1. what is nextjs ?
2. features of nextjs ?
3. example to demonstrate how to create a custom error page in Next.js?
4. Data Fetching Methods in Next.js

Qua1. What Is Nextjs ?

Ans. => Next.Js Is An Open-Source, Lightweight React.Js Framework That Facilitates Developers To Build Static And Server-Side Rendering Web Applications. It Was Created By Zeit. Next.Js Framework Is Based On React, Webpack, And Babel And Allows Us To Write Server-Rendered React Apps Easily. It Doesn't Require Any Webpack Configuration And Only Needs Npm Run Dev Start Building Your Next Feature-Filled Web Application.

Qua2. Features Of Nextjs ?

Ans. => Reasons Why The World's Leading Companies Prefer Next.Js:

1. Zero Setup: Next.Js Provides Automatic Code-Splitting, Filesystem-Based Routing, Hot Code Reloading, And Universal Rendering; That's Why The World's Leading Companies Prefer It.
2. Fully Extensible: Next.Js Is Fully Extensible And Has Complete Control Over Babel And Webpack. It Provides A Customizable Server, Routing, And Next Plugins.
3. Ready For Production: Next.Js Is Optimized For Smaller Build Sizes, Faster Dev Compilation, And Many Other Improvements, Making It A Popular Choice.
4. Next.Js Can Deploy Anywhere: Next.Js Is An Open-Source, Lightweight React.Js Framework That Facilitates Developers To Build Static And Server-Side Rendering Web Applications.
5. Js Provides The By Default And Easy Server Rendering.
6. Js Supports Static Exporting.
7. It Provides A Webpack-Based Dev Environment Which Supports Hot Module Replacement (HMR)
8. It Seaports Automatic Code-Splitting For Faster Page Loads.
9. It Supports Simple Client-Side Routing (Page-Based) Or File System-Based Routing.
10. It Provides Complete Webpack And Babel Control.
11. It Provides A Faster And Optimized Development Compilation.
12. It Can Be Implemented With Express Or Any Other Node.Js HTTP Server.
13. You Can Easily Customize It With Your Own Babel And Webpack Configurations.
14. It Supports Hot Code Reloading.

Qua. Example To Demonstrate How To Create A Custom Error Page In Next.Js? ?

Ans. =>

```
Import React From "React";
Class Error Extends React.Component {
  Static GetInitialProps({ Res, Err }) {
    Const StatusCode = Res ? Res.StatusCode : Err ? Err.StatusCode : Null;
    Return { StatusCode };
  }
  Render() {
    Return (

        {This.Props.StatusCode
          ? `An Error ${This.Props.StatusCode} Has Occurred On The Server`
          : "An Error Occurred On Client-Side"}

    );
  }
}
Export Default Error;
```

Qua4. Data Fetching Methods In Next.Js ?

Ans. =>

Next.Js Provides Three Data Fetching Methods And Based On These Methods, It Renders Content Differently. (You Can Learn About Different Rendering Methods Here.

1. GetStaticProps
2. GetStaticPaths
3. GetServerSideProps

GetStaticProps: It Preloads All Of The Data Needed For A Given Page And Renders Pages Ahead Of The User's Request At Build Time. For Speedier Retrieval, All Of The Data Is Cached On A Headless CMS. For Better SEO Performance, The Page Is Pre-Rendered And Cached. If No Other Data Fetching Method Is Specified, Next.Js Will Use This By Default. It Is Used To Implement Static Site Generation And Incremental Site Regeneration.

Properties Of GetStaticProps:

It Can Only Be Exported From The Page File, Not The Component File.
It Only Runs On Build Time.
It Runs On Every Subsequent Request In Development Mode.
Its Code Is Completely Excluded From The Client-Side Bundle.

GetStaticPaths: If A Page Uses GetStaticProps And Has Dynamic Routes, It Must Declare A List Of Paths That Will Be Statically Generated. Next.Js Will Statically Pre-Render All The Paths Defined By GetStaticPaths When We Export A Function Named GetStaticPaths From A Page.

Properties Of GetStaticPaths:

It Can Only Be Exported From A Page File.
It Is Meant For Dynamic Routes.
Page Must Also Implement GetStaticProps.

It Runs Only At Build Time In Production.
It Runs On Every Request In Development Mode.

GetServerSideProps: It Will Pre-Render The Page On Every Subsequent Request. It Is Slower As Compared To GetStaticProps As The Page Is Being Rendered On Every Request. GetServerSideProps Props Return JSON Which Will Be Used To Render The Page All This Work Will Be Handled Automatically By Next.Js. It Could Be Used For Calling A CMS, Database, Or Other APIs Directly From GetServerSideProps. It Is Used To Implement Server Side Rendering.

Properties Of GetServerSideProps:

It Runs On Every Subsequent Request In Development As Well As Production Mode.
Its Code Is Excluded From The Client-Side Bundle.
It Can Only Be Exported From Page File.
When To Use Which Data Fetching Method: If Your Page's Content Is Static Or Doesn't Change Frequently Then You Should Go For GetStaticProps As It Builds Pages On Build Time Hence Increasing Performance. If Your Page Has Dynamic Routes Then GetStaticPaths Should Be Used Along With GetStaticProps.

But If Your Website Contains A Page Whose Data Changes Very Frequently Then You Must Use GetServerSideProps As It Fetches Fresh Data On Every Request.

Example: We Will Build A Simple Next Js Application With Three Pages Of Albums, Posts, And A Users Page With Dynamic Routes. All Three Pages Will Implement Different Data Fetching Methods. For This Example, We Will Use JSONPlaceholder API To Fetch Random Data.

```
Import React From "React";
Import Link From "Next/Link";
Const Home = () => {
  // This Is The Home Page Which Will
  // Contain Links To All Other Pages
  Return (
    <>
      <H1>Hello Geeks</H1>
      <Ul>
        <Li>
          GetStaticProps :<Link Href={"/About"}>About Page</Link>
        </Li>
        <Li>
          GetStaticPaths :<Link Href={"/Users/1"}>User 1</Link>
        </Li>
        <Li>
          GetServerSideProps :<Link Href={"/Posts"}>Posts Page</Link>
        </Li>
      </Ul>
    </>
  );
};

Export Default Home;
```

/Pages/Albums.Jsx – Albums Page Will Implement Static Site Generation Using GetStaticProps, We Will Export The Data Fetching Method Along With The Page Component. We Can Send Fetched Data To The Page Component Using Props. Next Js Will Fetch All The Albums At Build Time Before The User's Request.

```
Import React From "React";

Export Const GetStaticProps = Async () => {
  // Fetching Data From Jsonplaceholder.
  Const Res = Await Fetch("Https://Jsonplaceholder.Typicode.Com/Albums");
  Let AllAlbums = Await Res.Json();

  // Sending Fetched Data To The Page Component Via Props.
  Return {
    Props: {
      AllAlbums: AllAlbums.Map((Album) => Album.Title),
    },
  };
};

Const Albums = ({ AllAlbums }) => {
  Return (
    <Div>
      <H1>All Albums</H1>
      {AllAlbums.Map((Album, Idx) => (
        <Div Key={Idx}>{Album}</Div>
      ))}
    </Div>
  );
};

Export Default Albums;
```

/Pages/Posts.Jsx – Posts Page Will Implement Server-Side Rendering Using GetServerSideProps. It'll Fetch Posts Data And Build Page At Each Request Made By User., And Send Fetched Data To The Component Using Props.

```
Import React From "React";

Export Const GetServerSideProps = Async (Ctx) => {
  // Ctx Is The Context Object Which Contains The Request,
  // Response And Props Passed To The Page.

  // Fetching Data From Jsonplaceholder.
  Const Res = Await Fetch("Https://Jsonplaceholder.Typicode.Com/Posts");
  Let AllPosts = Await Res.Json();

  // Sending Fetched Data To The Page Component Via Props.
  Return {
    Props: {
      AllPosts: AllPosts.Map((Post) => Post.Title),
    },
  };
};

Const Posts = ({ AllPosts }) => {
  Return (
    <Div>
      <H1>All Posts</H1>
      {AllPosts.Map((Post, Idx) => (
        <Div Key={Idx}>{Post}</Div>
      ))}
    </Div>
  );
};

Export Default Posts;
```

/Pages/Users/[Id].Jsx – Because This Is A Dynamic Page, We Must Pre-Define All Of The User Ids So That Next Js Can Retrieve Their Data At Build Time. As A Result, We Use GetStaticPaths And Define Ten User Ids.

```
Import React From "React";

Export Const GetStaticProps = Async (Ctx) => {
  // Ctx Will Contain Request Parameters
  Const { Params } = Ctx;

  // We Will Destructure Id From The Parameters
  Const UserId = Params.Id;

  // Fetching User Data
  Const Res = Await Fetch(
    `Https://Jsonplaceholder.Typicode.Com/Users/${UserId}`
  );
  Const UserData = Await Res.Json();

  // Sending Data To The Page Via Props
  Return {
    Props: {
      User: UserData,
    },
  };
};

Export Const GetStaticPaths = () => {
  // Specifying All The Routes To Be
  // Pre-Rendered By Next Js
  Return {
    Paths: [
      { Params: { Id: "1" } },
      { Params: { Id: "2" } },
      { Params: { Id: "3" } },
      { Params: { Id: "4" } },
      { Params: { Id: "5" } },
      { Params: { Id: "6" } },
      { Params: { Id: "7" } },
      { Params: { Id: "8" } },
      { Params: { Id: "9" } },
      { Params: { Id: "10" } },
    ],
    Fallback: False,
  };
};

Const User = ({ User }) => {
  Return (
    <>
      <H1>User {User.Id}</H1>
      <H2>Name : {User.Name}</H2>
      <H2>Email : {User.Email}</H2>
    </>
  );
};

Export Default User;
```