1. Explain the purpose of the backend?

The backend, also known as the server-side, is the software that powers a website or app. It's responsible for storing and organizing data, handling user requests, and delivering content to the front end.

2. What is a typical workflow for implementing a new feature on the backend?

Workflows for implementing features on the backend can vary depending on the company and the technology stack. However, a typical workflow would involve discussing the feature with the stakeholders, designing and prototyping the feature, writing the code, and doing Quality Assurance (QA) testing. In most cases, the backend developer will work with the front-end developer to ensure that data is correctly transmitted between the client and server. It's also essential to ensure that any new features are backward compatible with previous versions of the application.

3. Explain the essence of DRY and DIE principles?

The DRY (Don't Repeat Yourself) principle is a software development principle that states that developers should not duplicate code. Duplicated code can lead to maintenance issues because changes need to be made in multiple places. The DIE (Duplication Is Evil) principle is similar to the DRY principle. Still, it takes it one step further by stating that even slight amounts of duplication are inadequate and should be avoided.

4. What is a web server?

A web server is a computer that stores and delivers web pages. When you type a URL into your browser, the browser contacts the web server and requests the page. The web server then sends the page back to the browser, which displays it on your screen. Apache and NGINX are some of the most popular web servers used by backend applications.

Web servers can also host other resources, like images or videos.

5. What is the difference between a GET and a POST request?

A GET request retrieves data from a server, whereas a POST sends data to a server. With a GET request, parameters get passed in the URL. With a POST request, parameters get passed in the request's body.

6. What is an example of when you would use caching?

Caching is often used to improve the performance of an application. For example, frequently accessed information from a database may be cached to avoid unnecessary queries.

7. How would you select a cache strategy (e.g., LRU, FIFO)?

Selecting a cache strategy depends on the application's specific needs. For example, LRU (Least Recently Used) is a good choice for applications where frequently accessed data is also likely to be reaccessed. FIFO (First In, First Out) is a good choice for an application where data expires after a particular time.

8. What are some common issues with ORMs?

Some common issues with ORMs include performance degradation, incorrect data mapping, and difficulty handling complex queries.

9. When should you use asynchronous programming?

Asynchronous programming is often used when there is a need to improve the performance of an application. For example, if an application needs to make many database queries, it may be beneficial to

10. What is the difference between promises and callbacks?

A promise is an object that represents the result of an asynchronous operation. A callback is a function that is invoked when an asynchronous operation completes.

11. What is a closure?

A closure is a function that accesses variables “outside” itself, in another function’s scope, even after the parent function has closed.

12. What is the difference between a Class and an Interface in Java?

A class is a blueprint for an object, whereas an interface is a contract that defines the methods that a class must implement.

13. What is continuous integration?

Continuous Integration (CI) is a software development practice in which developers regularly merge their code changes into a shared repository. CI helps to ensure the codebase is always stable and there are no conflicts between different branches of code.

14. What is a software development kit (SDK)?

SDK is a set of tools that helps developers build software applications. It typically includes a compiler, debugger, and other utilities. Some common SDKs used for backend development include the Java Development Kit (JDK) and the Python Software Development Kit (SDK).

15. What are the tradeoffs of client-side rendering vs. server-side rendering?

There are a few tradeoffs when deciding between client-side rendering and server-side rendering.

Client-side rendering can be more complex to set up because the application needs to be able to run in a browser. Meaning the code must be transpiled from a higher-level language (such as JavaScript) to a lower-level language (such as HTML). In addition, client-side rendering can be slower because the application needs to download all of the necessary resources before it can start rendering.

On the other hand, server-side rendering is typically easier to set up because developers can use any language capable of generating HTML. In addition, server-side rendering can be faster because the HTML can be generated on the server and then sent to the client.

Please note that these tradeoffs often depend on the site's complexity and function.

16. What are high-order functions? Why are they useful?

High-order functions are functions that take other functions as arguments. These functions help abstract common patterns of code. For example, a high-order function could create a function that logs the arguments it is called with. This would be useful for debugging purposes.

17. What is a microservice?

A microservice is a small, independent component of a more extensive application. Each microservice has its own functionality and can be deployed independently.

Microservices often build into large, complex applications that are easy to maintain and scale. One of the benefits of using microservices is that they can be written in different programming languages and deployed on different servers.

Common examples of microservices include user authentication, payment processing, and image manipulation.

API Questions

18. How would you design an API?

When designing an API, it's helpful to consider the needs of the developers who use the API. The API should be easy to use and well-documented. It's also critical to consider the API's security and ensure that only authorized users can access the data. Additionally, the

API should be able to handle a large number of requests without overloading the server.

19. What is the difference between a RESTful and a SOAP API?

RESTful APIs are designed to be easy to use and well-documented. They use a standard set of rules, which makes them easy to learn and use. On the other hand, SOAP APIs are designed to be more secure and can handle a larger number of requests. However, they are more complex to learn and use.

20. How do you handle errors when making API calls?

When making API calls, it's industry standard to handle errors in a way consistent with the rest of the application. For example, if the API returns a 404 error, you might want to display a message to the user saying the data could not be found.