11 questions about experience and background

1. How did you get into coding?

2. What's your greatest strength as a coder?"

-Adaptability:

-"My greatest strength as a coder is my adaptability to new technologies and languages. During my time at Company A, I was initially hired as a backend developer. However, when the team needed assistance with frontend development due to an unexpected workload, I quickly learned JavaScript, HTML, and CSS to contribute effectively. This ability to adapt allowed me to contribute to multiple aspects of the project and ultimately expedited the delivery timeline. I'm confident that this adaptability will be valuable in roles that require working with diverse technology stacks and evolving industry trends."

-Code Optimization:

"I consider code optimization to be my primary strength as a coder. In a recent project at Company B, I identified several performance bottlenecks within the application's database queries. By rewriting and fine-tuning these queries, I was able to reduce the query execution time by 40%, resulting in a significantly improved user experience. I believe that efficient code not only enhances application speed but also reduces resource consumption and costs. This strength is particularly crucial in roles that involve optimizing existing codebases and enhancing overall system performance.

3. . If you were in charge of a tech company, how would you manage its developers?

I would strive to create a positive, productive, and collaborative environment that fosters innovation, growth, and the successful execution of projects. Here are some strategies I would consider implementing:

Skill Development:
Support developers' professional growth by providing opportunities for skill development and continuous learning. Offer training, workshops, and access to resources that enable them to stay up-to-date with the latest technologies and best practices.

Clear Vision and Goals:
Define a clear vision for the company and communicate it to the development team. Establish achievable goals that align with the company's mission and vision, and ensure that each developer understands how their work contributes to the overall success of the organization.

4. Tell me about a time when someone criticized your work and explain how you responded.

Being human, I can definitely admit to receiving criticism of my work in the past. Honestly, the only thing you can do at that point is to thank the person for pointing out the flaw.

5. How to dilver negative feedback ?
My way is :

Choose the Right Time and Place:
Find a private and comfortable setting to have the conversation. Avoid discussing negative feedback in public or high-stress situations. Ensure you have enough time for a meaningful conversation without rushing.

Focus on Behavior and Results, Not Personalities:
Frame the feedback around specific behaviors, actions, or outcomes rather than making it about the individual's personality or character. Stick to observable facts rather than making assumptions.

Be Specific and Concrete:
Clearly state the issue at hand by providing specific examples.

Try to support others by giving them resources and training that help them as possible

finally End on a Positive Note:
Emphasize the individual's strengths and positive qualities. Express confidence in their ability to make the necessary improvements. Ending on a positive note can help maintain their motivation and self-esteem.

6.Have you ever worked on a team project where you felt you were doing most of the work? How did you manage that?
Delegate and Empower:
If you find yourself taking on most of the work, it's possible that others may not feel empowered to contribute
Open Communication:
Initiate a conversation with your team members to discuss your concerns

7.Tell me about the coding accomplishment you’re most proud of?
By Develop a gallery project , it was static , and i converted it to dynamic connected with db and made it asynchronous , so it’s become more fast

8. What's the most challenging decision you've faced in your career?

9.What’s your experience with object-oriented programming (OOP)?
OOP is a programming paradigm that is widely used in software development. It is based on the concept of "objects," which are instances of classes that encapsulate data and behavior.

In OOP, you define classes to represent real-world entities or concepts, and these classes can have attributes (data) and methods (functions) associated with them. The key principles of OOP include:

Encapsulation: Encapsulation is the concept of bundling data (attributes) and the methods (functions) that operate on that data into a single unit, i.e., the class. This helps in hiding the internal details of how a class works, and it exposes only the necessary interfaces to the outside world.

Inheritance: Inheritance allows you to create a new class (subclass or derived class) by inheriting the properties and behaviors of an existing class (superclass or base class). This promotes code reuse and allows you to create a hierarchy of classes.

Polymorphism: Polymorphism means the ability of different objects to respond to the same method or function call in a way that is specific to their individual types. This is often achieved through method overriding and interfaces/abstract classes.

Abstraction: Abstraction is the process of simplifying complex reality by modeling classes based on the essential properties and behaviors of real-world entities. It helps in managing the complexity of large software systems.

Modularity: OOP promotes the development of modular code, where each class represents a self-contained unit of functionality. This makes it easier to understand, maintain, and extend software systems.

10.what is the differenece between interfaces/abstract classes?

Interfaces and abstract classes are both mechanisms used in object-oriented programming (OOP) to define contracts for classes, but they serve slightly different purposes and have some key differences:

Abstract Classes:

1. **Abstractness:** An abstract class can have a mix of concrete (implemented) methods and abstract (unimplemented) methods. Abstract methods are declared with the `abstract` keyword and have no implementation in the abstract class itself.

2. **Inheritance:** Abstract classes support inheritance, which means you can create subclasses that extend the abstract class and provide implementations for its abstract methods. Subclasses can also inherit and override the concrete methods of the abstract class.

3. **Constructors:** Abstract classes can have constructors, and when you create an instance of a subclass, the constructor of the abstract class is called before the constructor of the subclass.

4. **Fields (Properties):** Abstract classes can have fields (instance variables) just like regular classes, and they can also define properties.

5. **Use Cases:** Abstract classes are often used when you want to provide a common base for a group of related classes. They can contain shared code and enforce certain behavior in subclasses while allowing flexibility in how that behavior is implemented.

Interfaces:

1. **Abstractness:** Interfaces are purely abstract; they only declare method signatures but do not provide any implementation. All methods in an interface are implicitly `public` and `abstract`.

2. **Inheritance:** A class can implement multiple interfaces. This allows a class to inherit multiple sets of method signatures, making interfaces a way to achieve multiple inheritance in languages that don't support multiple inheritance with classes.

3. **Constructors:** Interfaces cannot have constructors because they don't have any implementation. You cannot create an instance of an interface.

4. **Fields (Properties):** Interfaces can declare constants (fields that are `public`, `static`, and `final`) but not instance variables (non-static fields). Starting with Java 8 and some other modern programming languages, interfaces can also define default methods, which are methods with default implementations.

5. **Use Cases:** Interfaces are used when you want to define a contract that multiple classes can adhere to, regardless of their inheritance hierarchy. They are particularly useful for achieving polymorphism and for defining common behaviors that are not tied to a specific base class.

11.What’s your experience with GoTo, and do you prefer structured programming?

The use of the "goto" statement is generally discouraged in modern programming because it can lead to code that is difficult to read, understand, and maintain. The "goto" statement can create spaghetti code, which is characterized by unstructured and convoluted control flow, making it challenging to debug and modify.

Structured programming, on the other hand, is a programming paradigm that promotes the use of structured control flow constructs such as loops, conditionals,

and functions or methods. These constructs make code more organized, readable, and maintainable.

Structured programming principles emphasize:

1. **Sequential execution:** Code is executed in a straight-line fashion, making it easier to follow.

2. **Structured control flow:** The use of loops (like "for" and "while" loops) and conditionals (like "if-else" statements) to control program flow.

3. **Modularization:** Breaking down code into smaller, reusable functions or methods to enhance code readability and reusability.

4. **Avoiding global variables:** Encouraging the use of local variables and parameters to limit the scope of variables, reducing the risk of unintended side effects.

The "goto" statement, which allows for unconditional jumps in code, can disrupt these principles and make code harder to reason about

10 in-depth questions

1.How would you explain the difference between design and architecture?

software architecture is about the high-level structure and organization of the entire system, focusing on global concerns and decisions that impact the entire project. Software design, on the other hand, is about the detailed internal structure and implementation of individual components or modules, focusing on specific functionality and algorithms. Both are critical aspects of software development and complement each other to create robust and maintainable software systems.

2.Define the terms “stack” and “heap.” What’s a stack overflow?
stack : Data structure used to store data , its a last-in first-out (Lifo) where the most current function called is in the above and when return it removed firstly.

Heap: is a region of memory used for dynamic allocation.It's a more flexible and expansive area of memory compared to the stack.
Stack Overflow:

A "stack overflow" is a runtime error that occurs when the stack, which has a limited size, becomes full due to excessive function calls or recursive calls. When a function is called, its call frame is pushed onto the stack, including its local variables and

other information. If there are too many nested function calls, the stack can run out of space, leading to a stack overflow.

Stack overflows are typically caused by issues such as infinite recursion (a function calling itself indefinitely) or deep recursion (a large number of nested function calls).

3.What's the difference between cohesion and coupling?
Cohesion:

Cohesion refers to the degree to which the elements (e.g., functions, methods, or modules) within a single module or component are related to each other and work together to perform a specific, well-defined task or function. In other words, it measures how focused and self-contained a module is.

Coupling:

Coupling refers to the degree of dependence or interaction between different modules or components within a software system. It measures how much one module relies on the internals of another module
Reducing coupling between modules is essential for building flexible and maintainable software systems. It allows for easier unit testing, code reuse, and the ability to change one part of the system without affecting others.

In summary, cohesion is about how well the elements within a single module work together, while coupling is about how modules interact with each other. Striving for high cohesion and low coupling is a fundamental principle in software design and architecture to create robust, maintainable, and extensible software systems.

4.When is refactoring useful?
Refactoring is a valuable practice in software development that involves making changes to the code to improve its structure, readability, maintainability, and/or performance without altering its external behavior. Refactoring is useful in several situation:
a.**Improving Readability:**
b.*Enhancing Maintainability:
c.Performance Optimization:
d.Code Reuse:*

5.What do the terms “high cohesion” and “loose coupling” mean?

High Cohesion:
Meaning: High cohesion refers to the degree to which the elements (e.g., functions, methods, or modules) within a single module or component are related to each other and work together to perform a specific, well-defined task or function.

Loose Coupling:
Meaning: Loose coupling refers to the degree of dependence or interaction between different modules or components within a software system. It measures how much one module relies on the internals of another module.
Example: A well-designed microservices architecture typically exhibits loose coupling. Each microservice operates independently, communicates through APIs or message queues, and doesn't have direct dependencies on the internal implementation of other microservices

In summary, high cohesion focuses on how well elements within a single module or component work together, aiming for a clear and focused purpose. Loose coupling, on the other hand, deals with how modules or components interact with each other, striving to minimize dependencies and promote independence. Both high cohesion and loose coupling are essential principles for creating maintainable and scalable software systems.

6.What are the pros and cons of holding domain logic in stored procedures?

In summary, the decision to use stored procedures for domain logic should be made based on the specific needs and constraints of your project. While they can provide performance and security benefits, they also introduce challenges related to code separation, portability, and maintainability. A balanced approach, such as using stored procedures for performance-critical operations while keeping the core business logic in the application layer, may be a suitable compromise in some cases.

7.What do you think makes object-oriented design the preferred approach?
Abstraction :allows developers to model real-world entities and their interactions in a natural and intuitive way.making it easier to understand and map the problem domain to the software solution.
Code Reusability: OOD encourages the creation of reusable classes and objects. Libraries of classes can be developed and shared, saving time and effort in future projects. Inheritance and composition mechanisms enable code reuse without duplicating code.
Encapsulation:** Encapsulation ensures that the internal state of an object is hidden from the outside world.
Flexibility and Adaptability: OOD allows for changes and updates to the software to be made more easily
Scalability: OOD scales well for building large and complex software systems

8.What do you find lacking in your favorite development language? How do you cope with those gaps?
Cross-Platform Development:

- **Limitation:** Historically, C# was primarily associated with Windows development, which limited its cross-platform capabilities.
- **Coping:** Microsoft has made significant efforts to enhance cross-platform support through .NET Core (now .NET 5+ and .NET 6+). Developers can target multiple platforms, including Windows, macOS, and Linux, using .NET Core and .NET 5+.

. **Limited Language Features:**
- **Limitation:** C# might be perceived as having fewer language features compared to some newer languages.
- **Coping:** C# has been evolving with each new version, introducing features like pattern matching, async/await for asynchronous programming, nullable reference types, and more. Developers can leverage these features to write modern and expressive code.

9.What do classes and closures have in common?
In summary, classes and closures share common principles related to encapsulation, state, behavior, scope, and reuse, but they have different primary purposes and lifecycles, making them suitable for distinct programming scenarios.

10.When are anonymous functions useful?
Functional Programming: Anonymous functions are a fundamental component of functional programming languages and paradigms. They enable the use of higher-order functions like `map`, `filter`, and `reduce` to process collections of data in a concise and expressive manner.

anonymous functions are a versatile tool in programming that allows you to write more concise, expressive, and flexible code in various situations where defining a named function would be excessive or less readable.
Event Handlers: When working with user interfaces, anonymous functions are often used to define event handlers for UI elements. This allows you to keep event handling logic close to where it's used.

Dynamic Code Generation: In some cases, you may need to generate code dynamically based on runtime conditions. Anonymous functions can be used to define code blocks that are generated and executed as needed.

2. What’s a reverse proxy?

Interviewers may ask questions like this to gain a better sense of your knowledge.

This question can allow you to expand on related topics as well. For instance, after

explaining a reverse proxy, you also can discuss its opposite, which is the forward proxy.

Example: *"A reverse proxy acts as an intermediary, retrieving resources from a server and returning them to a client, so it appears that the information originates from the proxy server itself. The forward proxy, also an intermediary, is what the client puts between itself and another server."*

3. What's the difference between threads and processes?

Interviewers ask foundational knowledge questions like this to check your specific competence. You can use questions like these as an opportunity to show you have a practical understanding of key back-end terms.

Example: *"A process is an active program being executed, while a thread is a lightweight process that a scheduler can manage independently. Threads also make up processes. Since threads are quicker at context switching, an OS can stop one thread and run another much faster than stopping and starting a process."*

4. What steps would you take to use mysqldump to restore MySQL?

Expect interviewers to include some language-specific questions to check your coding skills. Answer in simple terms, outlining your approach to the programming question. If interviewers want technical specifics, they usually ask for them.

Example: *"First, I'd create a new database using MySQL and give it the same name as the lost database. I'd check to see if the database name was in the root directory, then determine whether I should include the server name as well."*

*5. If you have a limited amount of memory, how would you handle a large amount of data?**Example:** "I would break up the large amount of data into small chunks. I'd do this by using an external sort or merge sort. I think this would be the fastest and simplest option."*

Compression: Compress the data before loading it into memory

6. Define and explain these nine server response error codes: 200, 201, 204, 301, 400, 401, 404, 409 and 500.

Example: *“200 means “OK” and everything went well. A 201 “Created”" message means the system created a resource at the client's request. A 204 “No Content” code means the server didn't send back a status. A 301 “Moved Permanently” message means a client-triggered action changed the resource URI. A 400 “Bad Request” error refers to a client-side error.*

If the client doesn't provide the correct authentication, you see a 402 “Unauthorized” code. A 404 “Not Found” return means it didn’t find a mapped resource. An inconsistent or impossible state returns code 409, “Conflict.” Server-side errors generate code 500, described as, “"Internal Server Error."