As the popularity of Nuxt continues to grow, so does the demand for developers with experience in the framework. To help you prepare for your next Nuxt interview, we've compiled a list of 10 of the most common Nuxt interview questions and answers for 2023. Whether you're a seasoned Nuxt developer or just getting started, this guide will provide you with the knowledge and confidence you need to ace your next interview.

**1. How would you explain the concept of server-side rendering in Nuxt?**

Server-side rendering (SSR) is a technique used in Nuxt.js to render a web page on the server instead of the client. This allows for faster page loads and improved SEO, as the content is already rendered on the server before it is sent to the client.  
  
Nuxt.js uses a Node.js server to render the page on the server. It uses the Vue.js framework to create the page, and then renders it on the server. This allows for faster page loads, as the page is already rendered before it is sent to the client.  
  
Nuxt.js also provides a number of features to make server-side rendering easier. These include:  
  
- Automatic code-splitting: Nuxt.js automatically splits the code into smaller chunks, which are then rendered on the server. This helps to reduce the amount of code that needs to be sent to the client.  
  
- Pre-rendering: Nuxt.js can pre-render pages on the server, which helps to reduce the amount of time it takes for the page to load.  
  
- Server-side caching: Nuxt.js can cache the rendered page on the server, which helps to reduce the amount of time it takes for the page to load.  
  
Overall, server-side rendering in Nuxt.js is a great way to improve the performance of your web application. It helps to reduce the amount of code that needs to be sent to the client, and can help to improve the page load time.

**2. What is the purpose of the Nuxt.js configuration file?**

The Nuxt.js configuration file is a key part of the Nuxt.js framework. It is used to define the configuration options for the Nuxt.js application. This includes options such as the mode, the server configuration, the build configuration, the plugins, the modules, and the routing configuration.  
  
The Nuxt.js configuration file is used to define the application's behavior and structure. It is used to define the application's entry point, the directory structure, the plugins, the modules, and the routing configuration. It also allows developers to customize the application's behavior and structure by setting options such as the mode, the server configuration, the build configuration, and the plugins.  
  
The Nuxt.js configuration file is also used to define the application's environment variables. This allows developers to set environment-specific variables such as the API URL, the database connection string, and the application's port.  
  
The Nuxt.js configuration file is an essential part of the Nuxt.js framework and is used to define the application's behavior and structure. It is used to define the application's entry point, the directory structure, the plugins, the modules, and the routing configuration. It also allows developers to customize the application's behavior and structure by setting options such as the mode, the server configuration, the build configuration, and the plugins.

**3. How would you set up a Nuxt project to use TypeScript?**

To set up a Nuxt project to use TypeScript, the following steps should be taken:  
  
1. Install the Nuxt TypeScript module:  
  
Using npm:  
  
npm install @nuxt/typescript  
  
Using yarn:  
  
yarn add @nuxt/typescript  
  
2. Create a tsconfig.json file in the root of your project:  
  
This file will contain the TypeScript configuration for your project.  
  
3. Add the following to your nuxt.config.js file:  
  
module.exports = { buildModules: [ '@nuxt/typescript' ], typescript: { typeCheck: { eslint: true } } }  
  
4. Create a .eslintrc.js file in the root of your project:  
  
This file will contain the ESLint configuration for your project.  
  
5. Create a .prettierrc.js file in the root of your project:  
  
This file will contain the Prettier configuration for your project.  
  
6. Create a .vscode folder in the root of your project:  
  
This folder will contain the VS Code configuration for your project.  
  
7. Create a src folder in the root of your project:  
  
This folder will contain all of your TypeScript files.  
  
8. Create a main.ts file in the src folder:  
  
This file will be the entry point for your project.  
  
9. Create a nuxt.config.ts file in the src folder:  
  
This file will contain the Nuxt configuration for your project.  
  
10. Create a tsconfig.json file in the src folder:  
  
This file will contain the TypeScript configuration for your project.  
  
11. Create a .eslintrc.js file in the src folder:  
  
This file will contain the ESLint configuration for your project.  
  
12. Create a .prettierrc.js file in the src folder:  
  
This file will contain the Prettier configuration for your project.  
  
13. Create a .vscode folder in the src folder:  
  
This folder will contain the VS Code configuration for your project.  
  
14. Run the following command to compile your TypeScript files:  
  
npx tsc  
  
15. Run the following command to start your Nuxt project:  
  
npm run dev

**4. What is the difference between Nuxt.js and Vue.js?**

Nuxt.js is a framework built on top of Vue.js that provides an opinionated structure for developing universal applications. It is designed to make the development of Vue applications easier and more powerful by abstracting away common development tasks such as routing, server-side rendering, and code-splitting.  
  
Vue.js is a progressive JavaScript framework for building user interfaces. It is designed to be incrementally adoptable, allowing developers to start with the basic structure of a single-page application and gradually scale up to more complex applications.  
  
The main difference between Nuxt.js and Vue.js is that Nuxt.js provides an opinionated structure for developing universal applications, while Vue.js is a progressive JavaScript framework for building user interfaces. Nuxt.js provides a set of features that make it easier to develop universal applications, such as routing, server-side rendering, and code-splitting. Vue.js, on the other hand, is designed to be incrementally adoptable, allowing developers to start with the basic structure of a single-page application and gradually scale up to more complex applications.

**5. How would you create a Nuxt plugin to add custom functionality to your application?**

Creating a Nuxt plugin to add custom functionality to an application is a relatively straightforward process.  
  
First, create a new directory in the plugins folder of your Nuxt application. This directory should contain a file named index.js, which will be the main entry point for your plugin.  
  
In the index.js file, you will need to export a function that will be called when the plugin is registered. This function should accept the Nuxt context as an argument. The context contains the Nuxt instance, which can be used to access the Nuxt application's configuration, store, router, and other components.  
  
Within the function, you can add custom functionality to the Nuxt application. This could include adding custom routes, registering global components, or adding custom middleware.  
  
Once you have added the custom functionality, you will need to register the plugin in the nuxt.config.js file. This can be done by adding the plugin to the plugins array.  
  
Finally, you will need to build and deploy the Nuxt application for the changes to take effect.  
  
By following these steps, you can create a Nuxt plugin to add custom functionality to your application.

**6. What is the purpose of the Nuxt.js router?**

The Nuxt.js router is a powerful routing system that provides a way to create and manage application routes. It is based on Vue Router and provides a number of features that make it easier to create and manage routes in a Nuxt.js application.  
  
The Nuxt.js router provides a number of features that make it easier to create and manage routes in a Nuxt.js application. It allows you to define routes in a single file, which makes it easier to maintain and update routes. It also provides a way to define dynamic routes, which allows you to create routes that are based on dynamic data. Additionally, it provides a way to define nested routes, which allows you to create routes that are nested within other routes. Finally, it provides a way to define custom routes, which allows you to create routes that are tailored to your application's specific needs.  
  
Overall, the Nuxt.js router is a powerful routing system that makes it easier to create and manage routes in a Nuxt.js application. It provides a number of features that make it easier to create and manage routes, and it allows you to create routes that are tailored to your application's specific needs.

**7. How would you set up Nuxt.js to use a custom server?**

To set up Nuxt.js to use a custom server, the following steps should be taken:  
  
1. Install Nuxt.js:  
  
First, you need to install Nuxt.js. This can be done using npm or yarn.  
  
2. Create a Nuxt.js project:  
  
Once Nuxt.js is installed, you can create a Nuxt.js project. This can be done using the Nuxt.js CLI.  
  
3. Configure Nuxt.js:  
  
Next, you need to configure Nuxt.js to use a custom server. This can be done by editing the nuxt.config.js file. In this file, you can specify the server configuration, such as the host, port, and other settings.  
  
4. Create a custom server:  
  
Once Nuxt.js is configured, you need to create a custom server. This can be done using a framework such as Express or Koa. The server should be configured to handle requests from Nuxt.js and respond with the appropriate data.  
  
5. Start the server:  
  
Finally, you need to start the server. This can be done using the Nuxt.js CLI. Once the server is running, Nuxt.js will be able to communicate with it and serve the appropriate data.

**8. How would you debug a Nuxt.js application?**

When debugging a Nuxt.js application, the first step is to identify the source of the issue. This can be done by examining the browser console for any errors or warnings that may be present. Additionally, it is important to check the Nuxt.js application logs for any errors or warnings that may be present.  
  
Once the source of the issue has been identified, the next step is to determine the cause of the issue. This can be done by examining the code for any errors or bugs that may be present. Additionally, it is important to check the Nuxt.js documentation for any known issues or bugs that may be related to the issue.  
  
Once the cause of the issue has been identified, the next step is to determine the best way to fix the issue. This can be done by examining the code for any potential solutions that may be present. Additionally, it is important to check the Nuxt.js documentation for any potential solutions that may be related to the issue.  
  
Finally, once the issue has been fixed, it is important to test the application to ensure that the issue has been resolved. This can be done by running the application in a development environment and testing the application for any potential issues. Additionally, it is important to check the Nuxt.js documentation for any known issues or bugs that may be related to the issue.

**9. What is the purpose of the Nuxt.js store?**

The Nuxt.js store is a centralized state management system for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion. The store can be used to store data that is retrieved from an API, manage the state of the application, and trigger side effects. It also provides a way to reactively update components when the store's state changes. The store can be used to keep the state of the application in sync across multiple components, and to ensure that the application is always in a consistent state. By using the store, developers can easily manage the state of their application and ensure that all components are always up-to-date.

**10. How would you set up Nuxt.js to use a custom API?**

To set up Nuxt.js to use a custom API, the following steps should be taken:  
  
1. Create a Nuxt.js project. This can be done by running the command `npx create-nuxt-app ` in the terminal.  
  
2. Install the Axios module. This can be done by running the command `npm install axios` in the terminal.  
  
3. Create a new file in the `/plugins` directory called `axios.js`. This file will be used to configure Axios to use the custom API.  
  
4. In the `axios.js` file, add the following code:  
  
import axios from 'axios'  
  
export default ({ app }, inject) => { const api = axios.create({ baseURL: '' }) inject('api', api) }  
  
5. In the `nuxt.config.js` file, add the following code to the `plugins` array:  
  
{ src: '~/plugins/axios.js', ssr: false }  
  
6. In the `nuxt.config.js` file, add the following code to the `env` object:  
  
API\_URL: ''  
  
7. In the `nuxt.config.js` file, add the following code to the `axios` object:  
  
baseURL: process.env.API\_URL  
  
8. In the `nuxt.config.js` file, add the following code to the `build` object:  
  
publicPath: process.env.API\_URL  
  
9. In the `nuxt.config.js` file, add the following code to the `head` object:  
  
script: [ { src: `${process.env.API\_URL}/script.js` } ]  
  
10. In the `nuxt.config.js` file, add the following code to the `modules` array:  
  
['@nuxtjs/axios']  
  
11. In the `nuxt.config.js` file, add the following code to the `axios` object:  
  
proxy: true  
  
12. In the `nuxt.config.js` file, add the following code to the `proxy` object:  
  
'/api': { target: process.env.API\_URL, pathRewrite: { '^/api': '' } }  
  
13. In the `nuxt.config.js` file, add the following code to the `build` object:  
  
extractCSS: true  
  
14. In the `nuxt.config.js` file, add the following code to the `build` object:  
  
postcss: { plugins: { 'postcss-url': { url: 'inline' } } }  
  
15. In the `nuxt.config.js` file, add the following code to the `build` object:  
  
extractCSS: { ignoreOrder: true }  
  
Once these steps have been completed, Nuxt.js should be set up to use the custom API.