Arquitetura Monolítica

Definição:

Uma aplicação monolítica é construída como **um único bloco de código unificado**, onde todas as funcionalidades e componentes estão interligados e executam no mesmo processo.

Características:

- Toda a aplicação é implantada como uma única unidade.
- Componentes compartilham o mesmo espaço de memória.
- Escalabilidade é feita na aplicação inteira, mesmo que apenas uma parte dela precise ser escalada.
- Mais simples de desenvolver e testar inicialmente.
- Dificuldade para manter e escalar à medida que o sistema cresce.

🗱 Arquitetura de Microserviços

Definição:

Microserviços se referem a um estilo arquitetural onde a aplicação é dividida em **pequenos serviços independentes**, que se comunicam entre si via APIs (geralmente HTTP/REST, gRPC, ou mensagens assíncronas).

Características:

- Cada microserviço é implantado independentemente.
- Cada serviço tem sua lógica e, muitas vezes, seu próprio banco de dados.
- Permite escalar apenas os serviços que realmente precisam.
- Requer uma infraestrutura mais complexa (orquestração, comunicação entre serviços, monitoramento).
- Facilita a adoção de tecnologias diferentes em serviços distintos.
- Maior resiliência: falhas em um serviço nem sempre afetam os outros.

Diferenças Diretas

Aspecto	Monolito	Microserviços
Estrutura de Código	Centralizada (único projeto)	Distribuída (vários projetos/serviços)
Escalabilidade	Global (a aplicação inteira)	Granular (por serviço)
Ciclo de Deploy	Único	Independente por serviço
Comunicação Interna	Direta (chamada de função/método)	Por rede (API REST, gRPC, mensagens)
Desenvolvimento	Mais simples no início	Mais complexo, mas escalável
Tolerância a Falhas	Menor (falha pode derrubar tudo)	Maior (falhas isoladas)

✓ Conclusão

Sim, os dois conceitos são fundamentalmente **opostos em abordagem**. O **monolito** centraliza tudo em uma única aplicação, enquanto os **microserviços** dividem a aplicação em partes autônomas e especializadas. A escolha entre eles depende de fatores como complexidade do sistema, equipe disponível, necessidade de escalabilidade e agilidade nos lançamentos.

Se quiser, posso te ajudar a identificar quando usar um ou outro — ou até como migrar de monolito para microserviços.

Você disse:

REST, grpc e RABBITMQ, kafka são APIs?

O ChatGPT disse:

Ótima pergunta — e a resposta é: **não exatamente.** Vamos entender com clareza o que são **REST, gRPC, RabbitMQ e Kafka** e qual é a relação deles com **APIs**:

O que é uma API?

API (Application Programming Interface) é um conjunto de regras e definições que permite que sistemas diferentes se comuniquem entre si. Ou seja, **API é um conceito**. É **a interface** usada para que aplicações conversem.

🚀 REST e gRPC são estilos de API

REST (Representational State Transfer)

- É um estilo arquitetural para construir APIs.
- Usa HTTP como protocolo de comunicação.
- As APIs REST normalmente trocam dados no formato JSON.
- Exemplo: GET /produtos/123

gRPC (Google Remote Procedure Call)

- É uma tecnologia para construir APIs de forma mais eficiente e performática.
- Usa Protocol Buffers (protobuf) em vez de JSON.
- Funciona sobre HTTP/2, o que permite chamadas assíncronas e streaming.
- Muito usado em sistemas distribuídos de alta performance.
- Exemplo: ProductService.GetProduct(id: 123)
- ☑ Conclusão: REST e gRPC são formas de implementar APIs.

RabbitMQ e Kafka são sistemas de mensageria (message brokers)

Eles **não são APIs por si só**, mas são **meios de comunicação entre serviços**, muitas vezes **usados em conjunto com APIs assíncronas**.

RabbitMQ

- Sistema de mensagens baseado em filas (message queue).
- Segue o padrão AMQP (Advanced Message Queuing Protocol).
- Um serviço publica mensagens em uma fila, e outro consome.
- Bom para tarefas assíncronas, como envio de e-mails ou processamento em background.

Apache Kafka

- Sistema de mensagens baseado em **streams** (fluxos de eventos).
- Alta performance e tolerância a falhas.
- Ideal para processamento de grandes volumes de dados em tempo real.

• Usado para event-driven architecture e log de eventos distribuído.

☑ Conclusão: RabbitMQ e Kafka são plataformas de mensageria, não APIs diretamente, mas usados para comunicação entre serviços — especialmente em arquiteturas de microserviços.

😂 Resumo final: É uma Tecnologia O que é? Tipo de comunicação Síncrona (cliente-REST Sim Estilo de API baseado em HTTP servidor) Framework para APIs rápidas com gRPC ✓ Sim Síncrona e assíncrona Protobuf Broker de mensagens baseado em RabbitMQ X Não Assíncrona filas Plataforma de mensagens baseada Kafka X Não Assíncrona

Se quiser, posso mostrar exemplos práticos de como esses meios são usados em uma arquitetura de microserviços.

em eventos

Você disse:

o que é mensageria são mensagens que enviam para o celular como SMS e push notification, não conheço o contexto do termo mensageria.

O ChatGPT disse:

Excelente observação — o termo "mensageria" realmente pode causar confusão, porque fora do contexto de computação ele costuma remeter a SMS, WhatsApp, push notifications e outras formas de mensagens para usuários finais.

Mas no **contexto de sistemas distribuídos e microserviços**, **mensageria** tem um significado bem diferente:

O que é Mensageria em Computação?

Mensageria é a comunicação entre sistemas (ou serviços) usando mensagens, geralmente de forma assíncrona, por meio de um sistema intermediário chamado broker de mensagens.

Ou seja:

- 👉 Um serviço envia uma mensagem para um canal/filha/stream
- 👉 Outro serviço lê essa mensagem quando estiver pronto

🧮 Exemplo simples:

Imagine dois microserviços:

- PedidoService: cria pedidos
- PagamentoService: processa pagamentos

Se o PedidoService quiser avisar o PagamentoService que um novo pedido foi criado, ele pode enviar uma mensagem como esta:

```
json
CopiarEditar
 "pedidold": 123,
 "valor": 89.90,
 "status": "criado"
}
```

Essa mensagem é **publicada em uma fila**, e o PagamentoService a consome e começa o processo de pagamento. Tudo isso sem que um precise chamar diretamente o outro.

🛠 Tecnologias que implementam mensageria:

- RabbitMQ: usa o padrão de filas.
- Apache Kafka: usa streams de eventos.
- ActiveMQ, Amazon SQS, Azure Service Bus: outras opções.
- 🚺 Mensageria x Mensagens para Usuários

Termo	Significado comum	Significado em computação
Mensageria	SMS, WhatsApp, Push	Comunicação entre sistemas via mensagens
Mensagem	Texto enviado para uma pessoa	Dados trocados entre serviços
Push Notification	Notificação para o usuário final	_
Broker de mensagens	_	Software que gerencia as mensagens entre sistemas

✓ Conclusão

Quando falamos de **mensageria em arquitetura de software**, estamos nos referindo a **troca de mensagens entre sistemas**, não mensagens para pessoas. É uma forma de integrar microserviços **sem que eles precisem ficar chamando diretamente uns aos outros**, melhorando a escalabilidade e a resiliência do sistema.

Se quiser, posso criar um diagrama simples ou simular uma troca de mensagens entre serviços usando RabbitMQ ou Kafka.

Você disse:

o que é o principio SOLID aplicado a microserviços?

O ChatGPT disse:

Letra Princípio

Excelente pergunta! Os **princípios SOLID**, originalmente formulados para **design orientado a objetos**, também podem ser aplicados — com algumas adaptações — ao **design de microserviços**. Eles ajudam a criar **serviços coesos**, **desacoplados**, **fáceis de manter e evoluir**.

Recapitulando: o que é SOLID?

SOLID é um acrônimo que representa cinco princípios fundamentais da engenharia de software:

S	Single Responsibility Principle	Princípio da Responsabilidade Única

Em português

Letra Princípio

Em português

0	Open/Closed Principle	Princípio Aberto/Fechado
L	Liskov Substitution Principle	Princípio da Substituição de Liskov
I	Interface Segregation Principle	Princípio da Segregação de Interfaces

D Dependency Inversion Principle Princípio da Inversão de Dependência

🗩 Como esses princípios se aplicam a microserviços?

1. S — Single Responsibility Principle (Responsabilidade Única)

No contexto de microserviços:

Cada microserviço deve ter **uma única responsabilidade clara** e **foco em um domínio específico do negócio**.

Exemplo:

Um microserviço PagamentoService **só cuida de pagamentos**, e não de pedidos ou notificações.

🦴 Benefício: Facilita manutenção, testes e escalabilidade.

• 2. O — Open/Closed Principle (Aberto para extensão, fechado para modificação)

No contexto de microserviços:

Você deve conseguir **adicionar novas funcionalidades criando novos serviços ou extensões**, **sem alterar o código existente de outros serviços**.

Exemplo:

Quer adicionar um serviço de cashback? Crie um CashbackService que consome eventos do PagamentoService — sem modificar o código original.

🦴 Benefício: Permite evolução do sistema com menor risco de quebra.

• 3. L — Liskov Substitution Principle

Originalmente:

Subtipos devem poder ser substituídos por seus tipos base sem alterar o comportamento do sistema.

Nos microserviços (adaptado):

Os contratos de comunicação entre serviços (como APIs ou eventos) **devem ser estáveis e confiáveis**. Um novo serviço que substitui outro **deve respeitar o mesmo contrato**.

Exemplo:

Se um serviço novo expõe a mesma API de um antigo, os consumidores não devem perceber diferença no comportamento funcional.

Benefício: Mantém compatibilidade entre serviços e evita que mudanças quebrem consumidores.

◆ 4. I — Interface Segregation Principle

Nos microserviços:

Evite serviços com interfaces **grandes e genéricas**. Prefira APIs **focadas e específicas** para cada tipo de consumidor.

Exemplo:

Em vez de um único endpoint /cliente/detalhes que retorna tudo (endereços, compras, saldo, notificações), crie endpoints separados e especializados: /cliente/compras, /cliente/saldo, etc.

🦴 Benefício: Facilita a evolução das APIs e reduz dependências desnecessárias.

• 5. D — Dependency Inversion Principle

Nos microserviços:

Serviços não devem depender diretamente de implementações concretas de outros serviços, mas sim de **contratos estáveis (APIs, eventos)**.

Exemplo:

PedidoService não deve conhecer diretamente a lógica do PagamentoService. Ele apenas **publica um evento**, como PedidoCriado, e o PagamentoService reage a esse evento.

Benefício: Reduz o acoplamento e facilita substituições ou atualizações de serviços.

Conclusão

Aplicar SOLID aos microserviços significa:

- Criar serviços pequenos, coesos e focados.
- Separar responsabilidades de forma clara.
- Minimizar acoplamentos entre serviços.
- Evoluir sistemas com segurança, sem quebrar dependências.

Esses princípios ajudam a manter a arquitetura **sustentável, escalável e resiliente** — especialmente em ambientes com muitos serviços e times diferentes.

Se quiser, posso te mostrar um exemplo real (ou com código) de como esses princípios se manifestam em um sistema de microserviços.

Você disse:

o que é polyglot persistent?

O ChatGPT disse:

Polyglot Persistence (ou **Persistência Poliglota**) é um conceito da arquitetura de software que significa:

Usar diferentes tipos de bancos de dados (SQL, NoSQL, grafos, tempo real etc.) **dentro de uma mesma aplicação ou sistema**, de acordo com a necessidade específica de cada parte do sistema.

Por que usar Polyglot Persistence?

Cada tipo de banco de dados tem pontos fortes e fracos. A ideia é **usar o banco certo para o problema certo**, em vez de forçar tudo a funcionar com um único tipo.

© Exemplo prático:

Imagine um sistema de e-commerce com os seguintes requisitos:

Componente	Requisito principal	Banco ideal
Produtos	Consultas complexas com filtros	Banco relacional (SQL)
Carrinho de compras	Gravação e leitura rápida de sessões	Redis ou MongoDB (NoSQL)

Componente	Requisito principal	Banco ideal
Recomendações de produtos	Relações entre itens e usuários	Banco de grafos (Neo4j)
Histórico de pedidos	Registro imutável em ordem cronológica	Banco de eventos ou logs (Kafka, Cassandra)
Log de erros e métricas	Escrita intensiva com leitura analítica	Elasticsearch ou InfluxDB

Esse sistema usa **vários bancos diferentes**, cada um especializado em um tipo de dado ou acesso — isso é **Polyglot Persistence**.

Name Quando vale a pena usar?

- Em **sistemas complexos e grandes** (como microserviços), com requisitos variados.
- Quando há diferentes tipos de dados e acessos (relacionais, documentos, gráficos, etc.).
- Quando a performance, escalabilidade ou manutenibilidade justifica a complexidade adicional.

⚠ Cuidados ao usar Polyglot Persistence

- Aumenta a complexidade da arquitetura (mais bancos = mais infraestrutura).
- Pode dificultar a manutenção e treinamento da equipe.
- Requer cuidado com consistência de dados entre bancos diferentes.

Conclusão

Polyglot Persistence não é regra, mas uma estratégia.

Ela permite que você **escolha o melhor banco para cada tipo de necessidade** dentro de um sistema.

É especialmente útil em **microserviços**, onde cada serviço pode usar sua própria tecnologia de persistência, sem impactar os outros.

Se quiser, posso te mostrar um diagrama ou exemplo de arquitetura real usando essa abordagem.

Você disse:

quais perguntas podem cair num concurso publico sobre jQuery

O ChatGPT disse:

Ótima pergunta! Embora o uso de **jQuery** tenha diminuído com o avanço de frameworks modernos como React e Vue, **muitos concursos públicos ainda cobram jQuery**, principalmente em provas de áreas como **TI, desenvolvimento web e analista de sistemas** — especialmente quando o edital menciona explicitamente **HTML**, **JavaScript e jQuery**.

Tipos de Perguntas sobre jQuery em Concurso Público

Geralmente, as questões caem nas seguintes categorias:

1. Sintaxe básica e seleção de elementos

Exemplo de pergunta:

Qual das alternativas abaixo seleciona todos os elementos em uma página HTML usando jQuery?

- a) document.querySelectorAll("p")
- b) \$("p")
- c) getElementById("p")
- d) \$("#p")
- e) \$.get("p")
- **Gabarito:** b) \$("p")

2. Eventos (event handling)

Exemplo:

Qual método do jQuery é usado para associar uma função a um clique em um botão com id btnSalvar?

- a) \$("#btnSalvar").bind("click", func)
- b) \$("#btnSalvar").click(func)
- c) \$("#btnSalvar").on("click", func)

- d) Todas as alternativas acima
- e) Nenhuma das anteriores
- ☑ **Gabarito:** d) Todas as alternativas acima (todas funcionam, mas on() é o mais moderno)

3. Manipulação de DOM (HTML e atributos)

Exemplo:

Qual método jQuery altera o texto de um elemento com id titulo?

- a) \$("#titulo").html("Novo Título")
- b) \$("#titulo").text("Novo Título")
- c) \$("#titulo").val("Novo Título")
- d) a e b estão corretas
- e) Todas estão corretas
- Gabarito: d) a e b estão corretas (depende se é para texto ou HTML)

4. Efeitos e animações

Exemplo:

Para esconder um elemento com id #menu, qual função é correta?

- a) \$("#menu").hide()
- b) \$("#menu").collapse()
- c) \$("#menu").invisible()
- d) \$("#menu").hidden()
- e) \$("#menu").display("none")
- Gabarito: a) \$("#menu").hide()

5. AJAX com jQuery

Exemplo:

Qual método jQuery envia uma requisição HTTP GET de forma assíncrona?

- a) \$.get()
- b) \$.ajax({ type: "POST" })
- c) fetch()

- d) \$.send()
- e) \$.request()
- Gabarito: a) \$.get()

6. Encadeamento de métodos (chaining)

Exemplo:

O que o seguinte código faz?

\$("#msg").css("color", "red").slideUp(2000).slideDown(2000);

- a) Aplica a cor vermelha e remove o elemento.
- b) Aplica a cor vermelha e faz o texto sumir e reaparecer.
- c) Apenas muda a cor.
- d) Só executa o último método.
- e) Gera erro.
- Gabarito: b) Aplica a cor vermelha e faz o texto sumir e reaparecer.

Dicas para estudar jQuery para concurso:

- Saiba selecionar elementos (\$("#id"), \$(".classe"), \$("tag"))
- Pratique eventos: .click(), .on(), .hover()
- Entenda manipulação de conteúdo: .text(), .html(), .val()
- Saiba aplicar efeitos: .hide(), .show(), .fadeIn(), .slideUp()
- Estude o básico de AJAX com jQuery: \$.get(), \$.post(), \$.ajax()

Se você me disser qual concurso está estudando (ou a banca), posso montar um **simulado com questões específicas sobre jQuery** no estilo daquela prova. Deseja isso?

Você disse:

o que pode ser uma pegadinha no jquery

O ChatGPT disse:

Ótima pergunta! Provas de concurso costumam usar **pegadinhas sutis** para testar o domínio real do conteúdo, e com **jQuery**, não é diferente. Aqui estão as **pegadinhas mais comuns em jQuery**:

1. Confundir seletores de jQuery com CSS/JavaScript puro

Exemplo:

Qual das opções seleciona o elemento com ID "menu"?

- a) \$("#menu")
- b) \$("menu")
- c) document.getElementById("menu")
- d) \$(".menu")
- Gabarito: a)

Pegadinha: opção b) parece correta, mas "menu" sem # seleciona elementos <menu>, não o ID.

1 2. Confundir text() com val()

Exemplo:

O que faz \$("#nome").val("João")?

- a) Altera o texto de um parágrafo
- b) Muda o valor de um campo de formulário
- c) Aplica uma classe "João"
- d) Substitui o HTML interno
- ✓ Gabarito: b)

Pegadinha: val() é usado para inputs, text() é para texto visível em elementos como , <div>, etc.

3. Erros sutis de sintaxe

Exemplo:

Qual das opções abaixo está correta para alterar a cor do texto para azul?

- a) \$("#titulo").style.color = "blue";
- b) \$("#titulo").css("color", "blue");
- c) \$("#titulo").setStyle("color", "blue");
- d) \$("#titulo").attr("color", "blue");

Gabarito: b)

Pegadinha: a) parece válida para quem pensa em JavaScript puro, mas style não é usado diretamente com jQuery.

1 4. Encadeamento incorreto de métodos

Exemplo:

O que acontece com:

\$("#msg").hide().text("Olá");

- a) O elemento é ocultado e depois recebe o texto "Olá"
- b) O elemento recebe o texto e depois é ocultado
- c) O elemento aparece com o texto novo
- d) O elemento continua visível
- **Gabarito:** a)

Pegadinha: O método .text() ainda é executado, mesmo com o elemento escondido. A ordem importa.

1 5. Uso incorreto de eventos

Exemplo:

Qual código executa uma ação ao clicar em um botão?

- a) \$("#botao").onClick(function() {...})
- b) \$("#botao").onclick = function() {...}
- c) \$("#botao").click(function() {...})
- d) click("#botao", function() {...})

✓ Gabarito: c)

Pegadinha: Opção a) parece correta, mas onClick não é válido no jQuery — o correto é .click() ou .on("click", ...).

1 6. Não diferenciar métodos semelhantes

Método jQuery O que faz

- .html() Define ou retorna o HTML interno
- .text() Define ou retorna o texto visível

Método jQuery O que faz

.val() Define ou retorna o valor de inputs

.attr() Lida com atributos HTML

.prop() Lida com propriedades (ex: checked)

Pegadinhas comuns:

- Usar .attr("checked") em vez de .prop("checked") para checkboxes.
- Achar que .text() altera valores de campos de formulário (não altera!).

Dicas para evitar pegadinhas:

- Leia com atenção os seletores: # é para ID, . para classe.
- Memorize diferenças entre métodos parecidos: .text() vs .val(), .attr() vs .prop().
- Lembre-se de que sintaxe misturada com JavaScript puro geralmente está errada.
- Teste na prática comandos simples usando o console do navegador.

Se quiser, posso te montar um **mini simulado só com pegadinhas de jQuery** para treinar. Deseja isso?

Você disse:

pode me perguntar pra ver se consigo ir bem

O ChatGPT disse:

Claro! Vamos testar seus conhecimentos com **5 perguntas de múltipla escolha** sobre **jQuery**, com foco em conceitos básicos, pegadinhas e prática real de concursos.

Questão 1 – Seletores

Qual das opções seleciona todos os elementos com a classe "ativo"?

- a) \$("#ativo")
- b) \$(".ativo")

- c) \$("ativo")
- d) \$(.ativo)

Questão 2 – Manipulação de conteúdo

O que faz o seguinte código?

javascript

CopiarEditar

\$("#mensagem").html("Olá!");

- a) Define o texto "Olá!" em negrito no elemento com id mensagem
- b) Define o valor de um campo com id mensagem
- c) Define o conteúdo HTML interno de todos os elementos com classe mensagem
- d) Gera um erro de sintaxe

✓ Questão 3 – Eventos

Qual das opções associa uma função ao clique em um botão com id btnSalvar?

- a) \$("#btnSalvar").click(function() { ... });
- b) \$("#btnSalvar").on("click", function() { ... });
- c) \$("#btnSalvar").bind("click", function() { ... });
- d) Todas as anteriores

Questão 4 – Efeitos

O que faz o método fadeOut() em jQuery?

- a) Oculta o elemento instantaneamente
- b) Oculta o elemento com uma animação de desvanecimento
- c) Remove o elemento da árvore DOM
- d) Aplica opacidade 100%

Questão 5 – Pegadinha de .val() x .text()

Qual comando ¡Query define o texto visível dentro de um <div>?

- a) \$("#caixa").val("Novo texto");
- b) \$("#caixa").text("Novo texto");

- c) \$("#caixa").attr("text", "Novo texto");
- d) \$("#caixa").html("Novo texto");
- e) b e d estão corretas

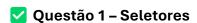
Responda quando quiser, e eu corrijo logo em seguida com explicações. Vamos ver como você se sai!

Você disse:

b

O ChatGPT disse:

Ótimo! Vamos corrigir questão por questão e ver como você foi 👇



Sua resposta: b) \$(".ativo")
Gabarito: ✓ b) \$(".ativo")

& Correto!

O seletor com . em jQuery seleciona **todos os elementos com a classe especificada**. Muito bem!

✓ Questão 2 – Manipulação de conteúdo

Sua resposta: b

Gabarito correto: X a) Define o texto "Olá!" em negrito no elemento com id mensagem

🔍 Explicação:

O método .html() **define o conteúdo HTML interno** de um elemento. Então ele **interpreta** a tag , deixando o texto "Olá!" em **negrito**.

Se fosse .text("Olá!"), aí sim apareceria literalmente "Olá!" como texto.

Questão 3 – Eventos

Sua resposta: (não foi enviada)

Quer tentar essa agora?

Pergunta:

Qual das opções associa uma função ao clique em um botão com id btnSalvar?

- a) \$("#btnSalvar").click(function() { ... });
- b) \$("#btnSalvar").on("click", function() { ... });
- c) \$("#btnSalvar").bind("click", function() { ... });
- d) Todas as anteriores

Questão 4 – Efeitos

Sua resposta: (não foi enviada)

Tente essa também:

Pergunta:

O que faz o método fadeOut() em jQuery?

- a) Oculta o elemento instantaneamente
- b) Oculta o elemento com uma animação de desvanecimento
- c) Remove o elemento da árvore DOM
- d) Aplica opacidade 100%

✓ Questão 5 – Pegadinha de .val() x .text()

Sua resposta: b)

Gabarito: 🗸 e) b e d estão corretas

Explicação:

Você disse:

desculpa pode pergunta uma por vez

O ChatGPT disse:

Claro! Vamos com calma, uma pergunta de cada vez. Aqui vai a primeira:

✓ Pergunta 1 – Seletores (básico)

Qual das opções seleciona todos os elementos com a classe "ativo"?

- a) \$("#ativo")
- b) \$(".ativo")

c) \$("ativo") d) \$(.ativo)
Qual sua resposta?
Você disse:
b
O ChatGPT disse:
✓ Correto!
A opção b) \$(".ativo") é a certa. Esse seletor jQuery retorna todos os elementos que têm a classe ativo — exatamente como o CSS.
🌀 Boa! Vamos para a próxima?
✓ Pergunta 2 – Manipulação de conteúdo
O que faz o seguinte código?
javascript
CopiarEditar
\$("#mensagem").html(" Olá! ");
 a) Define o texto "Olá!" em negrito no elemento com id mensagem b) Define o valor de um campo com id mensagem c) Define o conteúdo HTML interno de todos os elementos com classe mensagem d) Gera um erro de sintaxe
Qual sua resposta?
Você disse:
a
O ChatGPT disse:
✓ Correto de novo!
A opção a) Define o texto "Olá!" em negrito no elemento com id mensagem está certa .

RIL	Evalia	<u>~</u> -
4	Explic	açau.

O método .html() **insere HTML dentro do elemento**, então a tag é interpretada, deixando "Olá!" em **negrito**.

🤲 Mandou bem! Vamos para a próxima?

✓ Pergunta 3 – Eventos

Qual das opções associa uma função ao clique em um botão com id btnSalvar?

- a) \$("#btnSalvar").click(function() { ... });
- b) \$("#btnSalvar").on("click", function() { ... });
- c) \$("#btnSalvar").bind("click", function() { ... });
- d) Todas as anteriores

Qual sua resposta?

Você disse:

а

O ChatGPT disse:

Quase!

A sua resposta a) \$("#btnSalvar").click(function() { ... }); está correta, mas não é a mais completa.

Gabarito oficial: d) Todas as anteriores

Explicação:

Todas as opções funcionam para adicionar um evento de clique:

- a) .click() método simplificado, funciona bem.
- **b)** .on("click", ...) forma moderna e recomendada, mais flexível (funciona com eventos dinâmicos).
- **c)** .bind("click", ...) mais antiga, ainda compatível, mas menos usada hoje.
- A alternativa **d)** diz "todas as anteriores", e como todas funcionam, ela é a correta em uma prova.

✓ Pergunta 4 – Efeitos visuais
O que faz o método fadeOut() em jQuery?
 a) Oculta o elemento instantaneamente b) Oculta o elemento com uma animação de desvanecimento c) Remove o elemento da árvore DOM d) Aplica opacidade 100%
Qual a sua resposta?
Você disse:
b
O ChatGPT disse:
✓ Perfeito!
A resposta b) Oculta o elemento com uma animação de desvanecimento está certíssima .
Explicação: O método .fadeOut() reduz gradualmente a opacidade do elemento até 0, e então o esconde (como se fosse um .hide() com transição suave).
🎉 Muito bem! Vamos para a última da rodada:
Pergunta 5 – .text() vs .val() (pegadinha clássica)
Qual comando jQuery define o texto visível dentro de um <div></div> ?
a) \$("#caixa").val("Novo texto"); b) \$("#caixa").text("Novo texto"); c) \$("#caixa").attr("text", "Novo texto"); d) \$("#caixa").html("Novo texto"); e) b e d estão corretas
Qual a sua resposta?
Você disse:
e
O ChatGPT disse:

- Resposta correta: e) b e d estão corretas!
- **Muito bem!** Essa é uma pegadinha clássica que confunde muita gente.

Explicação:

- b) .text("Novo texto") → Insere apenas texto puro, sem interpretar HTML.
- d) .html("Novo texto") → Insere o conteúdo como HTML, podendo interpretar tags.
- Ambos funcionam para <div>, dependendo do que você quer:
 - Se for texto simples → .text()
 - Se for texto com formatação HTML → .html()

XX Resultado final:

Você acertou **4 de 5**, com observações técnicas boas! E na que errou, **entendeu o motivo e estava parcialmente certo**.

🥙 Parabéns! Está bem preparado para questões de jQuery em concursos.