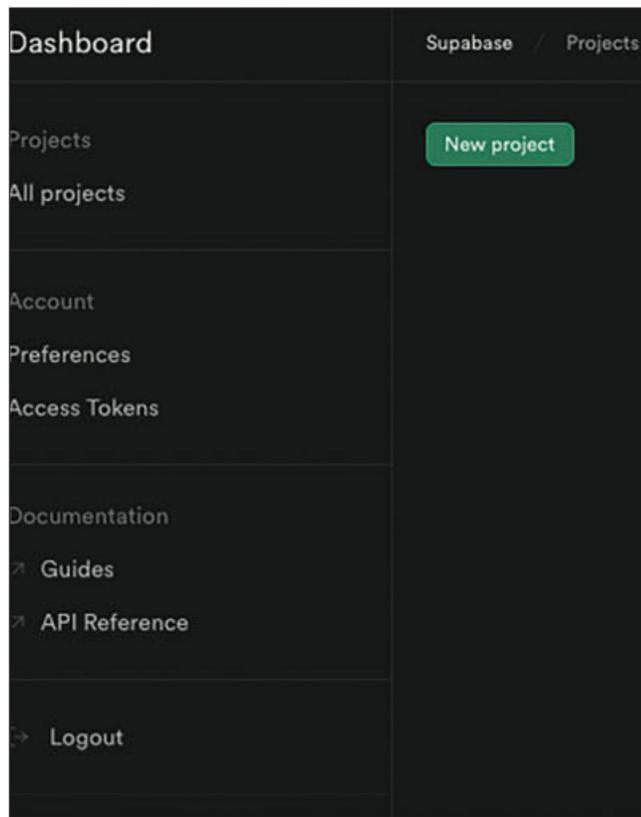


Figura 12.9: Panel de control de Supabase

Haga clic en el botón Nuevo proyecto y complete el Nombre y el Tipo de organización, como se muestra en la Figura 12.10:

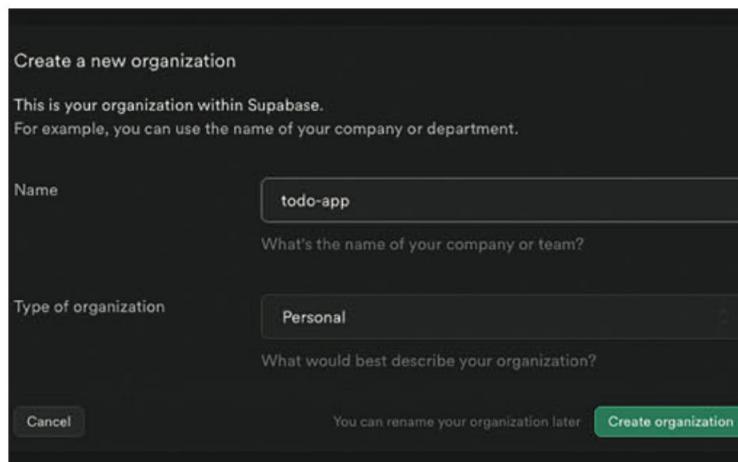


Figura 12.10: Panel de control de Supabase

Luego, completa los siguientes datos para crear un nuevo proyecto.

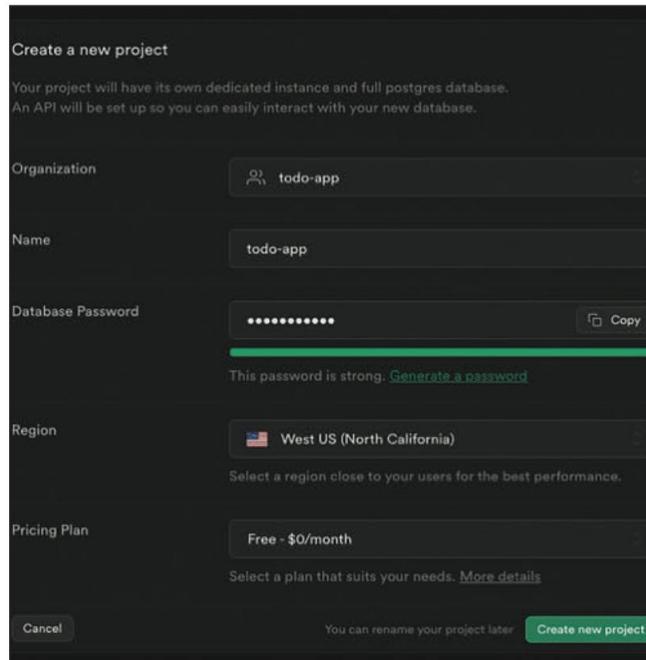


Figura 12.11: Pantalla Crear un nuevo proyecto

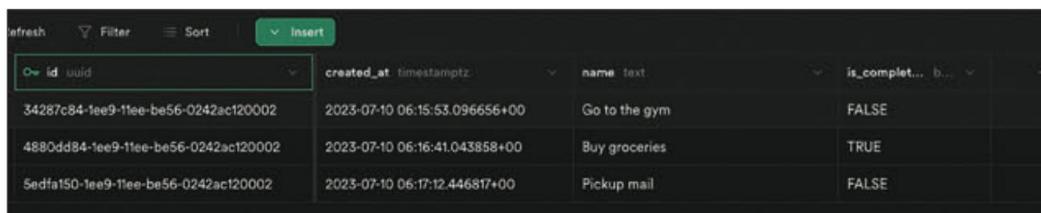
Vaya a la pestaña Editor de tablas , nombre la tabla como elementos, cree las siguientes cuatro columnas y haga clic en Guardar:

Name ⓘ	Type	Default Value ⓘ	Primary
id	uuid	NULL	<input checked="" type="checkbox"/>
created_at	timestamp	now()	<input type="checkbox"/>
name	text	NULL	<input type="checkbox"/>
is_completed	bool	false	<input type="checkbox"/>

Figura 12.12: Esquema de la base de datos

Ahora, usa el botón "Insertar" para agregar los mismos elementos de la tarea que usaste en Next.js. Puedes usar <https://www.uuidgenerator.net/version1> por ahora para crear UUID y agregarlos a una fila de la tabla.

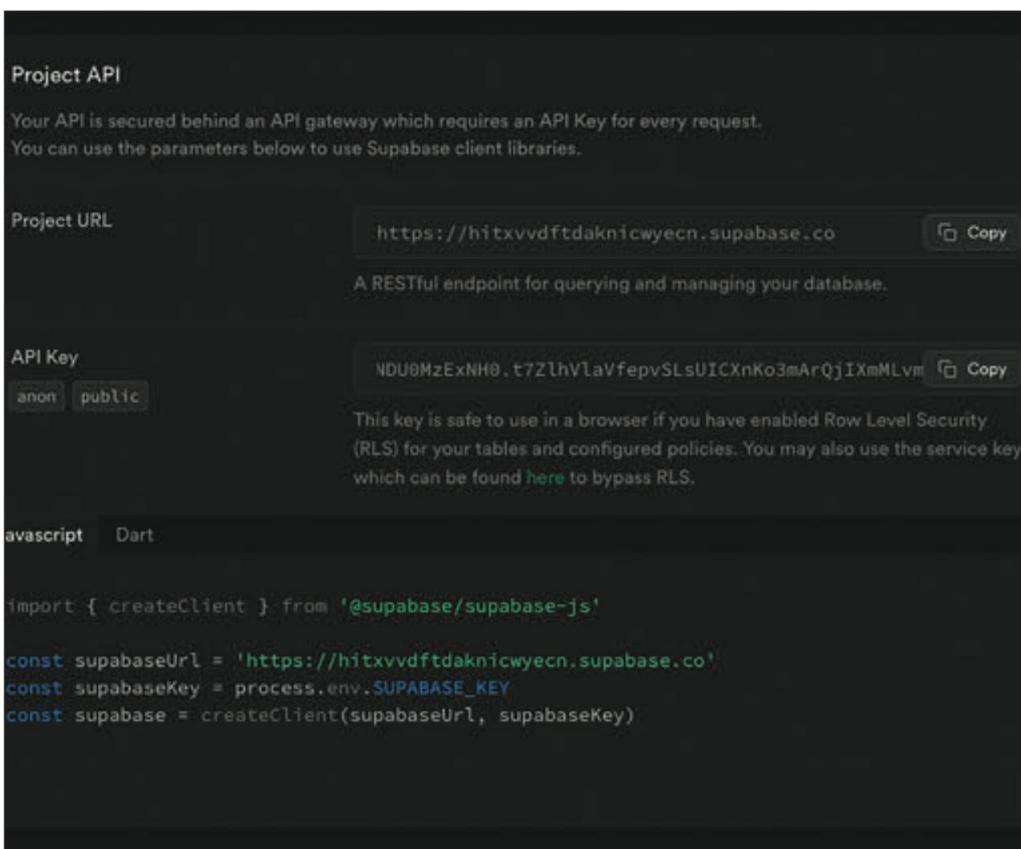
Después de agregar estos elementos, nuestra tabla se verá así:



id	created_at	name	is_complet...
34287c84-1ee9-11ee-be56-0242ac120002	2023-07-10 06:15:53.096656+00	Go to the gym	FALSE
4880dd84-1ee9-11ee-be56-0242ac120002	2023-07-10 06:16:41.043858+00	Buy groceries	TRUE
Sedfa150-1ee9-11ee-be56-0242ac120002	2023-07-10 06:17:12.446817+00	Pickup mail	FALSE

Figura 12.13: Entradas de la base de datos

¡Estamos progresando! El siguiente paso es obtener estos datos mediante una llamada a la API desde nuestra aplicación Next.js y renderizarlos en nuestra interfaz de usuario. Afortunadamente, Supabase ofrece una forma sencilla de hacerlo, como se muestra en la Figura 12.14:



Project API

Your API is secured behind an API gateway which requires an API Key for every request.
You can use the parameters below to use Supabase client libraries.

Project URL
<https://hitxvvdftdaknicwecn.supabase.co>

A RESTful endpoint for querying and managing your database.

API Key
 anon public

This key is safe to use in a browser if you have enabled Row Level Security (RLS) for your tables and configured policies. You may also use the service key which can be found [here](#) to bypass RLS.

JavaScript Dart

```
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = 'https://hitxvvdftdaknicwecn.supabase.co'
const supabaseKey = process.env.SUPABASE_KEY
const supabase = createClient(supabaseUrl, supabaseKey)
```

Figura 12.14: Panel de control de Supabase

Abra nuestra aplicación Next.js e instale el paquete `@supabase/supabase-js`.

Cree una nueva carpeta `utils` en el nivel raíz y cree un archivo `SupabaseClient.js` que inicialice la aplicación con el servidor Supabase:

```
importar { createClient } desde “@supabase/supabase-js”;
```

```
const supabase = crearCliente(  
    supabaseUrl,  
    supabaseKey  
)
```

```
exportar supabase predeterminada;
```

Reemplace `supabaseUrl` y `supabaseKey` con las claves API de su Supabase cuenta.

Actualice `crud/page.js` a:

```
“utilizar cliente”;
```

```
importar estilos desde “./page.module.css”;  
importar Todoltem desde “@/components/ToDoltem”;  
importar Lista desde “@mui/material/List”;  
importar supabase desde “@/utils/SupabaseClient”;
```

```
exportar función asíncrona predeterminada Crud() {  
    deje { datos: elementos, error } = await supabase.from(“elementos”).select(“*”);
```

```
    devolver (  
        <div className={styles.main}>  
            <h2>Elementos por hacer</h2>  
            <Lista>
```

```
{items.map((item) => (
  <ToDoltem item={item} />
))}

</Lista>

</div>

);

}
```

componentes/ToDoItem.jsx:

```
//Componente ToDoltem que representa un único elemento ToDo
```

```
“utilizar cliente”;
```

```
importar React desde “react”;
importar estilos desde “./ToDoItem.module.css”;
importar ListItem desde “@mui/material/ListItem”;
importar ListItemButton desde “@mui/material/ListItemButton”;
importar ListItemIcon desde “@mui/material/ListItemIcon”;
importar Casilla de verificación desde “@mui/material/Checkbox”;
importar ListItemText desde “@mui/material/ListItemText”;
importar DeleteIcon desde “@mui/icons-material/Delete”;
```

```
const ToDoItem = ({ elemento }) => {
  const [marcado, establecidoMarcado] = React.useState([0]);
```

```
const handleToggle = (valor) => () => {
  const currentIndex = check.indexOf(valor);
```

```
const newChecked = [...comprobado];

si (índiceActual === -1) {
    newChecked.push(valor);
} demás {
    newChecked.splice(índiceActual, 1);
}

establecerComprobado(nuevoComprobado);
};

devolver (
<ListItemIcon className={styles.item} clave={item.id}>
<ListItemIconButton onClick={handleToggle(item.id)}>
<íconoDeElementoDeLista>
<Casilla de verificación
comprobado={comprobado.índiceDe(item.id) !== -1}
índice de tabulación={-1}
nombreDeClase={styles.checkbox}
/>
</íconoDeElementoDeLista>
<ListItemText primary={item.name} />
</BotónDeElementoDeLista>
<íconoDeElementoDeLista>
<DeleteIcon className={styles.delete} />
</íconoDeElementoDeLista>
</ListItemIcon>
);

});
```

exportar predeterminado ToDoltem;

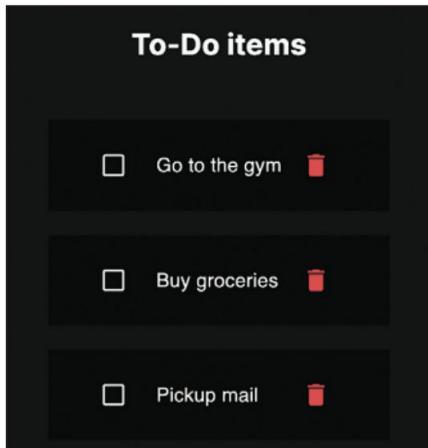


Figura 12.15: Elementos de la tarea pendiente generados a partir de los datos del servidor

En este paso, realizamos una llamada API para obtener datos de nuestro servidor Supabase, que ahora se utiliza para representar los elementos de Todo.

Agregar nuevos elementos de tareas pendientes (Crear)

En las secciones anteriores, implementamos la función de lectura y conectamos nuestra aplicación Next.js al backend de Supabase. En esta sección, intentaremos añadir un nuevo elemento de la lista de tareas pendientes y actualizar la tabla del backend de Supabase.

Como primer paso, agregaremos un campo de entrada y un botón dd que agrega un elemento de la lista de tareas pendientes:

```
crud/page.js
```

```
"utilizar cliente";
```

```
importar estilos desde "./page.module.css";
```

```
importar Todoltem desde "@/components/ToDoItem";
```

```
importar Lista desde "@mui/material/List";
```

```
importar supabase desde "@/utils/SupabaseClient";
```

```
exportar función asíncrona predeterminada Crud() {  
deje { datos: elementos, error } = await supabase.from("elementos").select("*");  
  
devolver (  
<div className={styles.main}>  
<h2>Elementos por hacer</h2>  
<div className={estilos.contenedor}>  
<nombreClaseInput={estilos.input} />  
<button className={styles.button}>¡Aregar a la lista!</button>  
</div>  
<Lista>  
{items.map((item) => (  
<TodolItem item={item} clave={item.key} />  
))}  
</Lista>  
</div>  
);  
}
```

crud/page.module.css

```
.principal {  
pantalla: ej;  
exdirección: columna;  
espacio: 40px;  
alinear-elementos: centro;  
relleno: 6rem;  
altura mínima: 100vh;  
}  
  
.botón de ir {
```

```
fondo: verde;  
relleno: 10px;  
}  
.aporte {  
ancho: 250px;  
tamaño de fuente: 20px;  
relleno: 5px;  
}  
  
.botón {  
relleno: 12px;  
peso de la fuente: 600;  
color de fondo: azul claro;  
}  
  
.contenedor {  
pantalla: ej;  
espacio: 12px;  
}
```

La interfaz de usuario renderizada se verá así:

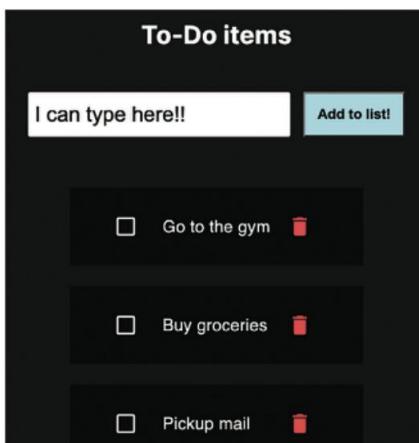


Figura 12.16: Elementos de la tarea pendiente generados a partir de los datos del servidor

Ahora, creemos la lógica que añade lo que escribimos en el campo de entrada a la lista de tareas pendientes cuando el usuario hace clic en el botón ¡Añadir a la lista!. Refactoricemos un poco el código y añadamos una nueva función para gestionar la adición de nuevos elementos a nuestra lista:

crud/page.js

“utilizar cliente”;

```
importar eact, { useState, useEffect } de react;
importar { v4 como uuidv4 } desde “uuid”;
importar estilos desde “./page.module.css”;
importar Todoltem desde “@/components/ToDoltem”;
importar Lista desde “@mui/material/List”;
importar supabase desde “@/utils/SupabaseClient”;
```

```
exportar función predeterminada Crud()
{ const [newItem, setNewItem] = useState(null); const
[items, setItems] = useState([]);
```

```
const fetchData = async () => { const
{ datos, error } = await supabase.from(“items”).select(“*”); setItems(datos);
};
```

```
usarEfecto(() => {
obtener datos();
}, []);
constante addlItem = async () => {
const { datos, error } = await supabase
.from(“elementos”)
.insert([{is_completed: false, nombre: newItem, id: uuidv4()}])
```

```
.soltero();
si (!error) {
obtener datos();
}
};

devolver (
<div className={styles.main}>
<h2>Elementos por hacer</h2>
<div className={estilos.contenedor}>
<entrada
nombreDeClase={styles.input}
onChange={(e) => setNewItem(e.objetivo.valor)}
/>
<button className={styles.button} onClick={addItem}>
¡Añadir a la lista!
</botón>
</div>
<Lista>
{items.map((item, índice) => (
<Todoltem item={item} clave={índice} />
))}
</Lista>
</div>
);
}
```

Ten en cuenta que hemos instalado una biblioteca llamada `uuidv4` para generar un UUID. Al escribir "Jugar fútbol" en el campo de entrada y hacer clic en el botón "¡Añadir a la lista!", se enviará una llamada API a la base de datos para añadir este nuevo elemento y actualizar la página para que podamos verlo en nuestra interfaz de usuario.

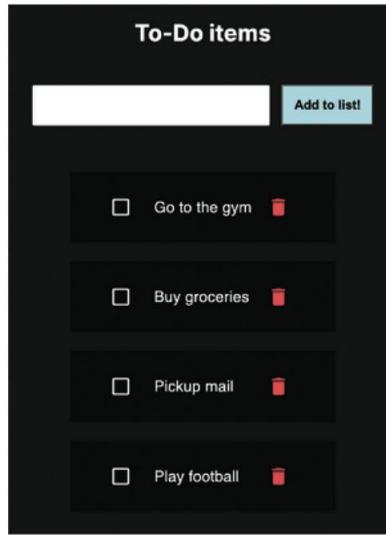


Figura 12.17: Nuevo elemento de tarea pendiente agregado

Edición de elementos de tareas pendientes (Actualización)

En esta parte, implementaremos si una tarea está completa o no según si está marcada en la interfaz de usuario. Si el usuario marca o desmarca una tarea pendiente, la actualizaremos en la tabla Supabase. Para ello, refactorizaremos ambos archivos de la siguiente manera.

crud/page.js

“utilizar cliente”;

```
importar eact, { useState, useEffect } de react;
importar { v4 como uuidv4 } desde “uuid”;
importar estilos desde “./page.module.css”;
importar Todoltem desde “@/components/Todoltem”;
importar Lista desde “@mui/material/List”;
importar supabase desde “@/utils/SupabaseClient”;
```

```
exportar función predeterminada Crud() {  
  const [newItem, setNewItem] = useState(null);  
  const [elementos, setItems] = useState([]);  
  
  
constante fetchData = async () => {  
  const { datos, error } = await supabase.from("items").select("*");  
  setItems(datos);  
};  
  
  
usarEfecto(() => {  
  obtener datos();  
}, []);  
  
  
const handleToggle = async (elemento) => {  
  const { datos, error } = await supabase  
  
.from("items") .update({ is_completed: !  
item.is_completed }) .eq("id", item.id);  
  si (!error) {  
    obtener datos();  
  }  
};  
  
  
constante addItem = async () => {  
  const { datos, error } = await supabase  
.from("elementos")  
.insert([{is_completed: false, nombre: newItem, id: uuidv4() }])  
.soltero();
```

```
si (!error) {  
    obtener datos();  
}  
};
```

```
devolver (  
    <div className={styles.main}>  
        <h2>Elementos por hacer</h2>  
        <div className={estilos.contenedor}>  
            <entrada  
                nombreDeClase={styles.input}  
                onChange={(e) => setNewItem(e.objetivo.valor)}  
            />  
            <button className={styles.button} onClick={addItem}>  
                ¡Añadir a la lista!  
            </botón>  
        </div>  
        <Lista>  
            {items.map((item) => (  
                <TodolItem item={item} key={item.id} onToggle={handleToggle} />  
            ))}  
        </Lista>  
    </div>  
);  
}
```

componentes/TodoItem.jsx

“utilizar cliente”;

```
importar React desde "react";
importar estilos desde "./Todoltem.module.css";
importar ListItem desde "@mui/material/ListItem";
importar ListItemButton desde "@mui/material/ListItemButton";
importar ListItemIcon desde "@mui/material/ListItemIcon";
importar Casilla de verificación desde "@mui/material/Checkbox";
importar ListItemText desde "@mui/material/ListItemText";
importar DeleteIcon desde "@mui/icons-material/Delete";
```

```
const ToDoltem = ({ elemento, onToggle }) => {
```

```
    const handleToggle = () =>
    { onToggle(item);
    };
```

```
    devolver (
        <ListItem className={styles.item}>
            <ListItemButton onClick={manejadorAlternar}>
                <íconoDeElementoDeLista>
                    <Casilla de verificación
                        comprobado={item.is_completed}
                        índice de tabulación={-1}
                    nombreDeClase={styles.checkbox}
                />
                </íconoDeElementoDeLista>
            <ListItemText primary={item.name} />
        </BotónDeElementoDeLista>
```

```
<IconoDeElementoDeLista>

<DeleteIcon className={styles.delete} />

</IconoDeElementoDeLista>

</ListItem>

);

};

};
```

```
exportar predeterminado ToDoItem;
```

Eliminar elementos de tareas pendientes (Eliminar)

Finalmente hemos llegado a la funcionalidad final, es decir, eliminar elementos cuando Presione el botón eliminar .

```
crud/page.js
```

```
“utilizar cliente”;
```

```
importar eact, { useState, useEffect } de react;
importar { v4 como uuidv4 } desde “uuid”;
importar estilos desde “./page.module.css”;
importar TodoItem desde “@/components/ToDoItem”;
importar Lista desde “@mui/material/List”;
importar supabase desde “@/utils/SupabaseClient”;
```

```
exportar función predeterminada Crud() {
const [newItem, setNewItem] = useState(null);
```

```
const [elementos, setItems] = useState([]);
```

```
constante fetchData = async () => {
  const { datos, error } = await supabase.from("items").select("*");
  setItems(datos);
};
```

```
usarEfecto(() => {
  obtener datos();
}, []);
```

```
const handleToggle = async (item) => {
  const
  { datos, error } = await supabase
```

```
.from("items") .update({ is_completed: !
  item.is_completed }) .eq("id", item.id);
  si (!error)
  { fetchData();
  }
};
```

```
constante handleDelete = async (id) => {
  const { datos, error } = await supabase.from("items").delete().eq("id", id);

  si (!error) {
    obtener datos();
  }
};
```

```
constante addlItem = async () => {
  const { datos, error } = await supabase
    .from("elementos")
    .insert([{is_completed: false, nombre: newItem, id: uuidv4()}])
    .soltero();
  si (!error) {
    obtener datos();
  }
};
```

```
devolver (
  <div className={styles.main}>
    <h2>Elementos por hacer</h2>
    <div className={estilos.contenedor}>
      <entrada
        nombreDeClase={styles.input}
        onChange={(e) => setNewItem(e.objetivo.valor)}
      />
      <button className={styles.button} onClick={addlItem}>
        ¡Añadir a la lista!
      </botón>
    </div>
    <Lista>
      {items.map((item) => (
        <TodoArtículo
          artículo={artículo}
          clave={item.id}
          onToggle={manejarResultar}
          onDelete={manejarResultar}
        </TodoArtículo>
      ))}
    </Lista>
  </div>
)
```

```
/>
))>
</Lista>
</div>
);
}
componentes/ToDoItem.jsx
```

“utilizar cliente”;

```
importar React desde “react”;
importar estilos desde “./ToDoItem.module.css”;
importar ListItem desde “@mui/material/ListItem”;
importar ListItemButton desde “@mui/material/ListItemButton”;
importar ListItemIcon desde “@mui/material/ListItemIcon”;
importar Casilla de verificación desde “@mui/material/Checkbox”;
importar ListItemText desde “@mui/material/ListItemText”;
importar DeleteIcon desde “@mui/icons-material/Delete”;
```

```
const ToDoItem = ({ elemento, onToggle, onDelete }) => {
constante handleToggle = () => {
onToggle(elemento);
};
```

```
constante handleDelete = () => {
onDelete(item.id);
};
```

```
devolver (   
    <ListItem className={styles.item}>   
        <ListItemIcon onClick={manejarAlternar}>   
            <IconoDeElementoDeLista>   
            <Casilla de verificación  
                comprobado={item.is_completed}>   
                índice de tabulación={-1}>   
            nombreDeClase={styles.checkbox}>   
        />   
        </IconoDeElementoDeLista>   
        <ListItemText primary={item.name} />   
        <BotónDeElementoDeLista>   
            <ListItemIcon onClick={manejarEliminar}>   
                <DeleteIcon className={styles.delete}> />   
            </IconoDeElementoDeLista>   
        </ListItemIcon>   
    </ListItem>   
);   
};
```

exportar predeterminado ToDoltem;

Al hacer clic en el botón Eliminar, se ejecutará una consulta de Supabase para eliminar ese elemento específico de la tabla de Supabase y actualizará la interfaz de usuario.

Implementación de la aplicación

Bueno, finalmente hemos añadido todas las funciones necesarias a nuestra aplicación.

Sin embargo, nuestra aplicación se ejecuta localmente en nuestra computadora y sería fantástico si pudiéramos compartir nuestra aplicación web con otras alojadas en un sitio web.

¡Eso es exactamente lo que vamos a hacer en esta sección!

Primero, cargaremos nuestro repositorio en nuestra cuenta remota de GitHub:

```
git remote add origin [la dirección de tu repositorio de GitHub]  
rama git -M principal  
git push -u origin main
```

Después, ve a vercel.com e inicia sesión con tu cuenta de GitHub. Importa el repositorio de la aplicación "todo-app" desde la lista y haz clic en "Implementar". ¡Listo! ¡Has implementado correctamente tu aplicación Next.js en un sitio web!

Conclusión

En este capítulo, desarrollamos una aplicación CRUD (Crear, Leer, Actualizar, Eliminar) integral con Next.js 13 y Supabase. Esta aplicación permite a los usuarios gestionar una lista de tareas creando nuevas, marcándolas como completadas y eliminándolas. Durante este proceso, demostramos cómo utilizar eficientemente las robustas funciones de Next.js, como su framework basado en React para renderizar componentes. También ilustramos cómo Supabase, con su potente y fácil de usar biblioteca del lado del cliente, simplifica la gestión de las operaciones de datos en el backend. La combinación de Next.js y Supabase nos permite crear de forma rápida y eficiente aplicaciones escalables sin servidor con experiencias de usuario mejoradas. Este proyecto mostró solo una muestra de su potencial, fomentando una mayor exploración y dominio de estas tecnologías para satisfacer diversas necesidades de desarrollo web.

En el próximo capítulo, analizaremos las diferentes formas de implementar una aplicación Next.js mediante el uso de proveedores de nube y servicios de alojamiento.

Capítulo 13

Explorando la arquitectura de implementación en aplicaciones Next.js

Implementar una aplicación Next.js de forma eficaz es fundamental en el ciclo de desarrollo, ya que garantiza que su aplicación web llegue a su público objetivo de forma segura y eficiente. En este capítulo, profundizaremos en el ámbito de la arquitectura de implementación, explorando las múltiples opciones disponibles para implementar una aplicación Next.js y ofreciendo las mejores prácticas para optimizar el proceso.

Para comenzar, descifraremos el proceso de implementación en Next.js y comprenderemos los pasos necesarios para llevar su aplicación del desarrollo al entorno de producción. Este conocimiento básico sentará las bases para explorar las diversas opciones y consideraciones de implementación.

Nos embarcaremos en un recorrido por diferentes plataformas de alojamiento, como Vercel, AWS y Heroku, entre otras. Descubrirás las ventajas y características de cada plataforma y aprenderás a implementar tu aplicación Next.js sin problemas en estos proveedores de nube. Te guiaremos a través de las configuraciones y los pasos necesarios para garantizar una implementación sin problemas.

A medida que prepara su aplicación para producción, profundizaremos en la tarea crucial de configurar variables de entorno diseñadas específicamente para la implementación. Comprenderá la importancia de las variables de entorno en la gestión de información confidencial y aprenderá cómo configurarlas de manera eficaz para su Next. Implementación de la aplicación js.

Optimizar el proceso de implementación es esencial para un lanzamiento rápido y eficiente. Exploraremos estrategias para optimizar el proceso de implementación, centrándonos en técnicas que mejoran el rendimiento y reducen el tiempo de inactividad. Al implementar

Con estas prácticas recomendadas, puede garantizar una experiencia de usuario perfecta y minimizar las interrupciones durante la fase de implementación.

En la era de la integración continua y el despliegue continuo (CICD), le guiaremos en el proceso de configuración de pipelines diseñados específicamente para aplicaciones Next.js. Aprenderá a aprovechar las herramientas y frameworks de CICD para automatizar el proceso de despliegue, lo que permite despliegues más rápidos y eficientes, manteniendo la calidad y la estabilidad del código.

La supervisión y la depuración de aplicaciones implementadas son cruciales para mantener su salud y abordar los problemas rápidamente. Profundizaremos en las técnicas y herramientas de supervisión, lo que le permitirá supervisar de forma proactiva el rendimiento de su aplicación Next.js implementada y abordar cualquier posible cuello de botella o error de manera efectiva.

A lo largo de este capítulo, cubriremos temas que incluyen la comprensión del proceso de implementación en Next.js, la implementación de aplicaciones Next.js en diferentes plataformas de alojamiento, la configuración de variables de entorno para la implementación, la configuración de aplicaciones Next.js para producción, la optimización del proceso de implementación para la eficiencia, la configuración de canalizaciones CICD para aplicaciones Next.js y el monitoreo y la depuración de aplicaciones implementadas.

Al finalizar este capítulo, comprenderá a fondo las opciones de implementación disponibles para las aplicaciones Next.js, lo que le permitirá tomar decisiones informadas sobre la mejor estrategia de implementación para su caso de uso específico. ¡Embárquese en este recorrido por la arquitectura de implementación y asegúrese de que su aplicación Next.js alcance su máximo potencial en el entorno real!

Estructura

En este capítulo cubriremos los siguientes temas

- Comprender el proceso de implementación en Next.js • Configurar variables de entorno para la implementación • Configurar la aplicación Next.js para producción • Configurar canalizaciones CICD de integración continua e implementación continua para aplicaciones Next.js • Implementar aplicaciones Next.js en diferentes plataformas de alojamiento, incluidas Vercel, AWS y Heroku optimizan
- el proceso de implementación para lograr implementaciones más rápidas y eficientes • Monitoreo y depuración de aplicaciones implementadas

Comprender el proceso de implementación en Next.js

En el contexto del desarrollo de software, la implementación se refiere al proceso de poner una aplicación de software a disposición de los usuarios. Implica preparar el código desarrollado para su ejecución en servidores u otros dispositivos informáticos, permitiendo a los usuarios acceder a la aplicación e interactuar con ella.

Durante el proceso de implementación, se realizan diversas tareas, como compilar código, optimizar recursos, configurar ajustes y configurar la infraestructura necesaria. El objetivo es garantizar que la aplicación funcione de forma fluida, segura y eficiente en el entorno previsto.

Tipos de entornos de implementación

Un entorno de implementación se refiere a la infraestructura informática donde se implementa una aplicación de software y se pone a disposición de los usuarios. Existen varios tipos de entornos de implementación, cada uno con diferentes propósitos. A continuación, se presentan algunos tipos comunes.

- Entorno de desarrollo

El entorno de desarrollo es donde los desarrolladores de software trabajan en la codificación, las pruebas y la depuración de la aplicación. Normalmente se ejecuta en las máquinas de cada desarrollador y puede incluir herramientas de desarrollo, bibliotecas y bases de datos. Este entorno no está destinado al uso público, sino que sirve como espacio aislado para la creación y el perfeccionamiento de la aplicación.

- Entorno de prueba/preparación

El entorno de pruebas o staging es un entorno dedicado a probar exhaustivamente la aplicación antes de su lanzamiento a producción. Estimula el entorno de producción al máximo y permite probar diferentes casos de uso, detectar y corregir errores, y garantizar que la aplicación se comporte como se espera. Ayuda a validar la funcionalidad, el rendimiento y la compatibilidad de la aplicación con diferentes sistemas.

- Entorno de producción

El entorno de producción es el entorno real donde la aplicación se implementa y se pone a disposición de los usuarios. Es el servidor o la infraestructura de alojamiento donde la aplicación se ejecuta y sirve al público objetivo. El entorno de producción debe ser altamente estable, seguro y...

Optimizado para un rendimiento óptimo en situaciones reales. A menudo requiere recursos y configuraciones adicionales para garantizar la fiabilidad, la escalabilidad y la alta disponibilidad.

- Plataformas de implementación

Las plataformas de implementación son servicios o proveedores de alojamiento que facilitan la implementación de aplicaciones. Estas plataformas ofrecen diversas funciones y admiten diferentes métodos de implementación. Algunas plataformas de implementación populares incluyen proveedores de nube como Amazon EB Services, Microsoft Azure u Google Cloud Platform, así como servicios de alojamiento especializados como Netlify, Vercel o Heroku. Estas plataformas proporcionan infraestructura, herramientas y servicios para simplificar el proceso de implementación y administrar la aplicación en diferentes entornos.

Es importante tener en cuenta que cada entorno de implementación cumple un propósito específico en el ciclo de vida del desarrollo de software, garantizando que la aplicación se pruebe exhaustivamente y funcione bien antes de lanzarse al entorno de producción donde los usuarios pueden acceder a ella y utilizarla.

En general, la implementación implica preparar y configurar la aplicación en diferentes entornos, cada uno de los cuales cumple una función única en el proceso de desarrollo y lanzamiento.

Diferentes tipos de procesos de implementación en Next.js:

- Alojamiento estático

Si su aplicación Next.js se genera estáticamente y no requiere renderizado del lado del servidor ni rutas API, puede implementarla como un conjunto de archivos estáticos. Este enfoque es adecuado para plataformas de alojamiento como Netlify, Vercel o GitHub Pages. Para implementar su aplicación utilizando alojamiento estático, siga estos pasos:

Cree su aplicación Next.js ejecutando el siguiente comando

```
npm ejecutar compilación
```

. Genere los archivos estáticos ejecutando el siguiente comando

```
npm ejecutar exportación
```

Cargue los archivos estáticos generados en la plataforma de alojamiento elegida, ya sea vinculando el repositorio de su proyecto o utilizando un método de transferencia de archivos.

. Configure su plataforma de alojamiento para servir los archivos estáticos y hacer Su aplicación accesible para los usuarios.

- Funciones sin servidor

Next.js ofrece una función de rutas de PI que permite definir funciones sin servidor, que se ejecutan bajo demanda y gestionan la lógica del lado del servidor. Este método es útil cuando la aplicación requiere procesamiento del lado del servidor, como el envío de formularios o la obtención de datos de PI externos. Puede implementar su aplicación Next.js con funciones sin servidor en plataformas como Vercel, AWS Lambda o Azure Functions. A continuación, se presenta un resumen general del proceso de implementación.

- Escriba sus funciones sin servidor creando archivos de ruta PI en el Directorio 'pages/api' de su proyecto Next.js.

Cree su aplicación Next.js ejecutando el siguiente comando

```
npm ejecutar compilación
```

3. Implemente su aplicación en la plataforma de alojamiento elegida, asegurándose de que la plataforma reconozca las rutas PI y las implemente junto con el código de la aplicación.

- . Configure cualquier variable de entorno o implementación necesaria configuraciones específicas de la plataforma de alojamiento.

- Servidor personalizado

Si su aplicación Next.js requiere capacidades avanzadas del lado del servidor o integración con una infraestructura específica, puede implementarla mediante un servidor personalizado. Este enfoque le brinda mayor control sobre el entorno del servidor y resulta útil cuando necesita implementar enruteamiento complejo o interactuar con bases de datos o servicios externos. El proceso de implementación de un servidor personalizado generalmente implica los siguientes pasos.

Cree un archivo de servidor personalizado, como server.js, en su proyecto Next.js.

Implementar la lógica del servidor necesaria, incluido el enruteamiento y cualquier funcionalidad adicional del lado del servidor.

Cree su aplicación Next.js ejecutando el siguiente comando

```
npm ejecutar compilación
```

Inicie el servidor personalizado ejecutando el siguiente comando

```
servidor de nodo.js
```

5. Implemente su servidor personalizado en el entorno de alojamiento elegido, asegurándose de que el script del servidor se ejecute correctamente.

Es importante tener en cuenta que los procesos de implementación pueden variar según la plataforma de alojamiento o la infraestructura específica que elija. Por lo tanto, se recomienda consultar la documentación o las guías proporcionadas por la plataforma o el servicio de alojamiento que planea utilizar para implementar su aplicación Next.js.

Configuración de variables de entorno para la implementación

Las variables de entorno desempeñan un papel crucial en la implementación de aplicaciones. Son valores de configuración externos al código de la aplicación y pueden variar según el entorno de implementación. Ofrecen una forma flexible de almacenar información confidencial, ajustes de configuración, claves PI, credenciales de base de datos o cualquier otro valor dinámico requerido por la aplicación.

Configurar variables de entorno para la implementación implica definirlas y gestionarlas en diferentes entornos, como desarrollo local, pruebas y producción. Exploraremos cómo configurar variables de entorno para la implementación en cada uno de estos entornos.

1. Entorno local

En el entorno de desarrollo local, normalmente se configuran variables de entorno para configurar el entorno de desarrollo y probar la aplicación en el equipo local. A continuación, se explica cómo configurar las variables de entorno para la implementación local.

- a. Cree un archivo `.env` en el directorio raíz de su proyecto Next.js. Este archivo almacenará sus variables de entorno.
- b. Defina sus variables de entorno en el archivo `.env` usando el formato Y. Por ejemplo

```
CLAVE API=12345
```

```
URL_DE_LA_BASE_DE_DATOS=mongodb://localhost:27017/mibasededatos
```

- c. En el código de su aplicación Next.js, acceda a las variables de entorno mediante un paquete como `'dotenv'` o el objeto `'process.env'` integrado . Por ejemplo, para acceder a la variable `'API_KEY'`

```
constante apiKey = proceso.env.API_KEY;
```

- d. Asegúrese de que el archivo `.env` se agregue a su archivo `'.gitignore'` para que no se envíe al control de versiones, ya que puede contener información confidencial.

2. Entorno de prueba

El entorno de pruebas es donde se implementa la aplicación para realizar pruebas exhaustivas antes de implementarla en producción. El proceso de configuración de variables de entorno para el entorno de pruebas es similar al del entorno local.

- a. Cree un archivo `.env.test` separado para almacenar las variables de entorno específicas del entorno de prueba.
- b. Defina las variables de entorno necesarias para realizar pruebas en el `.env` archivo de prueba.
- c. En los scripts de prueba o en la configuración del marco de prueba, cargue las variables de entorno del archivo ``.env.test`` para garantizar que las pruebas utilicen la configuración correcta.

3. Entorno de producción

El entorno de producción es el entorno de implementación final donde su aplicación se pone a disposición de los usuarios. Es fundamental configurar las variables de entorno de forma segura y eficiente. Este es un enfoque recomendado.

- a. En el servidor de producción o la plataforma de alojamiento, defina las variables de entorno mediante los mecanismos proporcionados por el servicio de alojamiento. Esto podría incluir el uso de una consola de administración, un panel de control o una interfaz de línea de comandos específica de la plataforma de alojamiento.
- b. Evite almacenar variables de entorno de producción directamente en el código base o en los archivos de configuración. En su lugar, configure las variables de entorno de forma segura en el servidor de producción o mediante la configuración de implementación de la plataforma de alojamiento.
- c. En el servidor de producción, asegúrese de que las variables de entorno estén correctamente cargadas y sean accesibles para su aplicación Next.js durante el tiempo de ejecución.

Siguiendo estos pasos, puede configurar variables de entorno para la implementación en diferentes entornos. Recuerde gestionar la información confidencial de forma segura y evitar exponer las variables de entorno en repositorios públicos o canales inseguros.

Puede pasar variables de entorno en el script al ejecutar su aplicación Next.js localmente en entornos Windows y Linux:

- Entorno de Windows

En Windows, puedes configurar variables de entorno en línea antes de ejecutar el script usando el comando `set` . Aquí hay un ejemplo.

```
establecer API_KEY=12345 y ejecutar npm dev
```

En el ejemplo anterior, `API_KEY` se establece en `12345` antes de ejecutar el script `npm run dev` . Puede reemplazar `API_KEY` y su valor con su variable de entorno específica.

- Entorno Linux

En inux, puedes configurar variables de entorno en línea antes de ejecutar el script usando el comando `export` . Aquí hay un ejemplo.

```
exportar API_KEY=12345 && npm ejecutar dev
```

Al igual que en el ejemplo de Windows, `API_KEY` se establece en `12345` antes de ejecutar el script `npm run dev` . Nuevamente, puede reemplazar `API_KEY` y su valor con su variable de entorno específica.

Alternativamente, puede almacenar las variables de entorno en un archivo separado y cargarlas antes de ejecutar el script.

- Método de archivos separados en el entorno de Windows

En Windows, puede crear un archivo separado, por ejemplo, env.bat, y usar el comando `set` para definir sus variables de entorno. Aquí hay un ejemplo.

```
@echo apagado
establecer API_KEY=12345
establecer DATABASE_URL=mongodb://localhost:27017/mydatabase
```

Guarde el archivo y luego ejecute el script ejecutándolo antes de ejecutar el script npm run dev

```
llamar a env.bat y npm ejecutar dev
```

- Método de archivos separados en el entorno Linux

En inux, puedes crear un archivo separado, por ejemplo, env.sh` , y usar el comando `export` para definir tus variables de entorno. Aquí tienes un ejemplo.

```
#!/bin/bash
exportar API_KEY=12345
```

```
exportar DATABASE_URL=mongodb://localhost:27017/misdatos-
base
```

Guarde el archivo y luego ejecute el script ejecutándolo antes de ejecutar el script `npm run dev`

fuente env.sh y npm run dev

Al utilizar el método en línea o el método de archivo separado, puede pasar variables de entorno al script cuando ejecuta su aplicación Next.js localmente en entornos Windows y Linux.

Paso 1: Configurar un archivo de configuración de flujo de trabajo de TI para probar, construir e implementar CI/CD para su aplicación Next.js:

te Crea uno archivo orflow

- En el repositorio de su proyecto Next.js en itub, navegue al directorio .githubworkflows.
- Cree un nuevo archivo Y, por ejemplo, ci-cd.yml en .github Directorio de flujos de trabajo para definir su flujo de trabajo.

te De finir a los aparejadores de flujo

- Especifique los eventos que activan su flujo de trabajo. Por ejemplo, puede activarlo al enviar eventos a la rama principal o al solicitar la incorporación de cambios.

nombre: /D Workow

en:

empujar:

sucursales:

- principal

solicitud de extracción:

sucursales:

te De finir os de flujo

- Define tareas dentro de tu flujo de trabajo. Por ejemplo, puedes tener tareas separadas para pruebas, desarrollo e implementación.

trabajos:

prueba:

nombre: Prueba

se ejecuta en: ubuntu-latest

pasos:

- nombre: Código de pago

usos: acciones/checkout@v2

- nombre: Instalar dependencias

ejecutar: npm ci

- nombre: Ejecutar pruebas

ejecutar: npm ejecutar prueba

construir:

nombre: Construir

se ejecuta en: ubuntu-latest

pasos:

- nombre: Código de pago

usos: acciones/checkout@v2

- nombre: Instalar dependencias

ejecutar: npm ci

- nombre: Crear la aplicación Next.js

ejecutar: npm run build

desplegar:

nombre: Implementar

necesidades: [construir]

se ejecuta en: ubuntu-latest

pasos:

- nombre: Código de pago

usos: acciones/checkout@v2

- nombre: Instalar dependencias

ejecutar: npm ci

- nombre: Implementar en AWS/Heroku/Vercel

correr: |

Agregue comandos de implementación aquí

te Con igre esting

- dd comandos y configuraciones de prueba apropiados dentro del trabajo de "prueba".

Te confieso

- Comandos y configuraciones dd para construir su Next.js aplicación dentro del trabajo 'build' .

Te Configre Deloment

- Agregue los comandos o scripts de implementación necesarios dentro del trabajo de implementación . Dependiendo de su destino de implementación (S, eroku, ercel), deberá usar los comandos y las configuraciones adecuados.

te Confirmar un archivo de flujo de trabajo

- Confirme y envíe el archivo de flujo de trabajo ci-cd.yml a su repositorio.

Paso 8: Ejecución de acciones de GitHub:

- Las acciones de itub ejecutarán automáticamente el flujo de trabajo definido siempre que se activen los eventos especificados, por ejemplo, en solicitudes de inserción o extracción a la rama "principal".
- El flujo de trabajo ejecutará los trabajos definidos de forma secuencial, por ejemplo, probar, compilar e implementar en función de las dependencias especificadas mediante `necesidades` en el trabajo `implementar`.

Listo. Has configurado un archivo de configuración de acciones de flujo de trabajo de itub para probar, compilar e implementar CICD en tu aplicación Next.js. Personaliza el archivo de flujo de trabajo según tus requisitos específicos y objetivos de implementación.

Implementación de aplicaciones Next.js en diferentes plataformas de alojamiento

La implementación de aplicaciones Next.js en diferentes plataformas de alojamiento implica algunos pasos específicos para cada plataforma. Esta descripción general incluye el proceso de implementación para Vercel, Heroku y AWS, junto con ejemplos de código para ayudar a navegar a través del proceso.

1. Implementación de aplicaciones Next.js en Vercel

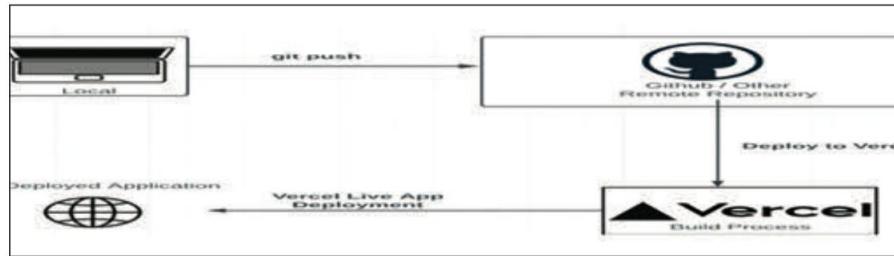


Figura 13.1: Proceso de implementación de Vercel

Para implementar una aplicación Next.js usando Vercel, realice los siguientes pasos

Paso 1: Instalar la CLI de Vercel globalmente

Asegúrate de tener instalados Node.js y npm. A continuación, abre la terminal y ejecuta el siguiente comando.

`npm install -g vercel`

te la **aliación de Next.js**

Navegue al directorio de su proyecto Next.js en la terminal y ejecute el siguiente comando para compilar su aplicación

`npm ejecutar compilación`

te Crea un archivo `vercel.json`

Cree un archivo `vercel.json` en el directorio raíz de su proyecto y agregue las configuraciones deseadas. Aquí hay un ejemplo de un archivo `vercel.json` básico

```
{
  "versión": 2,
  "construye": [
    {
      "fuente": next.config.s,
      "uso": "@vercel/next"
    }
  ],
  "rutas": [
    { "origen": "/api/(.*)", "destino": "api/$1" },
    { "origen": "/blog", "destino": "/blog/index.html" }
  ]
}
```

```

    },
    {
      "src": "/blog/(.*)",
      "dest": "/blog/$1.html"
    },
    {
      "origen": "/(.*)",
      "destino": "/$1"
    }
  ],
  "env": {
    "API_URL": "https://api.ejemplo.com",
    "API_KEY": "@env-api-key"
  },
  reescribe: [
    {
      "origen": "/usuario/:id",
      "destino": "/perfil?id=:id"
    }
  ],
  "redirecciones": [
    {
      "origen": "/acerca de",
      "destino": "/compañía/acerca de",
      "código de estado": 301
    }
  ]
}

```

Analicemos las diferentes secciones de la configuración de `vercel.json`

- `versión`: especifica la versión del archivo de configuración. se para el **última versión.**
- `builds`: Define la configuración de compilación de tu aplicación Next.js. En este ejemplo, usamos el compilador vercelnext, el compilador predeterminado para proyectos Next.js.
- `routes`: Configura las reglas de enrutamiento para su aplicación. Cada ruta tiene una propiedad `src` y `dest`. En este ejemplo, definimos rutas para puntos finales de PI, páginas de blog y una ruta general.
- `env`: le permite definir variables de entorno para su aplicación.
Puede especificar valores personalizados o utilizar la sintaxis `@env-variable` para cargar el valor de las variables de entorno del proyecto Vercel.
- `rewrites`: Especifica las reescrituras de R para su aplicación. En este ejemplo, reescribimos user id a profile?userid.
- `redirects`: configura redirecciones R. En este ejemplo, las solicitudes a about se redirigirán a companyabout con un código de estado.

Paso 4: Inicie sesión en su cuenta de Vercel a través de la CLI

Ejecute el siguiente comando y siga las instrucciones de autenticación

```
inicio de sesión en vercel
```

Paso 5: Implementar la aplicación

En la terminal, navegue hasta el directorio de su proyecto y ejecute el siguiente comando

Vercel

Vercel lo guiará a través del proceso de implementación y, una vez completado, le proporcionará una URL de implementación.

2. Implementación de aplicaciones Next.js en Heroku

Para implementar una aplicación Next.js usando erozu, siga estos pasos

Paso 1: Regístrese para obtener una cuenta de Heroku

Si aún no tienes una cuenta de erozu, ve al sitio web de erozu <https://www.heroku.com> y regístrate para obtener una cuenta gratuita.

Paso 2: Instalar la CLI de Heroku

Descargue e instale la interfaz de comandos de erozu CI adecuada para su sistema operativo. Puede encontrar las instrucciones de instalación en el sitio web del Centro de desarrollo de erozu (<https://devcenter.org/>).
[heroku.comartículos/heroku-clidescargar-e-instalar](https://devcenter.org/article/heroku-clidescargar-e-instalar)

Paso 3: Prepare su aplicación para su implementación

Asegúrese de que su proyecto Next.js esté listo para su implementación. Asegúrese de tener un archivo `package.json` en el directorio raíz y de que se incluyan las dependencias necesarias.

Paso 4: Crea una oficina

En el directorio raíz de su proyecto Next.js, cree un archivo llamado oce
Sin ninguna extensión de archivo. Abra la oficina y agregue la siguiente línea

```
web: inicio npm
```

Esta línea le dice a Heroku cómo iniciar su aplicación.

Paso 5: Inicialice Git y confirme su proyecto

En la terminal, navegue hasta el directorio de su proyecto y ejecute los siguientes comandos para inicializarlo y confirmar su proyecto.

```
git init
añadir git .
git commit -m "Confirmación inicial"
```

Paso 6: Inicie sesión en Heroku a través de la CLI

Ejecute el siguiente comando y siga las instrucciones para iniciar sesión en su Cuenta eroku a través de CI

inicio de sesión en heroku

Paso 7: Crea una nueva aplicación Heroku

Ejecute el siguiente comando para crear una nueva aplicación en eroku

```
heroku crear
```

Este comando generará un nombre único para su aplicación.

Paso 8: Implementa tu aplicación en Heroku

Para implementar su aplicación, ejecute el siguiente comando

```
git push heroku maestro
```

Paso 9: Abra la aplicación implementada

Una vez finalizado el proceso de implementación, Heroku te proporcionará una URL de implementación. Puedes abrir tu aplicación implementada ejecutando el siguiente comando.

heroku abierto

Siguiendo estos pasos, podrá implementar correctamente su aplicación Next.js en eroku. Recuerde consultar la documentación oficial de eroku para obtener información sobre pasos o configuraciones adicionales específicas de su aplicación:
<https://devcenter.heroku.com/categories/nodejs-support>

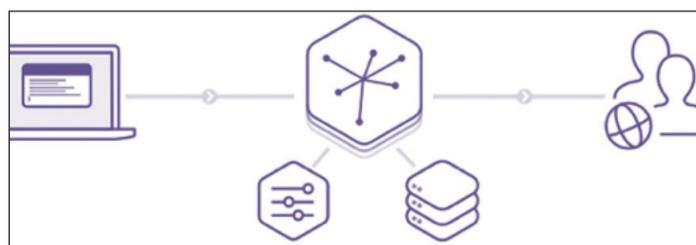


Figura 13.2: Proceso de implementación de Heroku

3. Implementación de aplicaciones Next.js en AWS

Para implementar una aplicación Next.js con AWS Amplify, siga estos pasos detallados, junto con ejemplos de código

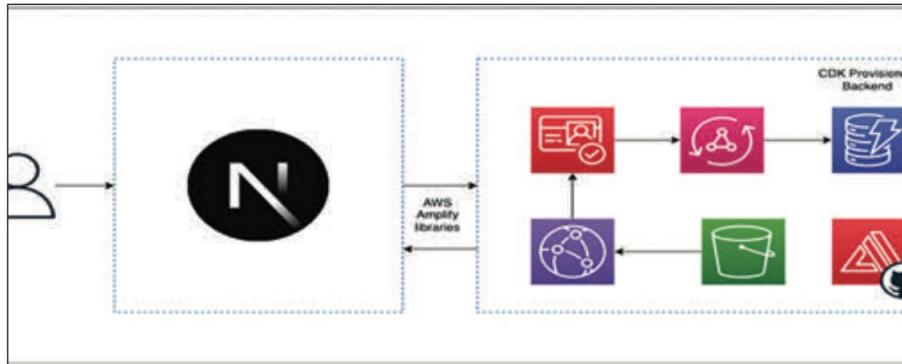


Figura 13.3: Proceso de implementación de AWS

Paso 1: Configurar un proyecto de AWS Amplify

- a. Inicie sesión en el Consola de administración y navegue hasta la servicio Amplify. S
- b. Haga clic en “Comenzar” o “Crear aplicación” para crear una nueva aplicación Amplify.
- c. Elija su proveedor de TI preferido, por ejemplo, itub, itab, Bitbucket y conecta tu repositorio.
- d. Seleccione la rama desde la que desea implementar y especifique la configuración de compilación. Para las aplicaciones Next.js, configure los siguientes valores.
 - i. Marco de referencia
 - ii. il omán npm ejecutar compilación
 - iii. Comando de inicio npm run start
- e. Configure los ajustes del dominio si desea utilizar un dominio personalizado. De lo contrario, Amplify proporcionará un dominio predeterminado para su aplicación.
- f. Revise la configuración y haga clic en “Siguiente” para crear la aplicación Amplify.
 - te y en una configuración de Deloment
 - a. En la consola de amplify, haga clic en su aplicación para acceder a los detalles de la aplicación.
 - b. En Configuración de la aplicación, haga clic en Configuración de compilación para configurar la compilación y configuraciones de implementación.

c. Marque el botón Editar de la rama que desea configurar, generalmente maestra o principal.

d. Asegúrese de que los siguientes ajustes estén configurados correctamente

i. il omán npm ejecutar compilación

ii Comando de inicio npm run start

a. Haga clic en Guardar e implementar para guardar la configuración.

Paso 3: Implementar la aplicación Next.js

a. Después de guardar la configuración, haga clic en “Implementar” para iniciar la implementación. proceso.

b. Amplify clonará automáticamente tu repositorio y compilará tu Next.js. aplicación e impleméntela utilizando recursos de AWS.

c. El progreso de la implementación se mostrará en la consola de Amplify. Espere a que la implementación se complete correctamente.

Paso 4: Acceda a su aplicación Next.js implementada

a. Una vez completada la implementación, Amplify le proporcionará un URL para acceder a su aplicación Next.js implementada.

b. Abra un navegador web y visite la URL proporcionada para ver su aplicación implementada en acción.

Paso 5: Implementación continua (opcional)

Para habilitar la implementación continua, que implementa automáticamente actualizaciones en su aplicación cada vez que envía cambios a su repositorio

a. En la consola amplify, vaya a la configuración de su aplicación.

b. En Ramas, haga clic en la rama para la que desea habilitar la implementación continua.

c. Cambie el interruptor de Implementación manual a Implementación automática.

d. Haga clic en Guardar para habilitar la implementación continua.

Ahora veamos un ejemplo de código para una aplicación Next.js simple:

- En su proyecto Next.js, asegúrese de tener los siguientes scripts en su archivo `package.json`

{

```

    "guiones": {
      "build": "próxima compilación",
      "inicio": "próximo inicio"
    }
  }
}

```

Confirme y envíe su proyecto Next.js al repositorio de TI conectado, por ejemplo, itub.

3. Siga los pasos anteriores para configurar una aplicación AWS Amplify y conectarla a su repositorio.
4. Una vez completada la implementación, Amplify compilará e implementará automáticamente su aplicación Next.js utilizando los scripts especificados.
5. Acceda a su aplicación Next.js implementada utilizando la URL proporcionada.

¡Listo! Has implementado correctamente tu aplicación Next.js con AWS Amplify. Amplify simplifica el proceso de implementación, permitiéndote concentrarte en el desarrollo de tu aplicación.

Optimizar el proceso de implementación para lograr elementos más eficientes

Optimizar el proceso de implementación para lograr implementaciones más rápidas y eficientes implica implementar diversas estrategias y mejores prácticas. Aquí tiene una guía paso a paso para ayudarle a optimizar su proceso de implementación.

te Timie II Roess

- Estos minificadores NLE minimizan sus archivos JavaScript, CSS y otros elementos para reducir su tamaño y mejorar los tiempos de carga. Utilice herramientas como glifyS o eraser para minimizar JavaScript y minificadores CSS para archivos CSS.

División de código : Implemente técnicas de división de código para dividir el código de su aplicación en fragmentos más pequeños. Esto permite la carga diferida de componentes y reduce el tiempo de carga inicial de su aplicación.

- **Habilitar la compresión** Habilite la compresión en su servidor para reducir el tamaño de los archivos transferidos. Zip y Brotli son algoritmos de compresión comúnmente utilizados.
- **Optimizar imágenes** Comprima y optimice las imágenes para reducir su tamaño de archivo. Herramientas como Imageoptim, inyPN o Squoosh pueden ayudar con la optimización de imágenes.

- Utilice la representación del lado del servidor (SSR) Implemente la representación del lado del servidor para pre-renderizar páginas en el servidor y reducir el tiempo de carga inicial para los usuarios.

Paso 2: Implementar el almacenamiento en caché

- Utilice redes de distribución de contenido (CDN) para almacenar en caché activos estáticos y servirlos desde servidores perimetrales más cercanos al usuario, lo que reduce la latencia y mejora el rendimiento.
- Almacenamiento en caché del servidor browser: Establezca encabezados de almacenamiento en caché adecuados para los recursos estáticos a fin de habilitar el almacenamiento en caché del navegador. Esto permite que el navegador almacene recursos localmente y evite solicitudes de red innecesarias.
- Usar encabezados de control de caché Establezca encabezados de control de caché para especificar Directivas de almacenamiento en caché para activos estáticos. Utilice herramientas como CloudFront o Ercel para configurar los ajustes de almacenamiento en caché.

Paso 3: Automatización e integración/implementación continua (CI/CD):

- Configure pipelines de CI/CD utilizando herramientas CICD como Jenkins, CircleCI o GitHub Actions para automatizar el proceso de compilación, prueba e implementación. Configurar canalizaciones para activar implementaciones en caso de cambios de código o cambios específicos eventos.
- Divida su proceso de implementación en pasos paralelos para agilizar la implementación general. Por ejemplo, puede ejecutar pruebas, compilar recursos e implementar en diferentes entornos simultáneamente.
- Utilice implementaciones continuas. Implemente implementaciones continuas para actualizar su aplicación gradualmente en lugar de hacerlo de una sola vez. Esto reduce el tiempo de inactividad y garantiza una transición más fluida para los usuarios.

Implementaciones canarias : Implemente implementaciones canarias para probar nuevas funciones o actualizaciones en un pequeño subconjunto de usuarios antes de implementarlas en toda la base de usuarios. Esto ayuda a detectar problemas de forma temprana y a mitigar riesgos.

Paso 4: Monitoreo y optimización del rendimiento

- Monitorizar las métricas de rendimiento : utilice herramientas como Google Analytics, New Relic o CloudWatch para supervisar métricas de rendimiento como el tiempo de respuesta, el tiempo de carga de la página y el uso de recursos. Identifique los cuellos de botella y las áreas de mejora.
- Optimización continua del rendimiento Analice y optimice continuamente el rendimiento de su aplicación identificando y resolviendo problemas de rendimiento, optimizando las consultas de base de datos y mejorando la eficiencia del código.
- Pruebas de mantenimiento Realice pruebas A/B para experimentar con diferentes versiones de su aplicación y medir su impacto en

Rendimiento. Esto ayuda a identificar cambios que impactan positivamente la experiencia del usuario.

- Error en la obtención de presupuestos de rendimiento Establezca presupuestos de rendimiento para definir límites para métricas como tamaño de archivo, número de solicitudes o tiempo de carga. Asegúrese de que su aplicación se mantenga dentro de estos límites para mantener un rendimiento óptimo.

Si sigue estos pasos, podrá optimizar el proceso de implementación de su aplicación Next.js, lo que dará como resultado implementaciones más rápidas y eficientes.

Supervise y ajuste periódicamente su proceso de implementación para adaptarse a los requisitos cambiantes y aprovechar las últimas tecnologías y las mejores prácticas.

Configurar la aliasación o producción de Next.js

Configurar una aplicación Next.js para producción implica optimizar el rendimiento, la seguridad y la escalabilidad de la aplicación para garantizar que funcione de manera fluida y eficiente en un entorno de producción. Estos son los aspectos clave a considerar al configurar una aplicación Next.js para producción.

- Configuración del entorno

Utilice variables de entorno para almacenar valores de configuración que difieren entre los entornos de desarrollo, ensayo y producción. Estas variables pueden incluir claves PI, cadenas de conexión a bases de datos, indicadores de características o cualquier otra información confidencial. Al separar la configuración del código, puede administrar y actualizar fácilmente la configuración específica de cada entorno sin modificar el código fuente de la aplicación.

- Optimización del rendimiento

Para mejorar el rendimiento, implemente las siguientes técnicas

La inicialización permite la minimización de código para reducir el tamaño de los archivos avaScript y CSS. Esto se puede lograr utilizando herramientas como ebpack o las funciones de optimización integradas de Next.js. La inicialización elimina caracteres innecesarios, espacios en blanco y comentarios, lo que resulta en tiempos de carga más rápidos para los usuarios.

Renderizado del lado del servidor (SSR): Aprovecha las capacidades de renderizado del lado del servidor de Next.js para generar HTML en el servidor antes de enviarlo al cliente. SSR mejora los tiempos de carga iniciales de la página y permite que los motores de búsqueda indexen tu sitio de forma más eficaz.

Optimización del lado del cliente: división de código promedio y carga diferida para garantizar que solo se cargue inicialmente el código necesario. Esto mejora

rendimiento al reducir el tamaño de la carga útil inicial y posponer la carga de recursos no críticos hasta que sean necesarios.

Optimización de imágenes se utiliza el componente de imagen integrado de Next.js, que optimiza automáticamente las imágenes comprimiéndolas y redimensionándolas. Esto reduce el tamaño del archivo de imagen sin sacrificar la calidad visual y mejora los tiempos de carga generales de la página.

Almacenamiento en caché: Implemente mecanismos de almacenamiento en caché para reducir la carga del servidor y mejorar los tiempos de respuesta. Puede aprovechar la compatibilidad integrada de Next.js con el almacenamiento en caché del lado del servidor o usar soluciones de almacenamiento en caché como Varnish o Redis para almacenar en caché los datos a los que se accede con frecuencia o las páginas renderizadas.

- **Medidas de seguridad**

Tome las siguientes medidas de seguridad para proteger su aplicación Next.js

Comunicación segura : aplique PS para todas las comunicaciones entre clientes y servidores. Utilice certificados SS para cifrar los datos en tránsito y evitar el acceso no autorizado o la manipulación.

Validación de entrada: valide y desinfecte la entrada del usuario para evitar vulnerabilidades de seguridad comunes como ataques de inyección de SS y S de scripts entre sitios. Utilice bibliotecas como Yup u oi para la validación de entrada y evite ejecutar directamente datos proporcionados por el usuario en consultas o comandos.

Autenticación y autorización Implemente mecanismos robustos de autenticación y autorización para controlar el acceso a áreas o datos sensibles. Utilice bibliotecas como Passport.js o Nextuth.js para manejar los flujos de autenticación y autorización de usuarios de forma segura.

Control de acceso: Asegúrese de que solo los usuarios autorizados puedan realizar acciones privilegiadas mediante la implementación de mecanismos de control de acceso adecuados. Aplique el principio del mínimo privilegio, otorgando a los usuarios solo los permisos necesarios para sus roles o responsabilidades.

Manejo de errores Implemente un manejo de errores y un registro de errores adecuados para evitar que la información confidencial quede expuesta a los usuarios.

Muestra mensajes de error fáciles de usar mientras registra errores detallados para fines de depuración.

- **Escalabilidad e implementación**

Prepare su aplicación Next.js para una escalabilidad y una implementación eficiente

Configura el balanceo de carga para distribuir el tráfico entrante entre varios servidores.

Balanceadores de carga como AWS Elastic Load Balancer B o Nginx pueden ayudar a distribuir la carga de trabajo y garantizar una alta disponibilidad .

Escalabilidad horizontal Diseñe su aplicación para que sea escalable horizontalmente, lo que le permitirá agregar o eliminar instancias dinámicamente

Bajo demanda. Utilice tecnologías de contenedорización como Docker o plataformas de orquestación como Kubernetes para facilitar el escalado.

Implementación automatizada Implemente canalizaciones CICD de integración continua e implementación continua para automatizar el proceso de implementación.

Herramientas como enkins, ravis CI o S CodePipeline pueden ayudarlo a automatizar compilaciones, pruebas e implementaciones, lo que garantiza un ciclo de lanzamiento fluido y eficiente.

Monitoreo del Rendimiento: Utilice herramientas de monitoreo como New Relic, Datadog o Cloudwatch para monitorear el rendimiento de su aplicación Next.js en producción.

Monitoree métricas como los tiempos de respuesta, la carga del servidor y el uso de recursos para identificar cuellos de botella en el rendimiento o necesidades de escalado.

Si sigue estas prácticas de configuración, podrá optimizar el rendimiento, la seguridad y la escalabilidad de su aplicación Next.js, garantizando así su correcto funcionamiento en un entorno de producción.

Configuración de pipelines (CI/CD) para aplicaciones Next.js

La configuración de pipelines CICD de integración continua e implementación continua para aplicaciones Next.js implica automatizar el proceso de compilación, prueba e implementación de la aplicación cada vez que se realizan cambios. Esto garantiza que el código base se integre, pruebe e implemente continuamente en diversos entornos. Aquí encontrará una guía paso a paso para configurar pipelines CICD para aplicaciones Next.js, incluyendo flujos de trabajo itub para ejecutar pruebas y compilar.

1. Sistema de control de versiones

Empieza por usar un sistema de control de versiones similar a CS para gestionar el código fuente de tu aplicación Next.js. Crea un repositorio de TI para almacenar y realizar un seguimiento de los cambios en tu proyecto.

2. Elija un servicio CI/CD

Seleccione un servicio CICD que se integre con su repositorio de TI. Algunas opciones populares son itub ctions, itab CICD y ravis CI. En esta guía, nos centraremos en GitHub Actions.

. Configúrelo

Habilite las acciones de itub en su repositorio navegando a la pestaña de acciones y siguiendo las instrucciones de configuración. Esto le permitirá definir flujos de trabajo que activan automáticamente ciertas acciones en función de eventos como envíos, solicitudes de extracción o intervalos programados.

. Crear un archivo orflow

Dentro de tu repositorio, crea un directorio .gitworflows si no existe. En ese directorio, crea un archivo Y (por ejemplo, main.yml) para definir tu flujo de trabajo. Este archivo contendrá los pasos necesarios para ejecutar pruebas y compilar tu aplicación Next.js.

. Deine est orflow

En el archivo de flujo de trabajo, defina un trabajo para ejecutar pruebas. Esto generalmente incluye pasos como verificar el repositorio, configurar el entorno Node.js, instalar dependencias y ejecutar pruebas. Aquí hay un ejemplo.

```
--yaml
nombre: Prueba
en:
    empujar:
        sucursales:
            - principal

trabajos:
    prueba:
        se ejecuta en: ubuntu-latest
        pasos:
            - nombre: Repositorio de pago
              usos: acciones/checkout@v2

            - nombre: Configurar Node.js
              usos: acciones/setup-node@v2
              con:
                  versión del nodo: '14'

            - nombre: Instalar dependencias
              ejecutar: npm ci

            - nombre: Ejecutar pruebas
```

```
ejecutar: npm test
```
. De fina el flujo de aire

A continuación, defina un trabajo para construir su aplicación Next.js. Este trabajo depende de la finalización exitosa del trabajo de prueba. Aquí hay un ejemplo.

```yaml
nombre: Construir

en:
    empujar:
        sucursales:
            - principal

trabajos:
    prueba:
        se ejecuta en: ubuntu-latest

pasos:
    - nombre: Repositorio de pago
        usos: acciones/checkout@v2

    - nombre: Configurar Node.js
        usos: acciones/setup-node@v2
        con:
            versión del nodo: '14'

    - nombre: Instalar dependencias
        ejecutar: npm ci

    - nombre: Crear la aplicación Next.js

```

ejecutar: npm run build

...

7. Comprometerse y empujar

Confirme los archivos de flujo de trabajo en su repositorio y envíe los cambios. Esto activará los flujos de trabajo definidos en su canalización CICD.

es un

Inicio

Las acciones de itub ahora ejecutarán automáticamente los flujos de trabajo definidos siempre que haya nuevas confirmaciones o solicitudes de extracción dirigidas a la rama principal. El flujo de trabajo ejecutará pruebas y, si tiene éxito, el flujo de trabajo de compilación compilará su aplicación Next.js.

Siguiendo estos pasos, puedes configurar pipelines CICD para tu aplicación Next.js mediante GitHub Actions. Estos pipelines ejecutarán pruebas y compilarán tu aplicación automáticamente cada vez que se realicen cambios, lo que te ayudará a detectar problemas de forma temprana y a garantizar que tu código esté listo para su implementación.

Monitoreo y depuración de aplicaciones implementadas

La monitorización y depuración de aplicaciones implementadas en Next.js implica el uso de diversas herramientas y técnicas para identificar y resolver problemas que puedan surgir en un entorno de producción. Next.js, un framework popular para crear aplicaciones React, ofrece funciones integradas y se integra a la perfección con plataformas de implementación como Vercel y AWS. Profundicemos en los detalles de la monitorización y depuración en Next.js. ercel y S

• Supervisión de aplicaciones implementadas

La monitorización le permite supervisar el estado, el rendimiento y la disponibilidad de sus aplicaciones Next.js implementadas. Le ayuda a identificar y solucionar problemas rápidamente. Existen algunas técnicas de monitorización para Next.js.

Registro: Implemente un registro robusto en su aplicación Next.js para capturar eventos relevantes, errores e información de depuración. Puede usar bibliotecas como inston o Pino para gestionar el registro. Los mensajes og se pueden almacenar en diversas ubicaciones, como archivos locales, servidores remotos o servicios de registro de terceros como oggly o Papertrail.

Seguimiento de errores: Integre servicios de seguimiento de errores como Sentry o Rollbar en su aplicación Next.js. Estos servicios recopilan y analizan los errores que ocurren en su aplicación y proporcionan información detallada.

Información sobre la fuente, el seguimiento de la pila y el contexto de los errores.

A menudo incluyen funciones como alertas en tiempo real y agrupación de problemas para una depuración eficiente.

Monitoreo del Rendimiento: Utilice herramientas como New Relic o Datadog para monitorear el rendimiento de su aplicación Next.js. Estas herramientas pueden capturar métricas relacionadas con los tiempos de respuesta, la carga del servidor, las consultas a la base de datos y otros aspectos relacionados con el rendimiento. Proporcionan información sobre posibles cuellos de botella o áreas de optimización.

Métricas de la aplicación: Instrumente su aplicación Next.js para capturar métricas personalizadas específicas para su comportamiento. Puede usar bibliotecas como Prometheus o StatsD para recopilar y almacenar métricas relacionadas con procesos específicos de la empresa o interacciones de los usuarios. Estas métricas pueden ayudarle a identificar patrones, rastrear tendencias y tomar decisiones basadas en datos para mejorar.

- Depuración de aplicaciones implementadas

La depuración de aplicaciones Next.js implementadas implica investigar y resolver problemas que afectan la funcionalidad o el rendimiento de la aplicación. Aquí hay algunas técnicas de depuración para Next.js.

Depuración remota : utilice herramientas de depuración remota como Chrome Devtools o las capacidades de depuración remota de S Code para conectarse a su aplicación Next.js en ejecución en el entorno de producción.

Esto le permite inspeccionar variables, recorrer el código y capturar información en tiempo de ejecución para solucionar problemas.

Análisis de errores: analice los registros de errores, los seguimientos de pila y los mensajes de error para identificar las causas raíz de los problemas. Aproveche los servicios de registro y seguimiento de errores para recopilar información relevante. Preste atención a patrones de error específicos, informes de usuarios o cualquier comportamiento anómalo para identificar el origen del problema.

Pruebas de carga: Realice pruebas de carga con herramientas como Pache-Ether o Rtillery para simular escenarios de alto tráfico. Sometiendo su Next...

Si aumenta la carga de una aplicación js, puede descubrir problemas de rendimiento, limitaciones de escalabilidad o cuellos de botella de recursos que pueden afectar su estabilidad o capacidad de respuesta.

Al probar un sistema de pruebas, se emplean técnicas de prueba B y indicadores de características para implementar gradualmente nuevas funciones o cambios. Esto permite aislar y analizar el impacto de cambios específicos en la aplicación Next.js implementada. Al habilitar o deshabilitar funciones de forma selectiva, se puede identificar si una función en particular está causando problemas o degradación del rendimiento.

Monitoreo de código Utilice servicios de monitoreo de código como psagon o

rackS para rastrear y localizar problemas en el código base de tu aplicación Next.js. Estos servicios pueden proporcionar información sobre la ejecución del código y los puntos críticos de rendimiento, y ayudar a identificar posibles áreas de mejora.

- Next.js con Vercel

Vercel es una plataforma de implementación popular para aplicaciones Next.js. Simplifica el proceso de implementación y ofrece funciones adicionales de monitorización y depuración.

Ganchos de implementación: Vercel le permite configurar ganchos de implementación, que son puntos finales P que se activan después de una implementación exitosa.

Puede utilizar estos ganchos para activar scripts de supervisión personalizados o notificar a servicios externos sobre el estado de la implementación.

Variables de entorno: Vercel le permite administrar variables de entorno para su aplicación Next.js, incluidas aquellas relacionadas con la supervisión y la depuración. Esto le permite configurar diferentes ajustes para diferentes entornos e integrarse sin problemas con servicios de terceros.

Análisis: Vercel ofrece análisis integrados para supervisar el rendimiento y el uso de su aplicación Next.js. Registra métricas como el número de solicitudes, el tiempo de respuesta y el uso del ancho de banda, lo que le permite evaluar el comportamiento de la aplicación y tomar decisiones informadas.

- Next.js con AWS

AWS ofrece una gama de servicios que pueden mejorar las capacidades de monitoreo y depuración de las aplicaciones Next.js

AWS CloudWatch: CloudWatch permite recopilar y supervisar registros, configurar alarmas y visualizar métricas de rendimiento. Se pueden integrar los registros de aplicaciones de Next.js con los registros de CloudWatch y configurar métricas personalizadas para el seguimiento de comportamientos específicos de la aplicación.

AWS X-Ray: Ray ayuda a rastrear las solicitudes en su aplicación Next.js e identificar cuellos de botella o errores de rendimiento. Proporciona una representación visual del flujo de solicitudes, incluyendo datos de latencia, para ayudarle a identificar componentes problemáticos.

Capas de AWS Lambda: Si utiliza Next.js con implementaciones sin servidor que utilizan AWS Lambda, puede aprovechar las Capas de Lambda para incluir bibliotecas adicionales de depuración y monitorización en su entorno de funciones. Esto le permite capturar registros, métricas y errores específicos de sus funciones sin servidor.

AWS CloudFormation CloudFormation le permite definir

Su infraestructura de Next.js como código. Puede usar plantillas de CloudFormation para configurar recursos de monitorización, como alarmas, destinos de registro o métricas personalizadas, como parte del proceso de aprovisionamiento de su infraestructura.

En resumen, la monitorización y depuración de aplicaciones Next.js implementadas implica implementar técnicas de registro, seguimiento de errores, monitorización del rendimiento y depuración. Plataformas de implementación como Ercel y proveedores de nube como S ofrecen funciones y servicios adicionales para mejorar las capacidades de monitorización y depuración. Al aprovechar estas herramientas eficazmente, puede garantizar la estabilidad, el rendimiento y la fiabilidad de sus aplicaciones Next.js en entornos de producción.

Conclusión

Este último capítulo del libro ofrece una exploración exhaustiva de la arquitectura de implementación en aplicaciones Next.js. Como culminación de nuestro recorrido, profundizamos en las complejidades de implementar una aplicación Next.js de forma eficaz, garantizando que llegue a su público objetivo de forma segura y eficiente.

Comenzamos descifrando el proceso de implementación en Next.js, comprendiendo los pasos necesarios para llevar una aplicación del desarrollo al entorno de producción. Este conocimiento fundamental sentó las bases para explorar las diversas opciones y consideraciones de implementación.

A lo largo de este capítulo, exploramos diferentes plataformas de alojamiento como Vercel, AWS y Heroku, entre otras. Descubrimos las ventajas y características de cada plataforma y aprendimos a implementar nuestras aplicaciones Next.js sin problemas en estos proveedores de nube. Se nos guió a través de las configuraciones y los pasos necesarios para garantizar una implementación sin problemas.

Además, enfatizamos la importancia de las variables de entorno diseñadas específicamente para la implementación, entendiendo su papel en la gestión de información confidencial y cómo configurarlas de manera efectiva para nuestras aplicaciones Next.js.

Además, exploramos estrategias para optimizar el proceso de implementación, centrándonos en técnicas que mejoran el rendimiento y reducen el tiempo de inactividad. Al implementar estas prácticas recomendadas, podemos garantizar una experiencia de usuario fluida y minimizar las interrupciones durante la fase de implementación.

En la era de la integración continua y la implementación continua (CICD), también analizamos la configuración de canalizaciones diseñadas específicamente para aplicaciones Next.js.

Al aprovechar las herramientas y los marcos CICD, podemos automatizar el proceso de implementación, lo que permite implementaciones más rápidas y eficientes y, al mismo tiempo, mantener la calidad y la estabilidad del código.

Por último, profundizamos en las técnicas y herramientas de monitoreo y depuración, lo que nos permitió monitorear de manera proactiva el rendimiento de nuestras aplicaciones Next.js implementadas y abordar cualquier posible cuello de botella o error de manera efectiva.

En resumen, este capítulo final nos proporcionó una comprensión integral de las opciones de implementación disponibles para las aplicaciones Next.js. Al incorporar los conocimientos y las mejores prácticas compartidas, podemos tomar decisiones informadas sobre la mejor estrategia de implementación para nuestros casos de uso específicos. Permítanos aprovechar este conocimiento y asegurar que nuestras aplicaciones Next.js alcancen su máximo potencial en el entorno real.

Preguntas de opción múltiple

1. ¿Cuál es el enfoque de este capítulo?
 - A. Gestión de bases de datos en aplicaciones Next.js
 - B. Optimización del rendimiento en aplicaciones Next.js
 - C. Arquitectura de implementación en aplicaciones Next.js
 - D. Diseño de interfaz de usuario en aplicaciones Next.js
2. ¿Qué plataformas de alojamiento se mencionan en este capítulo?
 - A. Firebase y ongoDB tlas
 - B. Netlify y Azure
 - C. Vercel, AWS y Heroku
 - D. Páginas de itub y Digitalcean
3. ¿Cuál es la importancia de las variables de entorno en la implementación?
 - A. Mejoran la apariencia visual de la aplicación.
 - B. Mejoran el rendimiento de la aplicación.
 - C. Gestionan información sensible para la aplicación.
 - D. Automatizan el proceso de implementación.

4. ¿Cuál es uno de los temas tratados en este capítulo?
- A. Implementación de estructuras de datos avanzadas en aplicaciones Next.js
 - B. Diseño de formularios fáciles de usar en aplicaciones Next.js
 - C. Configuración de pipelines CICD para aplicaciones Next.js
 - D. Exploración de algoritmos de aprendizaje automático en aplicaciones Next.js
5. ¿Por qué es importante la monitorización y la depuración para las aplicaciones implementadas?
- A. Para aumentar la seguridad de la aplicación.
 - B. Para optimizar el proceso de implementación
 - C. o garantizar la calidad y estabilidad del código
 - D. Para mejorar la interfaz de usuario de la aplicación.

Respuestas

1 C	
2 C	
3 C	
4 C	
5 °C	

Índice

A

listas de control de acceso (ACL) 285
api asincrónicamente 124
Manejo de
 errores de llamadas API 211-214
 Manejo de excepciones 211-214
Protocolos y arquitecturas API sobre 185-186

 Llamada a procedimiento remoto de Google (gRPC) 185
 GraphQL 186
 Transferencia del Estado Representacional (DESCANSO) 185
 Protocolo simple de acceso a objetos (SOAP) 185
Rutas API
 sobre 70, 77
 protegiendo, con autenticación 271-274
Seguridad de la API

autenticación, en la aplicación Next.js 214, 216
mejores prácticas 214, 216 Cliente Apollo acerca de 214
ApolloProvider, en Next.js 201 caché 201, 204 datos, consulta 184, 185 mutación 197, 198 configuración, en Next.js 183 usado, para integrar la API GraphQL en la aplicación Next.js 190-193 Servidor Apollo usado para configurar la API GraphQL en Next.js 184 usado, para configurar la API GraphQL en Next.js 174-182 Interfaces de programación de aplicaciones (API) acerca de 183

- en el desarrollo web moderno 184
- tipos 184
- trabajando 205
- Enrutador de aplicaciones
 - alrededor de 94
- Convenciones de
 - archivos, carpetas y
 - archivos, funciones de flecha 47
 - programación asíncrona
- Código asíncrono, que se integra con el componente React 63
- Código React asíncrono, mejores prácticas 59, 60
- Código React asíncrono, manejo de errores 64
- Sintaxis asíncrona para un código asíncrono más limpio 63
- Sintaxis de espera para un código asíncrono más limpio 63
- funciones de devolución de llamada 61
- limitaciones de devolución de llamada 56, 57
- en JavaScript 61
- promesas y encadenamiento 62
- autenticación
 - alrededor de 283
 - autenticación multifactor (MFA) 285
- inicio de sesión de usuario 285
- registro de usuario 285
- autenticación y seguridad
 - registros de auditoría, manteniendo 306
 - autorización y control de acceso, implementando 306
 - Mejores prácticas en la aplicación Next.js 274
 - Ataque de secuencias de comandos entre sitios (XSS) 275
 - Ataque CSRF, evitando 275
 - Dependencias y parches, actualización 275
- limitación de velocidad, implementación 275
- auditorías de seguridad y pruebas, realización 276
- encabezado de seguridad, implementación 275
- Mecanismos de autenticación fuerte 274
- Seguridad de la capa de transporte (TLS), implementación 275
- almacenamiento de credenciales de usuario 275
- entrada del usuario, desinfección 275
- entrada del usuario, validando 275
- educación de los usuarios 276
- proveedores de autenticación
 - alrededor de 263
- proveedores personalizados 269-271
- autenticación de correo electrónico/contraseña 267, 268
- Implementando 266
- Autenticación JWT 268
- Proveedor de OAuth 266, 267
- autorización
 - alrededor de 256
- listas de control de acceso (ACL) 285
- control de acceso basado en roles (RBAC) 256
- enrutamiento automático 64, 73
- Servicios de AWS
 - Formación de nubes 337
 - CloudWatch 337
 - Capas lambda 337
 - Rayos X 337
- B
 - Forma normal de Boyce-Codd (FNBC) 204
 - tamaño del paquete
 - analizando 115
 - reduciendo 115
 - tamaño del paquete, técnicas
 - división de código 115
 - Integración y despliegue
 - continuos (CI/CD) 116

378 aplicaciones web modernas con Next.JS

- Dependencias, optimización 115
- Estrategias de implementación y mejores prácticas 115
- Minificación 115
- Monitoreo del rendimiento 116
- Escalabilidad y equilibrio de carga 116
- Consideraciones de seguridad 116
- Implementaciones sin servidor 116
- Tree Shaking 115
- do
 - caché 331
 - devolución
 - de llamada
 - acerca de 263 opciones de configuración
 - 263 jwt 263
 - redirección 263
 - sesión 263
 - inicio de sesión 263 Hojas de estilo en cascada (CSS) 3, 4 configuración
 - de canalizaciones CI/CD para la aplicación Next.js
 - 333-335 componentes de clase
 - 31, 32 componente de cliente
 - 88 lado del
 - cliente utilizado para integrar puntos finales de API en Next.js 182
 - caché del lado del cliente 114
 - optimización del lado del cliente 331
 - representación del lado del cliente (CSR) acerca de 18-20, 108, 236 ventajas 108
 - beneficios 238
 - mejores prácticas en la aplicación Next.js 250, 251
 - representación de componentes 244, 245
 - inconvenientes 239
 - obtención dinámica de datos 245, 246
 - interactividad y actualización del lado del cliente 246, 247
- versus renderizado del lado del servidor (SSR) 236-238
- con Next.js 240, 244
- minimización de código 331
- división de código 112
- arquitectura basada en componentes
 - acerca de 30, 31
 - composición 30
 - principio de responsabilidad única 30
 - composición 30 enlace
 - de constructor 43 implementación de aplicación CRUD 313
 - operaciones CRUD
 - con base de datos seleccionada 211-215
 - Módulo CSS
 - alrededor de 78
 - composición y clases globales 80, 81 creando 79
 - usando 79, 80
 - hojas de estilos
 - CSS que sirven 100, 101 implementación
 - de estilos CSS
 - en Next.js con módulos CSS 78
- proveedores personalizados 269-271
- D
 - configuración
 - de la base de datos, con Supabase 293-297
 - configuración de la conexión
 - a la base de datos, en Next.js 207-211
 - manejo de errores de base de datos 216
 - manejo, en Next.js con TypeORM 217 descripción general del sistema de gestión de bases de datos (DBMS) 201, 203
 - normalización de la base de datos Forma normal de Boyce-Codd (BCNF) 204

- 
- Primera forma normal (1NF) 204
 - Cuarta Forma Normal (4NF) 204
 - Segunda forma normal (2NF) 204
 - Tercera forma normal (3NF) 204 técnicas de rendimiento de bases de datos 230 escalamiento de bases de datos acerca de 230 para alta disponibilidad 230 para rendimiento 230 escalamiento horizontal 230 monitoreo y optimización 231 escalamiento vertical 230 auditoría y registro de seguridad de bases de datos 221 mejores prácticas en Next.js 220 acceso a bases de datos, limitación 222 credenciales de bases de datos, protección 221 protección de datos 221 recuperación ante desastres 222 cifrado 221 validación de entrada 221 pruebas de penetración 222 parametrización de consultas 221 copias de seguridad regulares 222 evaluaciones de seguridad regulares 222 actualizaciones y parches regulares 221 implementación segura 222 alojamiento seguro 222 autenticación de usuario, implementación 221 autorización de usuarios, implementación de la centralización de datos 221 203 consistencia de datos 203 obtención de datos aproximadamente 92 función asíncrona, uso en el componente de servidor 92 función de espera, uso en el componente de servidor 92 almacenamiento en caché 114
 - obtención dinámica de datos 93 mejora 114 obtención estática de datos 92 independencia de datos 203 integridad de datos 203 mantenimiento de datos 203
 - Modelado de datos y diseño de esquemas sobre 222 columna 223 entidad 222 migración 223 mapeo relacional de objetos (ORM) 224 clave principal 223 consulta 223 relación 223 transacciones 223
 - con TypeORM 224-230 consulta de datos 203 recuperación de datos 203 informes de datos 203 escalabilidad de datos 203 seguridad de datos 203 intercambio de datos 203
 - Beneficios del DMS centralización de datos 203 consistencia de los datos 203 independencia de los datos 203 integridad de los datos 203 mantenimiento de los datos 203
 - Consulta de datos 203 Recuperación de datos 203 Informes de datos 203 Escalabilidad de datos 203 Seguridad de datos 203 Intercambio de datos 203 Rendimiento mejorado 203
 - DBMS, componentes clave, copia de seguridad y recuperación 202, control de concurrencia 202, datos 201 base de datos 201

380 aplicaciones web modernas con Next.JS

- 
- esquema de base de datos 201
 - lenguaje de definición de datos DDL 202 lenguaje de manipulación de datos (DML) 201
 - optimización de consultas 202
 - seguridad y autorización 202 gestión de transacciones 202
 - DBMS, tipos
 - DBMS jerárquico 202
 - Sistema de gestión de bases de datos de red 202
 - Sistema de gestión de bases de datos NewSQL 202
 - Sistema de gestión de bases de datos NoSQL 202
 - Sistema de gestión de bases de datos orientado a objetos (SGBOO) 202
 - Gestión de bases de datos relacionales
 - Técnicas de depuración del sistema (RDBMS) 202 sobre 216
 - Pruebas A/B 336 revisión de código 217
 - herramientas de depuración 216 análisis de errores 336 indicadores de características 336 manejo, en Next.js con TypeORM 218-220
 - pruebas de carga 336
 - registro 216 optimización de consultas 216 depuración remota 336 pruebas unitarias 217 tipos de entorno de implementación alrededor de 316 servidor personalizado 317, 318 plataforma de implementación 316 entorno de desarrollo 316 entorno de producción 316 función sin servidor 317 entorno de ensayo 316 alojamiento estático 317 entorno de prueba 316 optimización del proceso de implementación 329, 330
 - Estrategias de implementación y mejores prácticas
 - alrededor de 116-118 integración e implementación continuas (CI/CD) 116, 117 monitoreo del rendimiento 117 escalabilidad y balanceo de carga 117 consideración de seguridad 117 implementación sin servidor 116 representación dinámica del lado del cliente 247-250 obtención dinámica de datos 93 importaciones dinámicas 112, 113 representación dinámica 91 definición de ruta dinámica 132 acceso a parámetro dinámico 133 vinculación 132 trabajar en Next.js 132 rutas dinámicas 73, 90, 91
 - mi
 - Variables de entorno para el método de archivo de implementación en el entorno Linux 320
 - Método de archivo en el entorno de Windows 320
 - Configuración de acciones de flujo de trabajo de Gitub 321-323
 - Entorno Linux 320 Entorno local 318
 - Entorno de producción 319
 - Configuración 318 Entorno de prueba 319
 - Entorno de Windows 320 error excepción en llamadas API 193-195
 - manejo de errores acerca de 91 conexión de base de datos 216 límites de error 216 mensajes de error 216 en llamadas API 193-195

ejecución de consultas 216

páginas de error 74

agrupación de eventos 42

almacenamiento en caché externo 115

F

convenciones de
archivo

acerca de 87

error.js 88

layout.js 88

loading.js 88

page.js 87 Primera Forma Normal
(1NF)

204 Flux

acerca de 146

componentes 146

conceptos 146-153 Cuarta Forma

Normal (4NF) 204 componentes funcionales 31

GRAMO

Función getServerSideProps 241 Función

getStaticPaths 243, 244 Función

getStaticProps 242

Llamada a procedimiento remoto de Google (gRPC)
169

GraphQL

aproximadamente 169

mutación 174

consulta

174 resolver

174 Configuración

de API GraphQL en Next.js

con Apollo Server 174-182

integración con Apollo Client

en la aplicación Next.js 190-193

Configuración en Next.js con Apollo

Servidor 174

versus API RESTful 170

H

componente principal

utilizado para agregar metadatos a las páginas
97-99

DBMS jerárquico 202 técnicas

de alta disponibilidad

231 escala horizontal

230 HTML versus JavaScript

XML

(JSX) 40-42 renderizado híbrido 20-22 enfoque

de renderizado híbrido con

Next.js 240 Lenguaje de marcado

de hipertexto (HTML)

2, 3

I

Optimización de imágenes

331

imágenes

sirviendo 100 rendimiento mejorado 203

Regeneración estática incremental (ISR) 64

funciones de flecha en línea 44

Yo

JavaScript

sobre 4, 5

programación asíncrona 56

Fundamentos de JavaScript para
Next.js

acerca de 9 funciones

de flecha 15 async/

await 16

clases 13 declaraciones de flujo de

control 10, 11

destrucción 15

funciones 12

módulos 13, 14 promesas 14

382 aplicaciones web modernas con Next.js

- operador de propagación 16
 - literales de plantilla 17
 - variables y tipos de datos 9, 10 archivos
 - avaScript que sirven
 - 101, 102
 - JavaScript XML (JSX) acerca de
 - 37
 - representación condicional, manejo 39,
 - 40
 - Expresiones de JavaScript, incrustación 39 bucles 39, 40
 - reglas y mejores
 - prácticas 38, 39 frente a HTML 40-42
 - Autenticación JWT 268
 - Yo
 - diseño propiedad 104 faro 118
 - carga propiedad 106
 - METRO
 - Adición de
 - metadatos a páginas con componentes de encabezado
 - middleware nent
 - 97-99 91
 - MobX 138
 - técnicas de monitoreo métricas de
 - aplicación 336 seguimiento de errores
 - 336 registro 336 monitoreo
 - de rendimiento
 - 336
 - norte
 - rutas anidadas 73
 - Sistema de gestión de bases de datos de red 202
 - Sistema de gestión de bases de datos NewSQL 202
 - Siguiente autorización
 - Características principales
 - 258 Descripción general
 - 257-260 Configuración en la aplicación Next.js
 - 260-266
- Representación del lado del cliente (CSR) de Next.js 240,
- 244 conexión de base de datos, configuración 207-
- 211
- seguridad de bases de datos, mejores prácticas 220
- definir 5 rutas
- dinámicas, trabajando 132 características 5,
- 6 renderizado 235
- renderizado del lado
- del servidor (SSR) 240, 241 servicio de archivos
- estáticos implementando 99 casos de uso 6, 7 Next.js
- 13 aplicación acerca
- de 87
- configuración 85
- etapas 85, 86
- autenticación y seguridad de
- la aplicación Next.js, mejores prácticas 274
- mejores prácticas 144, 145
- representación del lado del cliente
- (CSR), mejores prácticas 250, 251
- supervisión de código 337 creación
- 25 depuración 335, 336
- implementación,
- en AWS 327-329
- implementación, en Heroku 325, 326
- implementación, en plataformas de alojamiento
- 323 implementación, en Vercel 323, 324, 325 aplicación
- Hello World, creación 26,
- 27
- monitoreo 335, 336 eventos de
- navegación, manejo 129 navegación, realización
- 129
- Siguiente Auth, configuración de páginas
- 260-266, navegación con el proyecto del enrutador
- 128, configuración de la estructura
- del proyecto 25, 26
- Punto final de API REST, integración con el lado
- del cliente 182-190
- objeto enrutador, accediendo a 129

- Representación del lado del servidor (SSR), mejores prácticas 250, 251 gestión de estado, implementación con React contexto 155-161
- gestión de estado, implementación con Redux 145 utilizado, para configurar canalizaciones de CI/CD 333-335
- Gancho useRouter, importando 128 con AWS 337, 338 con Vercel 337
- Aplicación Next.js para producción que configura 331 configuraciones específicas del entorno 331 optimización del rendimiento 331 escalabilidad e implementación 332 medidas de seguridad 331, 332
- beneficios de las aplicaciones Next.js 97 rendimiento, monitorización 118 rendimiento, optimización 118, 119
- arquitectura de Next.js acerca de 109, 110 representación del lado del cliente (CSR) 108 representación del lado del servidor (SSR) 108 funcionamiento 108 proceso de implementación de Next.js acerca de 315 tipos de entorno de implementación 316 entorno de desarrollo de Next.js dependencias adicionales, instalación 25 app/global.css 282 app/layout.css 282 app/page.css 284 app/page.module.css 283 servidor de desarrollo, implementación 24 proyecto Next.js, creación 23 Node.js, instalación 22, 23 npm, instalación 22, 23 requisitos previos 280, 281 configuración 22, 280
- Next.js para un rendimiento óptimo configurando 110
- Next.js para el desarrollo de aplicaciones web alrededor de 7
- División y optimización automática de código 7
- Renderizado del lado del servidor integrado 7 Soporte integrado para React 8 Extensible y personalizable 8
- Opción de obtención de datos flexible 8 Experiencia de desarrollador mejorada 8 Comunidad grande y activa 8
- Configuración e implementación 7 Generación de sitios estáticos 7
- Compatibilidad con TypeScript 8
- Marco Next.js Acerca de 63, 64 Ventajas 64, 65 Marcos de JavaScript 65 Casos de uso 65-67
- Componente de imagen Next.js para optimización de imágenes 102, 103 propiedad de diseño 104 propiedad de carga 106 propiedad objectFit 104 propiedad objectPosition 105 propiedad de prioridad 106, 107
- Componente de enlace Next.js acerca de 125 uso 125-128
- Página Next.js creando 74, 75 componente de diseño, creando 76-78 vinculando 75 directorio de páginas 72 recapitulación 76 renderizado 76 visualización 75 Proyecto Next.js creando 67, 68 instalación 67, 68

Prerrequisitos 67
 Configuración 68,
 69 Estructura de carpetas del proyecto
 Next.js Acerca de 69
 descripción general
 69, 70 archivo package.json
 72 directorio de páginas, explorando 70
 directorio público, explorando 71 directorio
 de estilos, explorando 71, 72 Rol de enrutador
 Next.js 122-125

Gestión de estados de Next.js sobre
 137, 138 MobX 138
 opciones 138-140
 Contexto de reacción 138
 Retroceso 139
 Redux 138
 Estado actual 139

Enlace
 de referencia de Node.js
 22 normalización 204

DBMS NoSQL
 acerca de 202, 205
 modelo orientado a documentos 205
 consultas e indexación 205, 206 replicación y
 alta disponibilidad 205 escalabilidad y alto rendimiento
 205 flexibilidad de esquema 205
 casos de uso y contexto 206
 versus SQL DBMS 206, 207

Oh

propiedad objectFit 104
 Programación Orientada a Objetos (POO) 17, 18

propiedad objectPosition 105
 Procesamiento analítico en línea (OLAP) 204
 Procesamiento de transacciones en línea (OLTP) 204,
 205

PAG
 páginas
 protectoras, con autenticación 271-
 Directorio de 274
 páginas sobre 72
 Rutas API 73
 enrutamiento automático 73
 rutas dinámicas 73 páginas
 de error 74 rutas
 anidadas 73
 Método PATCH 173
 propiedad de prioridad 106, 107
 Método PUT 173

R

Reaccionar
 alrededor de 29
 componentes de clase 31
 arquitectura basada en componentes 30, 31
 componentes funcionales 31 DOM
 virtual 29, 30

Manejo de errores de
 componentes de React 35-37
 métodos de ciclo de vida, descripción
 general 33 métodos de fase, montaje 33, 34
 métodos de fase, desmontaje 35 métodos de
 fase, actualización 34 uso 33

Contexto de reacción
 alrededor de 138
 utilizado para implementar la gestión estatal
 miento
 En la aplicación Next.js 155-161, funciones
 de devolución de llamada
 de paso de datos de React, uso
 para la comunicación padre-hijo 45

Estado compartido, gestión del estado
 45-47, elevación del 45-47 a
 través de los apoyos 44

- Manejo de eventos de React
 - sobre 42
 - mejores prácticas 42
 - enlace a componentes 43 agrupación de eventos 42
 - Eventos sintéticos 42
- React Hooks
 - acerca de 51 incorporado 56 Hooks personalizados, implementando para lógica reutilizable 53 uso 51 useContext Hooks, para administración avanzada de estado 54 useEffect, dependencias 52 useEffect, trabajando con 52 useReducer Hooks, para administración avanzada de estado 54 useState Hook, usando 51
- Componente de
 - propiedades de React comunicación 48 validación, con PropTypes 49 trabajando con 47
- Estado de reacción
 - Mejores prácticas 48
 - Componentes 47
 - Gestión del flujo de datos con Context API 49-51 inmutabilidad, manejo 48 inicialización 47 gestión, con Context API 49-51 actualización 47 utilizado, para implementar la gestión de estado 141 trabajando con 47
 - monitorización de usuarios reales (RUM) 118
 - Retroceso 139
 - proceso de reconciliación 29
- Redux
 - alrededor de 138
 - conceptos 146-153
 - principios 145
- utilizado para implementar la gestión estatal
mento en la aplicación Next.js 145
- Redux Thunk
 - sobre la implementación 153, con ejemplo de código 153-155 necesidad de 153
- Gestión de bases de datos relacionales
 - Sistema (RDBMS) 202, 204
- relaciones tipos entidad
 - de muchos a muchos 223 entidad de muchos a uno 223 entidad de uno a muchos 223 entidad de uno a uno 223 representación acerca de 91
- renderizado dinámico 91
- renderizado estático 91
- Transferencia de Estado Representacional (REST)
 - 169
- Integración de puntos finales de API REST con el lado del cliente en la aplicación Next.js 182, 187-190
- Configuración
 - de API RESTful con Next.js 171, 172 Método DELETE 173, 174 Método GET 171
- Método POST 172
- Método PUT 172
- configuración, en Next.js 171, 172 versus GraphQL API 170 control de acceso basado en roles (RBAC) 256 enrutador
 - usado para navegar páginas En la aplicación Next.js, el enrutamiento es de aproximadamente 128.
- rutas dinámicas 90, 91 middleware 91

S

Segunda forma normal (2NF) 204
 seguridad
 acerca de
 256 protección de datos
 256 validación y saneamiento de entrada 256
 actualizaciones de seguridad regulares
 257 comunicación segura 256
 encabezados de seguridad
 257 pruebas
 257 componente de servidor
 88-90 almacenamiento
 en caché
 del lado del servidor acerca
 de 114 implementación 111, 112
 representación del lado del
 servidor (SSR)
 acerca de 5, 18,
 19, 108, 236 ventajas 108 beneficios 239
 mejores
 prácticas, en la
 aplicación Next.js 250, 251 inconvenientes
 239 frente
 a la representación del lado
 del cliente (CSR) 236-238 con Next.js 240,
 241 representación del lado del
 servidor (SSR) 64, 331 configuración de sesión
 263
 Protocolo simple de acceso a objetos
 (SOAP) 169
 principio de responsabilidad única 30
 DBMS SQL frente a DBMS NoSQL 206, 207 Gestión
 del estado del gancho 182 de obsoleto durante la revalidación (SWR)
 Estudio de caso 162-164
 Implementando, con React contexto en
 Next.js aplicación 155-161
 Implementando, con React estado 141
 Implementando, con Redux
 En la opción de gestión de estado
 de la aplicación Next.js 145
 contras 140, 141

Pros 140, 141

obtención de datos estáticos
 92 servicio de archivos
 estáticos hojas de estilos CSS, servicio
 100, 101 imágenes,
 servicio 100 implementación, en
 Next.js 99 servicio de archivos avaScript
 101, 102 representación
 estática 91 generación de sitios estáticos (SSG)
 5, 64, 236
 Componentes Supabase/ToDoItem.jsx
 297-299 usado, para configurar la base de datos
 293-297 Eventos sintéticos 42

T

Tercera forma normal (3NF) 204
 Elementos de tarea (Crear)
 añadir 300 crud/
 page.js 300-304 crud/
 page.module.css 301, 302 Elementos de
 tarea (Eliminar)
 componentes/ToDoItem.jsx 311, 312 crud/
 page.js 308-310 eliminar 308
 Elementos de
 tarea (Leer) componentes/
 ToDoItem.module.css 290 crud/page.js 291,
 292
 mostrar 285 archivo page.js
 287, 289
 page.module.css 286
 Elementos de tarea
 (Actualizar) componentes/
 ToDoItem.jsx 307, 308 crud/page.js 305, 306
 editar 305

Tú

useCallback 56
 useContext 142-144
 useLayoutEffect 56
 useMemo 56

useRef 56
Gancho useRouter
 propiedades 129-131
useState 141, 142
función useSWR()
 acerca de 182
 datos 183
 error 183
 captador 182
 estáValidando 183
 clave 182
 mutar 183
 opciones 182, 183
 revalidate() 183
 shouldRevalidate 183

V

Vercel
 acerca de
 337 análisis 337
ganchos de implementación
337 variables de entorno 337

escalado vertical 230
DOM virtual
 alrededor de 29,
 30 beneficios 29

O

Aplicación web
 sobre 1, 2
bloques de construcción 2
Hojas de estilo en cascada (CSS) 3, 4
Lenguaje de marcado de hipertexto (HTML) 2, 3

JavaScript 4, 5
versus sitio web 2

sitio web
 acerca de
 2 versus aplicación web 2

Z

Estado actual 139

Modern Web Applications with Next.js

This practical handbook takes you on a journey from foundational principles to advanced techniques, offering a complete exploration of Next.js, the cutting-edge framework for building performant and dynamic web applications.

Beginning with an introductory overview of web applications utilizing Next.js and JavaScript, the book reintroduces React to ensure a strong footing in the core concepts. It then delves into the fundamentals of Next.js, providing insights into the latest version's core advancements and optimizations.

It will help you explore the intricacies of Next.js applications, including an in-depth look at optimizing performance. It will then move on to demystify routing in Next.js, mastering state management, and implementing RESTful and GraphQL APIs. By the end of it, you will understand the usage of diverse databases and discover the significance of client-side and server-side rendering in Next.js applications.

This book also covers crucial aspects of securing applications using NextAuth. It will help you learn to develop a complete CRUD application, gaining hands-on experience and insight into deployment architectures that can turn your projects into scalable and production-ready applications.

WHAT YOU WILL LEARN

- Gain a comprehensive understanding of web applications utilizing the latest version of Next.js and JavaScript, and refresh yourself with React's core concepts.
- Learn how to optimize Next.js applications, by improving their speed and efficiency for better user experiences.
- Understand the intricate mechanism of routing in Next.js to create dynamic web applications.
- Implement advanced State Management techniques within your Next.js applications for efficient data handling.
- Learn the implementation of both RESTful and GraphQL APIs by their integration into Next.js applications.
- Explore the usage of various databases and understand how to employ them effectively within Next.js applications.

WHO IS THIS BOOK FOR?

This book caters to the needs of developers operating at an intermediate to advanced level in web development and software engineering. Proficiency in JavaScript and a solid grasp of React fundamentals are recommended prerequisites for an optimal learning experience. Those with prior exposure to web development concepts and tools will find this book a valuable resource, augmenting their understanding and practical application of the content within.



www.orangeava.com

ORANGE EDUCATION PVT LTD

ISBN: 978-93-88590-97-6



9789388590976

(For sale in the Indian Subcontinent only)