Next JS Questions

1. Q: What is Next.js and why would you use it?
   A: Next.js is a React framework for building server-side rendered (SSR) and static websites with built-in routing and other useful features. It offers benefits like improved SEO, performance optimization, and easy setup.
2. Q: How do you create a new Next.js project?
   A: To create a new Next.js project, you can use the `create-next-app` command as follows:

```
npx create-next-app my-next-app
```

1. Q: How do you handle CSS in Next.js?
   A: Next.js allows you to import CSS files directly into your components. You can create a `styles` directory and import the CSS file within your components.

```
// styles/Home.module.css
.title {
  color: blue;
}
```

```
// pages/index.js
import styles from '../styles/Home.module.css';

function Home() {
  return <h1 className={styles.title}>Hello Next.js</h1>;
}

export default Home;
```

1. Q: How can you pass data from a parent component to a child component in Next.js?
   A: You can pass data from a parent component to a child component using props.

```
// ParentComponent.js
function ParentComponent() {
  const data = 'Hello from parent';
  return <ChildComponent data={data} />;
```

```
}

// ChildComponent.js
function ChildComponent({ data }) {
  return <p>{data}</p>;
}
```

1. Q: What is the purpose of the `_app.js` file in Next.js?
   A: The `_app.js` file is used to customize the root component of the Next.js app and apply global styles or layouts. It is a great place to include components that should persist across all pages.

```
// _app.js
import '../styles/globals.css'

function MyApp({ Component, pageProps }) {
  return <Component {...pageProps} />
}

export default MyApp
```

1. Q: How do you handle dynamic routes in Next.js?
   A: Dynamic routes can be handled by creating files with square brackets in the `pages` directory. For example, to handle `/products/123`, create a file named `[id].js`.

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

function Product() {
  const router = useRouter();
  const { id } = router.query;

  return <h1>Product ID: {id}</h1>;
}

export default Product;
```

1. Q: How can you add metadata to the `head` of a Next.js page?
   A: You can use the `next/head` component and include meta tags inside it to add metadata to the `head` of a page.

```
// pages/index.js
import Head from 'next/head';
```

```
function Home() {
  return (
    <div>
      <Head>
        <title>My Next.js App</title>
        <meta name="description" content="This is a Next.js app" />
      </Head>
      <h1>Hello Next.js</h1>
    </div>
  );
}

export default Home;
```

1. Q: How can you create a link with a custom route in Next.js?
   A: You can use the `Link` component from `next/link` and pass the custom route as the `href` prop.

```
// pages/index.js
import Link from 'next/link';

function Home() {
  return (
    <div>
      <h1>Hello Next.js</h1>
      <Link href="/about">
        <a>About</a>
      </Link>
    </div>
  );
}

export default Home;
```

1. Q: How do you handle form submissions in Next.js?
   A: Form submissions can be handled using the `onSubmit` event in a form component. For example:

```
// pages/contact.js
import { useState } from 'react';

function Contact() {
  const [message, setMessage] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    // Submit logic here
    console.log('Submitted message:', message);
  };
```

```
  return (
    <form onSubmit={handleSubmit}>
      <textarea value={message} onChange={(e) => setMessage(e.target.value)} />
      <button type="submit">Submit</button>
    </form>
  );
}

export default Contact;
```

1. Q: How can you implement client-side routing in Next.js?
   A: Next.js automatically handles client-side routing using the `Link` component from `next/link`.

```
// pages/index.js
import Link from 'next/link';

function Home() {
  return (
    <div>
      <h1>Hello Next.js</h1>
      <Link href="/about">
        <a>About</a>
      </Link>
    </div>
  );
}

export default Home;
```

1. Q: Explain the use of the `getStaticProps` function in Next.js and how it helps with performance optimization.
   A: `getStaticProps` is used for static site generation (SSG) in Next.js. It fetches data at build time and generates static HTML pages. This helps optimize performance as the content is pre-rendered and served from a CDN.

```
// pages/posts/[id].js
export async function getStaticProps({ params }) {
  const res = await fetch(`https://api.example.com/posts/${params.id}`);
  const post = await res.json();

  return {
    props: {
      post,
    },
    revalidate: 60, // Revalidate every 60 seconds for incremental static regeneration
  };
```

```
}

function Post({ post }) {
  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.body}</p>
    </div>
  );
}

export default Post;
```

1. Q: How do you handle data fetching in Next.js for both server-side and client-side rendering?
   A: Next.js provides different data fetching functions for server-side rendering (`getServerSideProps`) and client-side rendering (`useEffect` or `fetch`). You can choose the appropriate method based on your requirements.

```
// pages/posts/[id].js
import { useEffect, useState } from 'react';

export async function getServerSideProps({ params }) {
  const res = await fetch(`https://api.example.com/posts/${params.id}`);
  const post = await res.json();

  return {
    props: {
      post,
    },
  };
}

function Post({ post }) {
  const [comments, setComments] = useState([]);

  useEffect(() => {
    fetch(`https://api.example.com/posts/${post.id}/comments`)
      .then((response) => response.json())
      .then((data) => setComments(data));
  }, [post.id]);

  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.body}</p>
      <ul>
        {comments.map((comment) => (
          <li key={comment.id}>{comment.body}</li>
        ))}
      </ul>
    </div>
  );
```

```
}

export default Post;
```

1. Q: How can you implement client-side routing with parameters in Next.js?
   A: You can use the `useRouter` hook from `next/router` to access the route parameters and perform client-side navigation.

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

function Product() {
  const router = useRouter();
  const { id } = router.query;

  return (
    <div>
      <h1>Product ID: {id}</h1>
      {/* Other product details */}
    </div>
  );
}

export default Product;
```

1. Q: How do you implement dynamic meta tags (SEO) in Next.js for each page?
   A: You can use the `next/head` component along with the data fetched in `getServerSideProps` or `getStaticProps` to set dynamic meta tags.

```
// pages/posts/[id].js
import Head from 'next/head';

export async function getServerSideProps({ params }) {
  const res = await fetch(`https://api.example.com/posts/${params.id}`);
  const post = await res.json();

  return {
    props: {
      post,
    },
  };
}

function Post({ post }) {
  return (
    <div>
      <Head>
        <title>{post.title}</title>
        <meta name="description" content={post.excerpt} />
      </Head>
```

```
      <h1>{post.title}</h1>
      <p>{post.body}</p>
    </div>
  );
}

export default Post;
```

1. Q: How can you optimize images in Next.js to improve performance?
   A: Next.js provides built-in image optimization with the `next/image` component. You can use the `layout` prop to optimize the images for performance and responsive design.

```
// pages/index.js
import Image from 'next/image';

function Home() {
  return (
    <div>
      <h1>Hello Next.js</h1>
      <Image
        src="/image.jpg"
        alt="Image"
        width={500}
        height={300}
        layout="responsive"
      />
    </div>
  );
}

export default Home;
```

1. Q: How do you handle API routes in Next.js for serverless functions?
   A: You can create API routes in the `pages/api` directory. These routes automatically become serverless functions that can be used to handle API requests.

```
// pages/api/products.js
export default function handler(req, res) {
  const products = fetchProductsFromDatabase();
  res.status(200).json(products);
}
```

1. Q: How can you handle user authentication and protected routes in Next.js?
   A: User authentication can be handled using external providers like Auth0 or Firebase, or by creating custom authentication logic using serverless functions.

```
// pages/api/login.js
export default function handler(req, res) {
  if (req.method === 'POST') {
    // Perform authentication logic here
    // Store user session or token
    res.status(200).json({ message: 'Login successful' });
  } else {
    res.status(405).json({ message: 'Method Not Allowed' });
  }
}

// pages/dashboard.js
import { useEffect, useState } from 'react';
import { useRouter } from 'next/router';

function Dashboard() {
  const [user, setUser] = useState(null);
  const router = useRouter();

  useEffect(() => {
    // Check user authentication
    // Redirect to login page if not authenticated
    if (!user) {
      router.push('/login');
    }
  }, [user, router]);

  return (
    <div>
      {user ? <h1>Welcome, {user.name}!</h1> : <p>Loading...</p>}
    </div>
  );
}

export default Dashboard;
```

1. Q: Explain the use of `getStaticPaths` in Next.js for dynamic routes with pre-rendering.
   A: `getStaticPaths` is used to specify dynamic route parameters that should be pre-rendered at build time. It allows you to generate static HTML pages for specific paths.

```
// pages/products/[id].js
export async function getStaticPaths() {
  const products = fetchAllProducts();
  const paths = products.map((product) => ({
    params: { id: product.id.toString() },
  }));

  return {
    paths,
    fallback: false,
```

```
  };
}

export async function getStaticProps({ params }) {
  const product = fetchProductById(params.id);

  return {
    props: {
      product,
    },
  };
}

function Product({ product }) {
  return (
    <div>
      <h1>{product.name}</h1>
      <p>{product.description}</p>
    </div>
  );
}

export default Product;
```

1. Q: How do you handle errors and display a custom error page in Next.js?
   A: You can use the `next.config.js` file to define custom error handling and create a custom error page.

```
// next.config.js
module.exports = {
  async redirects() {
    return [
      {
        source: '/old-path',
        destination: '/new-path',
        permanent: true,
      },
    ];
  },
  async rewrites() {
    return [
      {
        source: '/old-route',
        destination: '/new-route',
      },
    ];
  },
  async notFound(req, res, next) {
    // Custom 404 logic here
    res.status(404).send('Not Found');
  },
  async onError(error, req, res) {
    // Custom error handling here
```

```
    res.status(500).json({ error: 'Internal Server Error' });
  },
};
```

1. Q: How can you use external libraries in Next.js for specific pages without affecting the entire app?
   A: You can use the `noSSR` higher-order component (HOC) from `next/dynamic` to import external libraries dynamically, allowing them to be loaded only on the client-side for specific pages.

```
// pages/special-page.js
import dynamic from 'next/dynamic';

const DynamicComponent = dynamic(() => import('../components/special-library'), {
  ssr: false,
});

function SpecialPage() {
  return (
    <div>
      <h1>Special Page</h1>
      <DynamicComponent />
    </div>
  );
}

export default SpecialPage;
```

1. Q: How can you implement server-side rendering (SSR) with authentication in Next.js?
   A: You can fetch the user data on the server-side using `getServerSideProps` and pass it to the client-side as a prop. For authentication, you can use a token-based approach like JWT (JSON Web Tokens).

```
// pages/profile.js
import { parse } from 'cookie';
import jwt from 'jsonwebtoken';

export async function getServerSideProps(context) {
  const { req } = context;
  const cookies = parse(req.headers.cookie || '');
  const token = cookies.token;

  if (!token) {
    return {
      redirect: {
```

```
        destination: '/login',
        permanent: false,
      },
    };
  }

  try {
    const user = jwt.verify(token, 'secret-key');
    return {
      props: {
        user,
      },
    };
  } catch (error) {
    return {
      redirect: {
        destination: '/login',
        permanent: false,
      },
    };
  }
}

function Profile({ user }) {
  return <h1>Welcome, {user.name}!</h1>;
}

export default Profile;
```

1. Q: How can you implement pagination with server-side rendering in Next.js and maintain SEO-friendliness?
   A: You can use `getServerSideProps` to fetch paginated data based on the current page and pass it as props to the component. Additionally, use the `rel="prev"` and `rel="next"` link tags in the head to signal pagination to search engines.

```
// pages/products/[page].js
export async function getServerSideProps({ params }) {
  const pageNumber = parseInt(params.page);
  const pageSize = 10;
  // Fetch data for the current page
  const products = await fetchProducts(pageNumber, pageSize);

  return {
    props: {
      products,
    },
  };
}

function Products({ products }) {
  return (
    <div>
      {/* Render the list of products */}
```

```
    </div>
  );
}

export default Products;
```

```
<!-- Head of the page -->
<head>
  <link rel="prev" href="/products/1" />
  <link rel="next" href="/products/3" />
</head>
```

1. Q: How do you implement user authentication and authorization with role-based access control in Next.js?
   A: You can use an authentication provider like Auth0 or Firebase for user authentication and store user roles in the database. Then, use server-side or client-side checks to control access based on user roles.

```
// pages/dashboard.js
import { useAuth } from '../utils/auth';

function Dashboard() {
  const { user, isLoading } = useAuth();

  if (isLoading) {
    return <p>Loading...</p>;
  }

  if (!user) {
    return <p>You must log in to access the dashboard</p>;
  }

  if (user.role === 'admin') {
    return <p>Welcome, Admin!</p>;
  } else {
    return <p>Welcome, User!</p>;
  }
}

export default Dashboard;
```

1. Q: How can you implement search functionality in Next.js with server-side rendering?
   A: You can use a query parameter to pass the search query to the server-side and fetch relevant data based on the search term using `getServerSideProps`.

```
// pages/search.js
export async function getServerSideProps({ query }) {
  const searchQuery = query.q;
  const results = await searchProducts(searchQuery);

  return {
    props: {
      results,
    },
  };
}

function Search({ results }) {
  return (
    <div>
      {/* Display search results */}
    </div>
  );
}

export default Search;
```

1. Q: How can you optimize assets and lazy load components in Next.js for better performance?
   A: You can use the `next/image` component to optimize images automatically. For lazy loading components, use the `next/dynamic` package with the `defer` option.

```
// pages/index.js
import dynamic from 'next/dynamic';

const LazyComponent = dynamic(() => import('../components/LazyComponent'), {
  ssr: false,
  loading: () => <p>Loading...</p>,
});

function Home() {
  return (
    <div>
      <h1>Hello Next.js</h1>
      <LazyComponent />
    </div>
  );
}

export default Home;
```

1. Q: How can you implement server-side rendering for a multilingual website in Next.js?

A: Use `getServerSideProps` to fetch translations based on the user's language preference and pass the translated content as props to the components.

```
// pages/index.js
export async function getServerSideProps({ locale }) {
  const translations = await fetchTranslations(locale);
  return {
    props: {
      translations,
    },
  };
}

function Home({ translations }) {
  return (
    <div>
      <h1>{translations.homeTitle}</h1>
      <p>{translations.homeContent}</p>
    </div>
  );
}

export default Home;
```

1. Q: How do you handle real-time data and updates in Next.js with WebSockets?
   A: You can use a WebSocket library like `socket.io` or `ws` to set up a WebSocket server and handle real-time data updates on the client-side.

```
// pages/chat.js
import { useEffect, useState } from 'react';
import io from 'socket.io-client';

function Chat() {
  const [messages, setMessages] = useState([]);
  const socket = io('<https://api.example.com>');

  useEffect(() => {
    // Connect to WebSocket server
    socket.connect();

    // Listen for new messages
    socket.on('message', (message) => {
      setMessages((prevMessages) => [...prevMessages, message]);
    });

    return () => {
      // Disconnect from WebSocket server on unmount
      socket.disconnect();
    };
  }, []);

  const handleSendMessage = (message) => {
```

```
    // Send message to server
    socket.emit('message', message);
  };

  return (
    <div>
      <div>
        {messages.map((message) => (
          <p key={message.id}>{message.text}</p>
        ))}
      </div>
      <input type="text" onChange={(e) => handleSendMessage(e.target.value)} />
    </div>
  );
}

export default Chat;
```

1. Q: How can

you implement server-side caching in Next.js to reduce database queries and improve performance?
A: You can use a caching mechanism like Redis or Memcached to store the results of database queries and fetch data from the cache instead of the database when possible.

```
// pages/products/[id].js
import { getCache, setCache } from '../utils/cache';

export async function getServerSideProps({ params }) {
  const cacheKey = `product-${params.id}`;
  const cachedData = getCache(cacheKey);

  if (cachedData) {
    return {
      props: {
        product: cachedData,
      },
    };
  }

  const product = await fetchProductFromDatabase(params.id);
  setCache(cacheKey, product);

  return {
    props: {
      product,
    },
  };
}

function Product({ product }) {
  return (
```

```
    <div>
      <h1>{product.name}</h1>
      <p>{product.description}</p>
    </div>
  );
}

export default Product;
```

1. Q: How can you implement internationalization (i18n) with static site generation in Next.js?
   A: Use the `next-translate` package or the built-in Next.js internationalization support to define translations and generate static HTML pages for each supported language.

```
// pages/index.js
import useTranslation from 'next-translate/useTranslation';

function Home() {
  const { t } = useTranslation('common');

  return (
    <div>
      <h1>{t('title')}</h1>
      <p>{t('content')}</p>
    </div>
  );
}

export default Home;
```

1. Q: How do you handle data validation and input sanitization in Next.js for user-submitted data?
   A: Use libraries like `yup` or `validator` to validate user input on the server-side before storing it in the database or performing any operations.

```
// pages/contact.js
import { useState } from 'react';
import { useForm } from 'react-hook-form';
import * as yup from 'yup';

const schema = yup.object().shape({
  name: yup.string().required('Name is required'),
  email: yup.string().email('Invalid email').required('Email is required'),
  message: yup.string().required('Message is required'),
});

function Contact() {
  const { register, handleSubmit, errors } = useForm({
```

```
    validationSchema: schema,
  });

  const onSubmit = (data) => {
    // Handle form submission
    console.log(data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input type="text" name="name" ref={register} />
      {errors.name && <p>{errors.name.message}</p>}

      <input type="email" name="email" ref={register} />
      {errors.email && <p>{errors.email.message}</p>}

      <textarea name="message" ref={register} />
      {errors.message && <p>{errors.message.message}</p>}

      <button type="submit">Submit</button>
    </form>
  );
}

export default Contact;
```