DOMINANDO O JAVASCRIPT EM 1 ANO

Caderno de Exercícios

Exercícios teóricos e práticos do curso

Node.js e navegador · 9 módulos · 77 exercícios · 1 desafio integrador

Autor Desconhecido · theknowledgeisle.com

Nome do aluno: _____ Início: _____ Término: _____

Como usar este caderno

Cada módulo reúne um bloco de assuntos do curso, com a lista dos artigos do site que tratam deles. Os exercícios são **teóricos** (explique com as suas palavras), **práticos** (escreva e rode o código, no Node.js ou no navegador) ou **mistos**.

Tente sempre resolver sozinho antes de abrir o **Caderno de Respostas**. É no erro que o conhecimento fixa.

Sumário

Módulo	Tema	Artigos	Exercícios
MÓDULO 1	Fundamentos do JavaScript	#01–#06, #11–#13, #15	1.1–1.11
MÓDULO 2	JavaScript Moderno (ES6+)	#07–#09, #18–#22, #27	2.1–2.10
MÓDULO 3	Node.js e Back-end	#30–#35	3.1–3.8
MÓDULO 4	APIs Avançadas	#33–#36	4.1–4.8
MÓDULO 5	Qualidade de Código	#36–#40	5.1–5.8
MÓDULO 6	React e Front-end Moderno	#41–#45	6.1–6.8
MÓDULO 7	Deploy e Infraestrutura	#46–#49	7.1–7.7
MÓDULO 8	Arquitetura e Padrões	#50–#52	8.1–8.8
MÓDULO 9	Tópicos Avançados	#53–#55	9.1–9.9
DESAFIOS	Projetos integradores	todos	D1

Legenda: os exercícios são numerados por módulo (**3.2** é o segundo do módulo 3). As linhas pautadas são para a sua resposta; nos práticos, use também o computador: **todo programa deve ser executado**.

MÓDULO 1

Fundamentos do JavaScript

Variáveis, tipos, operadores, controle de fluxo, funções, arrays, objetos, DOM e eventos. A base sobre a qual todo o resto é construído.

Artigos do site

#01 O que é JavaScript e por que aprender?
#02 Condicionais: if, else e switch
#03 Laços de Repetição: for, while e do...while
#04 Funções: declaração, expressão e arrow functions
#05 Arrays: criando e manipulando listas
#06 Objetos: estruturando dados do mundo real
#11 O que é o DOM e como o JavaScript interage com o HTML
#12 Eventos: click, input, submit e muito mais
#13 Criando e Removendo Elementos Dinamicamente
#15 LocalStorage e SessionStorage

Variáveis, Tipos e Controle de Fluxo

1.1 TEÓRICO

Explique a diferença entre var, let e const. Em quais situações você usaria cada um? Por que var é considerado problemático no JavaScript moderno?

1.2 TEÓRICO

O que é coerção de tipos em JavaScript? Explique a diferença entre == e === com dois exemplos práticos que ilustrem situações onde o resultado seria diferente.

1.3 PRÁTICO

Escreva uma função chamada classificarIdade(idade) que receba um número e retorne:

- Se idade < 0: "Valor inválido"
- Se 0 ≤ idade < 13: "Criança"
- Se 13 ≤ idade < 18: "Adolescente"
- Se 18 ≤ idade < 60: "Adulto"
- Se idade ≥ 60: "Idoso"

Dica: Use if/else if ou switch. Teste com os valores: -1, 5, 15, 30, 65.

1.4 PRÁTICO

(a) Imprima os números ímpares de 1 a 20 com `for` e depois com `while`, sem testar número por número. (b) Sem executar, diga o que o código abaixo imprime; depois rode, explique o resultado e corrija trocando uma única palavra.

```
for (var i = 0; i < 3; i++) setTimeout(() => console.log(i));
```

Dica: qual é o escopo de `var`, e quando os callbacks do `setTimeout` rodam?

Funções, Arrays e Objetos

1.5 MISTO

Explique o que é closure e para que serve. Depois implemente uma função criarContador() que retorna um objeto com três métodos: incrementar(), decrementar() e valor(). Cada chamada deve manter o estado interno sem usar variáveis globais.

1.6 PRÁTICO

Dado o array de produtos abaixo, use os métodos de array para:

- Filtrar apenas os produtos com estoque > 0 (filter)
- Obter um array com apenas os nomes dos produtos disponíveis (map)
- Calcular o valor total do estoque disponível — preço × estoque (reduce)
- Encontrar o produto mais caro (reduce ou sort)

Dica: const produtos = [{nome:"Teclado",preco:250,estoque:5},{nome:"Mouse",preco:80,estoque:0}, {nome:"Monitor",preco:1200,estoque:3},{nome:"Webcam",preco:300,estoque:0}]

1.7 PRÁTICO

Crie um objeto livro com as propriedades: titulo, autor, anoPublicacao, paginas e disponivel. Depois escreva uma função formatarLivro(livro) que retorne uma string formatada: "Titulo (Ano) - Autor | X páginas | [Disponível/Indisponível]". Use destructuring na função.

DOM e Eventos

1.8 TEÓRICO

O que é o DOM (Document Object Model)? Explique a diferença entre getElementById, querySelector e querySelectorAll. Quando você usaria cada um?

1.9 PRÁTICO

Escreva o código JavaScript para criar dinamicamente uma lista <ul> com 5 itens de sua escolha e adicioná-la ao body da página. Use createElement e appendChild. Cada item deve ter um evento de clique que exibe um alert com o texto do item clicado.

Dica: Você pode testar no console do navegador.

1.10 MISTO

Explique o que é event delegation e por que ela é mais eficiente do que adicionar um listener em cada elemento de uma lista. Depois reescreva o código do exercício 1.9 usando event delegation — um único listener no <ul> em vez de um em cada <li>.

Projeto: To-do List

1.11 PRÁTICO

Implemente uma to-do list funcional no navegador (HTML + JS). Deve ter:

- Campo de texto + botão para adicionar tarefas
- Cada tarefa exibida com checkbox e botão de remover
- Clicar no checkbox risca o texto da tarefa (classList.toggle)
- Contador de tarefas pendentes atualizado em tempo real
- Botão "Limpar concluídas" que remove todas as marcadas como feitas

MÓDULO 2

JavaScript Moderno (ES6+)

Arrow functions, destructuring, spread/rest, Promises, async/await, Fetch API, módulos ES e classes. O JavaScript como é escrito profissionalmente hoje.

Artigos do site

#07 Desestruturação, Spread e Rest Operator
#08 Escopo, Hoisting e Closures
#09 Tratamento de Erros com try, catch e finally
#18 O que é Programação Assíncrona? O Event Loop Explicado
#19 Callbacks: o começo de tudo
#20 Promises: resolvendo o Callback Hell
#21 Async/Await: escrevendo código assíncrono de forma limpa
#22 Fetch API: consumindo dados da internet
#27 Tratamento de erros em requisições HTTP

ES6+ Sintaxe Moderna

2.1 PRÁTICO

Reescreva as funções abaixo usando arrow functions. Depois explique em qual situação arrow functions NÃO devem ser usadas (dica: relacionado ao this).

- function dobrar(n) { return n * 2; }
- function saudar(nome) { return "Olá, " + nome + "!"; }
- function soma(a, b) { return a + b; }

2.2 PRÁTICO

Dado o objeto usuario abaixo, use destructuring para extrair as propriedades em variáveis separadas. Depois use spread para criar um novo objeto com a propriedade ativo: true adicionada, sem modificar o original.

Dica: const usuario = { id: 1, nome: "Ana", email: "ana@email.com", cidade: "São Paulo" }

2.3 PRÁTICO

Escreva uma função somaTudo(...numeros) que aceite qualquer quantidade de argumentos numéricos e retorne a soma de todos. Use o parâmetro rest. Teste com: somaTudo(1,2,3), somaTudo(10,20,30,40,50), somaTudo().

Dica: somaTudo() sem argumentos deve retornar 0.

Promises, Async/Await e Fetch

2.4 TEÓRICO

Explique o que é o Event Loop do JavaScript e por que a linguagem é "single-threaded mas não bloqueante". O que significa uma Promise estar em estado "pending", "fulfilled" ou "rejected"?

2.5 PRÁTICO

Escreva uma função aguardar(ms) que retorna uma Promise que resolve após o número de milissegundos informado. Depois use async/await para escrever uma sequência que: aguarda 1s, loga "passo 1", aguarda 500ms, loga "passo 2", aguarda 200ms, loga "passo 3".

Dica: Use new Promise(resolve => setTimeout(resolve, ms)).

2.6 PRÁTICO

Use a Fetch API para consumir o endpoint público https://jsonplaceholder.typicode.com/users e:

- Exibir no console o nome e email de cada usuário
- Tratar erros com try/catch
- Adicionar um timeout de 5 segundos (use Promise.race com uma promise de timeout)
- Exibir uma mensagem "Carregando..." antes da requisição e removê-la após

Dica: jsonplaceholder.typicode.com é uma API pública gratuita para testes.

Módulos e Classes

2.7 TEÓRICO

Explique a diferença entre export default e export nomeado. Quando você usaria cada um? Como você importaria um módulo que tem tanto um export default quanto exports nomeados?

2.8 PRÁTICO

Crie uma classe ContaBancaria com:

- Propriedade privada #saldo inicializada no constructor
- Método depositar(valor) que valida valor > 0
- Método sacar(valor) que valida valor > 0 e saldo suficiente
- Getter saldo que retorna o saldo atual
- Método extrato() que retorna um array com o histórico de transações
- Método estático criarContaZerada() que retorna uma nova conta com saldo 0

Tratamento de Erros

2.9 MISTO

Explique a diferença entre erros síncronos e assíncronos em JavaScript. Por que um try/catch comum não captura erros dentro de callbacks assíncronos sem await? Depois escreva um exemplo demonstrando o comportamento correto com async/await e o comportamento problemático com callbacks.

2.10 PRÁTICO

Crie classes de erro customizadas: ErroValidacao, ErroNaoEncontrado e ErroAutenticacao, todas estendendo Error. Cada uma deve ter: mensagem, código HTTP correspondente (422, 404, 401) e o nome da classe como propriedade name. Escreva uma função tratarErro(erro) que identifica o tipo e retorna um objeto { status, mensagem } adequado.

MÓDULO 3

Node.js e Back-end

Node.js, NPM, Express, middlewares, JWT, MongoDB, Mongoose e API REST completa. Construindo servidores profissionais.

Artigos do site

#30 Introdução ao Node.js: JavaScript fora do navegador
#31 NPM: gerenciando pacotes e dependências
#32 Criando um servidor HTTP com Node.js puro
#33 Express.js: o framework web do Node
#34 MongoDB e Mongoose: banco de dados com Node
#35 Projeto: API REST com autenticação JWT

Node.js e NPM

3.1 TEÓRICO

Explique as diferenças entre require() (CommonJS) e import/export (ESModules) no Node.js. Quando você usaria cada um? O que é necessário configurar no package.json para usar ESModules?

3.2 PRÁTICO

Escreva um script Node.js que:

- Leia um arquivo chamado dados.json do disco
- Adicione um novo objeto com id, nome e data de criação ao array existente
- Salve o arquivo atualizado no disco
- Lide com erros: arquivo inexistente, JSON malformado, sem permissão de escrita

Dica: Use fs.promises (a versão assíncrona com async/await).

Express e Middlewares

3.3 TEÓRICO

O que é um middleware no Express? Explique o ciclo req → middleware → middleware → rota → res. O que acontece se um middleware não chamar next()? Qual a diferença entre app.use() e router.use()?

3.4 PRÁTICO

Implemente os seguintes middlewares do zero (sem bibliotecas):

- logger: registra método HTTP, URL, status e tempo de resposta
- autenticar: verifica se há um header Authorization com o valor "Bearer token-secreto"
- limitarTamanho: rejeita requisições com body maior que 1KB
- naoEncontrado: responde com 404 JSON para rotas não definidas

3.5 PRÁTICO

Implemente autenticação JWT do zero. Crie os endpoints:

- POST /auth/registrar — recebe nome, email, senha; salva em array em memória; retorna JWT
- POST /auth/login — verifica credenciais; retorna JWT
- GET /auth/perfil — rota protegida; retorna dados do usuário a partir do token
- Use bcryptjs para senha e jsonwebtoken para o token. Não use banco de dados neste exercício.

MongoDB e API REST

3.6 TEÓRICO

Explique a diferença entre banco de dados relacional (SQL) e não-relacional (NoSQL) como MongoDB. Quais são as vantagens e desvantagens de cada abordagem? Em que tipo de projeto você escolheria um em vez do outro?

3.7 PRÁTICO

Crie um schema Mongoose para Produto com:

- nome (string, obrigatório, min 2 chars)
- preco (number, obrigatório, min 0)
- categoria (enum: ["eletronico","roupa","alimento","outro"])
- estoque (number, default 0)
- ativo (boolean, default true)
- criadoEm (Date, default Date.now)
- Um método de instância resumo() que retorna "Nome — R$ preço (estoque un.)"
- Um índice composto em categoria + ativo

3.8 PRÁTICO

Implemente um CRUD completo em Express + Mongoose para o modelo Produto do exercício anterior:

- GET /produtos — listar todos com filtro opcional por ?categoria=
- GET /produtos/:id — buscar por ID com tratamento de ID inválido
- POST /produtos — criar com validação
- PUT /produtos/:id — atualizar parcialmente com $set
- DELETE /produtos/:id — remover com confirmação no retorno

MÓDULO 4

APIs Avançadas

Relacionamentos, validação com Zod, upload de arquivos, paginação, testes com Jest/Supertest e documentação com Swagger.

Artigos do site

#33 Express.js: o framework web do Node
#34 MongoDB e Mongoose: banco de dados com Node
#35 Projeto: API REST com autenticação JWT
#36 Testes automatizados com Jest

Relacionamentos e Validação

4.1 TEÓRICO

Explique as três formas de modelar relacionamentos no MongoDB: embedding (documentos aninhados), referência (populate) e arrays de IDs. Quando você usaria cada abordagem? Dê um exemplo prático para cada.

4.2 PRÁTICO

Crie um schema Mongoose para Pedido que referencia Usuario e contém um array de itens com referência a Produto. O pedido deve ter status (enum), total calculado e método calcularTotal() que soma preco × quantidade de cada item. Implemente uma rota GET /pedidos/:id que popula usuario e produtos dos itens.

Dica: Use .populate("usuario", "nome email") para trazer apenas os campos necessários.

4.3 PRÁTICO

Usando a biblioteca Zod, crie schemas de validação para:

- Cadastro de usuário: nome (min 2), email (válido), senha (min 8, com número e maiúscula), confirmarSenha (igual à senha)
- Produto: nome, preco (positivo), categoria (enum), estoque (inteiro >= 0)
- Um middleware validar(schema) genérico que pode ser usado em qualquer rota
- Teste o middleware na rota POST /usuarios

Upload e Paginação

4.4 PRÁTICO

Configure Multer para aceitar upload de imagens com:

- Tipos aceitos: jpeg, jpg, png, webp
- Tamanho máximo: 5MB
- Renomear o arquivo para: timestamp-nomeoriginal
- Rota POST /produtos/:id/imagem que atualiza o campo imagemUrl do produto
- Tratamento de erro quando o tipo de arquivo não é aceito

4.5 MISTO

Explique o problema de performance do uso de skip/limit em coleções muito grandes no MongoDB. Depois implemente paginação baseada em cursor para a rota GET /produtos usando o campo _id como cursor: a resposta deve incluir os dados, um campo nextCursor e hasMore.

Dica: Cursor: produtos com _id > lastId, limitado por pageSize.

Testes e Documentação

4.6 TEÓRICO

Explique a diferença entre testes unitários, de integração e end-to-end (E2E). Quais são as vantagens e limitações de cada nível? O que é a "pirâmide de testes" e por que ela recomenda mais testes unitários?

4.7 PRÁTICO

Escreva testes com Jest para uma função calculadoraFrete(peso, distancia, expresso) que retorna o custo do frete. Cubra:

- Frete padrão: R$ 2,50/kg + R$ 0,01/km
- Frete expresso: 3× o padrão, mínimo R$ 15,00
- Retirada em loja: sempre R$ 0,00
- Erros: peso negativo, distância negativa, tipo inválido
- Use describe para agrupar, it para cada caso, e toBeCloseTo para floats

4.8 PRÁTICO

Escreva testes de integração com Supertest para as rotas de Produto. Cubra ao menos:

- POST /produtos com dados válidos — espera 201
- POST /produtos com dados inválidos — espera 422 com mensagem de erro
- GET /produtos/:id com ID inexistente — espera 404
- DELETE /produtos/:id de outro usuário — espera 403
- Use beforeAll/afterAll para conectar/desconectar o banco de teste

MÓDULO 5

Qualidade de Código

TypeScript, ESLint, Prettier, Git avançado, commits semânticos e TDD. Código que o time consegue manter.

Artigos do site

#36 Testes automatizados com Jest
#37 ESLint e Prettier: código limpo e padronizado
#38 Git avançado e fluxo de trabalho em equipe
#39 Introdução ao TypeScript
#40 Revisão + Projeto: API tipada e testada

TypeScript

5.1 TEÓRICO

Explique a diferença entre interface e type em TypeScript. Em quais situações cada um é mais adequado? O que são union types e intersection types? Dê um exemplo prático de cada.

5.2 PRÁTICO

Crie os seguintes tipos TypeScript para a API de tarefas:

- Interface Tarefa com: id, titulo, descricao?, status (union type), prioridade (enum), prazo?, usuarioId, criadoEm
- Type TarefaInput = Omit<Tarefa, "id" | "criadoEm">
- Type TarefaAtualizar = Partial<TarefaInput>
- Interface PaginaResultado<T> genérica com dados, total, pagina, totalPaginas
- Função tipada buscarTarefas(filtros: Partial<Tarefa>): Promise<PaginaResultado<Tarefa>>

5.3 PRÁTICO

Implemente uma função genérica validarOuLancar<T>(valor: unknown, schema: ZodSchema<T>): T que recebe um valor desconhecido, valida com um schema Zod e retorna o valor tipado ou lança um ErroValidacao customizado. Demonstre o uso em uma rota Express tipada.

Dica: Use z.safeParse() para capturar o erro sem lançar exceção diretamente.

ESLint, Prettier e Git

5.4 TEÓRICO

O que é linting e por que ele é importante em projetos de equipe? Qual a diferença entre erros e warnings no ESLint? Explique o que é o prettier e como ele difere do ESLint conceitualmente.

5.5 TEÓRICO

O que é um commit semântico (Conventional Commits)? Explique os tipos: feat, fix, docs, style, refactor, test, chore. Por que eles são importantes para geração automática de CHANGELOG e versionamento semântico?

5.6 PRÁTICO

Para cada situação abaixo, escreva a mensagem de commit semântico correta:

- Adicionar autenticação JWT às rotas de usuário
- Corrigir bug onde o total do pedido não incluía o frete
- Atualizar a documentação do endpoint de produtos
- Refatorar o serviço de email para usar Clean Architecture
- Adicionar testes unitários para a entidade Tarefa
- Configurar ESLint e Prettier no projeto

TDD e Testes Avançados

5.7 MISTO

Explique o ciclo Red → Green → Refactor do TDD. Quais são os benefícios e as desvantagens de escrever testes antes do código? Quando o TDD é mais valioso e quando pode ser uma sobrecarga desnecessária?

5.8 PRÁTICO

Aplique TDD para implementar uma função processarPagamento(valor, cartao, config). Comece escrevendo os testes para os seguintes cenários ANTES de implementar a função:

- Pagamento aprovado com cartão válido
- Rejeitar valor negativo ou zero
- Rejeitar cartão expirado (data de vencimento passada)
- Rejeitar CVV com menos de 3 dígitos
- Simular falha de gateway com jest.fn() retornando erro

Dica: Escreva os testes primeiro, veja-os falhar (red), implemente o mínimo para passar (green), depois refatore.

MÓDULO 6

React e Front-end Moderno

Componentes, JSX, hooks, estado global com Zustand, dados do servidor com React Query e SPAs com React Router.

Artigos do site

#41 Introdução ao React
#42 React Hooks em profundidade
#43 Estado global com Zustand e React Query
#44 React Router: navegação em SPAs
#45 Revisão + Projeto Final: SPA Completa

Componentes e Hooks

6.1 TEÓRICO

Explique o que são "re-renders" no React e como o Virtual DOM funciona. Quando um componente re-renderiza? Quais hooks existem para evitar re-renders desnecessários e como funcionam?

6.2 PRÁTICO

Crie um componente React ListaTarefas que:

- Mantém o estado de uma lista de tarefas com useState
- Permite adicionar novas tarefas (input controlado)
- Permite marcar tarefas como concluídas (toggle)
- Permite remover tarefas
- Exibe contagem de pendentes / total
- É extraído em subcomponentes: ItemTarefa, FormNovaTarefa, ContadorTarefas

Dica: Use useCallback para as funções passadas como props.

6.3 PRÁTICO

Implemente um hook customizado useDebounce(valor, delay) que retorna o valor "estabilizado" após o delay. Depois implemente useLocalStorage(chave, valorInicial) que persiste e lê do localStorage. Demonstre os dois em um componente de busca que salva o termo no localStorage.

Zustand e React Query

6.4 TEÓRICO

Explique a diferença entre "estado de UI" e "estado de servidor". Por que React Query (ou SWR) é mais adequado para estado de servidor do que useState ou Zustand? O que é "stale time" e "cache time"?

6.5 PRÁTICO

Configure uma store Zustand para autenticação com:

- Estado: usuario, token, carregando, erro
- Ação login(email, senha): chama a API, salva token no localStorage
- Ação logout: limpa estado e localStorage
- Ação verificarToken: verifica se há token salvo e restaura sessão
- Middleware persist para sobreviver a reload da página
- Selector para saber se o usuário é admin

6.6 PRÁTICO

Usando React Query, implemente:

- useQuery para listar tarefas com filtros (status, prioridade) como queryKey
- useMutation para criar tarefa com invalidação do cache na conclusão
- useMutation com update otimista para toggle de status (concluir/reabrir)
- Tratamento de loading e erro em todos os casos
- Refetch automático quando a janela recebe foco

React Router e Projeto

6.7 PRÁTICO

Implemente um sistema de rotas com React Router v6:

- Rota pública /login com redirecionamento para /dashboard se já logado
- Rotas protegidas /dashboard, /tarefas, /perfil que redirecionam para /login se não autenticado
- Rota /tarefas/:id para detalhes, com useParams
- Rota 404 com componente NaoEncontrado
- Lazy loading em todas as páginas com React.lazy e Suspense

6.8 MISTO

Explique o que é "prop drilling" e como o Context API resolve esse problema. Quando você usaria Context em vez de Zustand? Implemente um tema claro/escuro usando Context API puro (sem Zustand), com um hook useTheme() e um componente ThemeProvider.

MÓDULO 7

Deploy e Infraestrutura

Deploy em Vercel e Railway, MongoDB Atlas, segurança (OWASP, JWT, bcrypt), performance (Web Vitals, bundle, índices) e CI/CD.

Artigos do site

#46 Deploy: do código ao ar
#47 Segurança em aplicações web
#48 Performance em aplicações web
#49 Revisão + Projeto Final: Produção Real

Deploy

7.1 TEÓRICO

Explique a diferença entre deployment de front-end (arquivos estáticos em CDN) e back-end (processo Node.js em servidor). Por que uma SPA precisa de um arquivo de configuração especial (vercel.json / netlify.toml / nginx.conf) para o roteamento funcionar corretamente?

7.2 PRÁTICO

Escreva o arquivo .github/workflows/deploy.yml para um pipeline CI/CD que:

- Roda npm run test em todo Pull Request aberto
- Faz deploy automático na Vercel (front) e Railway (back) apenas em push para main
- O deploy do front-end aguarda o deploy do back-end terminar (needs)
- Verifica o health check da API antes de declarar sucesso
- Usa secrets do GitHub para tokens sensíveis

Segurança

7.3 TEÓRICO

Explique os seguintes vetores de ataque e suas defesas:

- SQL/NoSQL Injection: como ocorre e como validar/sanitizar
- XSS (Cross-Site Scripting): como React protege por padrão e quando não protege
- CSRF: por que JWT em Authorization header é imune por design
- Timing attacks em comparação de senhas: por que usar bcrypt.compare e não ===

7.4 PRÁTICO

Implemente as seguintes proteções de segurança em uma API Express:

- Rate limiting de 5 requisições por 15 minutos apenas no endpoint /auth/login
- Middleware que valida se o usuário só acessa seus próprios recursos (ownership check)
- Configuração CORS que permite apenas a URL do front-end em produção
- Configuração do Helmet com Content-Security-Policy básica

Performance

7.5 TEÓRICO

Explique os três Core Web Vitals: LCP, INP e CLS. O que mede cada um, qual o valor considerado "bom" e quais técnicas ajudam a melhorar cada métrica?

7.6 PRÁTICO

Dado o código React abaixo com problemas de performance, identifique e corrija todos os problemas:

- const ListaProdutos = ({ produtos, onComprar }) => { ... }: componente re-renderiza toda vez que o pai renderiza mesmo com os mesmos produtos
- Dentro da lista, const precoFormatado = produtos.map(p => formatarMoeda(p.preco)): recalculado a cada render
- O handler onComprarClick é recriado a cada render e passado para ItemProduto
- Uma lista de 1000 produtos é toda renderizada no DOM de uma vez

Dica: Use React.memo, useMemo, useCallback e virtualização para resolver.

7.7 PRÁTICO

Escreva a query MongoDB e o índice necessário para buscar tarefas de um usuário específico filtradas por status, ordenadas por prazo. Use .explain("executionStats") para verificar que usa IXSCAN e não COLLSCAN. Depois otimize a query com .lean() e .select() mostrando os campos corretos.

MÓDULO 8

Arquitetura e Padrões

Padrões de projeto (GoF), princípios SOLID, Clean Architecture e Domain-Driven Design. Organizando código para durar.

Artigos do site

#50 Padrões de Projeto em JavaScript
#51 Arquitetura de Software: SOLID, Clean Architecture e DDD
#52 Revisão do Módulo 8 + Projeto: Refatoração com Clean Architecture

Padrões de Projeto

8.1 TEÓRICO

Explique os padrões abaixo e dê um exemplo real de onde cada um é usado no ecossistema JavaScript/Node.js:

- Singleton: onde o Node.js já o usa por padrão?
- Observer: qual API nativa do Node.js implementa este padrão?
- Strategy: cite um exemplo no Express ou em bibliotecas populares
- Adapter: por que ele é essencial ao integrar APIs de pagamento externas?

8.2 PRÁTICO

Implemente o padrão Builder para construção de emails. A classe EmailBuilder deve ter métodos encadeáveis: para(destinatario), assunto(texto), corpo(html), cc(emails), cco(emails), prioridade(nivel), build(). O método build() deve validar que para e assunto foram definidos antes de retornar o objeto email final.

8.3 PRÁTICO

Implemente o padrão Observer para um sistema de eventos de domínio:

- Classe EventEmitter com on(evento, handler), off(evento, handler), emit(evento, dados)
- Os handlers devem rodar em paralelo com Promise.allSettled
- Erros em um handler não devem cancelar os outros
- Método once(evento, handler) para handlers que disparam apenas uma vez
- Demonstre com os eventos: tarefa:criada, tarefa:concluida, tarefa:atrasada

SOLID

8.4 TEÓRICO

Cada trecho abaixo viola um princípio SOLID (o da letra indicada). Explique a violação e como corrigi-la:

- S: class ServicoUsuario { cadastrar(dados) { /* valida */ /* salva no banco */ /* envia email */ /* gera PDF */ } }
- O: function calcularDesconto(tipo) { if(tipo==="vip") ... else if(tipo==="regular") ... } // adicionar "premium" exige modificar a função
- D: class ServicoTarefa { constructor() { this.db = new MongoDB(); this.email = new Nodemailer(); } }

8.5 PRÁTICO

Refatore o código abaixo aplicando o Dependency Inversion Principle. O ServicoRelatorio deve receber suas dependências via constructor em vez de instanciá-las internamente:

```
class ServicoRelatorio {
  async gerar(tipo) {
    const db = new MongoDB(process.env.MONGO_URL);
    const dados = await db.query("SELECT * FROM pedidos");
    const pdf = new PDFGenerator();
    return pdf.gerar(dados);
  }
}
```

Clean Architecture

8.6 TEÓRICO

Descreva as quatro camadas da Clean Architecture e a responsabilidade de cada uma. Por que a regra de dependência afirma que as dependências sempre apontam para dentro? O que aconteceria se a camada de domínio importasse algo do Express ou do Mongoose?

8.7 PRÁTICO

Implemente a feature de "criar produto" completa em Clean Architecture:

- domain/entities/Produto.js: entidade com validações invariantes (preço > 0, nome min 2)
- domain/repositories/IProdutoRepository.js: interface com salvar(), buscarPorId()
- application/useCases/CriarProduto.js: orquestra entidade + repositório + evento
- infrastructure/database/MongoProdutoRepository.js: implementação com Mongoose
- interfaces/http/controllers/ProdutoController.js: traduz HTTP → caso de uso

Dica: O domínio não deve ter nenhum require() para express, mongoose ou qualquer biblioteca externa.

8.8 PRÁTICO

Escreva testes unitários para a entidade Produto e para o caso de uso CriarProduto criados no exercício anterior. Os testes não devem usar banco de dados — use um RepositorioEmMemoria que implemente a interface IProdutoRepository. Cubra pelo menos: criação válida, validação de preço negativo, validação de nome curto, emissão de evento ao criar.

MÓDULO 9

Tópicos Avançados

WebSockets com Socket.IO, filas de tarefas com BullMQ/Redis e projeto final integrando toda a série.

Artigos do site

#53 WebSockets e Comunicação em Tempo Real
#54 Filas de Tarefas e Processamento Assíncrono
#55 Projeto Final: Revisão Completa e Aplicação de Produção

WebSockets

9.1 TEÓRICO

Compare HTTP (requisição-resposta) com WebSocket (conexão persistente bidirecional). Em quais situações cada protocolo é mais adequado? Explique também quando usar Server-Sent Events (SSE) em vez de WebSocket.

9.2 PRÁTICO

Configure Socket.IO em um servidor Express existente com:

- Autenticação JWT no middleware de conexão (handshake)
- O usuário entra automaticamente em uma sala "usuario:ID" ao conectar
- Evento entrar:projeto que verifica permissão e entra na sala "projeto:ID"
- Evento tarefa:criada broadcast para todos na sala do projeto EXCETO o criador
- Handler de disconnect que loga o motivo

Dica: Não é necessário front-end — teste com o cliente Socket.IO em Node.

9.3 PRÁTICO

Escreva o hook React useSocket() que:

- Mantém uma instância singleton da conexão
- Recebe o token do Zustand e conecta ao servidor
- Oferece funções emitir(evento, dados) e ouvir(evento, callback)
- A função ouvir retorna um cleanup function para remover o listener
- Reconecta automaticamente quando o token muda (usuário faz login)

Filas de Tarefas

9.4 TEÓRICO

Explique o padrão produtor-consumidor e por que ele é fundamental para sistemas escaláveis. Quais problemas surgem quando processamos tarefas longas de forma síncrona na requisição HTTP? Cite pelo menos quatro cenários reais onde filas de tarefas são essenciais.

9.5 PRÁTICO

Configure uma fila BullMQ para envio de emails:

- Crie a fila filaEmail com: 3 tentativas, backoff exponencial de 1s, removeOnComplete após 24h
- Implemente o worker com templates para: boasVindas, redefinirSenha, tarefaCriada
- Adicione rate limiting: máximo 10 emails por segundo
- Registre eventos de completed e failed para logging
- Crie a função enfileirarEmail(tipo, para, dados) para uso nos controllers

9.6 PRÁTICO

Implemente um job recorrente com BullMQ que:

- Roda todo dia às 8h (horário de Brasília) via cron pattern
- Busca todas as tarefas com prazo nas próximas 24 horas
- Enfileira um email de lembrete para cada usuário afetado
- Usa jobId único para evitar duplicatas se o servidor reiniciar
- Loga quantas notificações foram enviadas ao final

Projeto Final

9.7 MISTO

Faça uma revisão da arquitetura completa do projeto final. Responda:

- Por que o index.js inicia a conexão com o banco ANTES de criar as rotas?
- Por que o Socket.IO compartilha a mesma porta que o Express em vez de usar uma porta diferente?
- Por que o container.js é o único arquivo que conhece as implementações concretas (MongoDB, Redis)?
- Como os WebSockets e a API REST colaboram sem se duplicar?
- Por que o deploy do front-end aguarda o deploy do back-end no CI/CD?

9.8 PRÁTICO

Implemente o módulo completo de "Projetos" seguindo Clean Architecture, integrando:

- Entidade Projeto com: nome, descricao, membros (array de userId), status, criadoPor
- Casos de uso: CriarProjeto, AdicionarMembro, ArquivarProjeto
- Ao criar um projeto: enfileira email de confirmação + emite evento WebSocket para todos os membros
- Rota protegida: apenas o criador pode arquivar
- Testes unitários para a entidade e o caso de uso CriarProjeto

Dica: Este é o exercício de maior escopo da série — dedique tempo adequado.

9.9 PRÁTICO

Escreva o checklist pessoal de deploy para o seu projeto final, adaptando o checklist do Artigo 52 para o seu contexto. Para cada item, documente:

- O que verificar e como verificar
- Qual o impacto se este item falhar em produção
- Como você automatizaria a verificação no CI/CD

Dica: Este exercício é reflexivo — não há resposta certa, apenas uma análise pessoal do sistema.

PARA IR ALÉM

Desafios integradores

Projetos maiores, que juntam vários módulos. Não há uma única resposta certa.

D1

Uma calculadora, duas interfaces

Escreva uma calculadora com sete operações: soma, subtração, multiplicação, divisão, potência (expoente inteiro), porcentagem (`15 % 200` é 15% de 200) e raiz quadrada. Ela deve: (a) acertar `0,1 + 0,2 = 0,3`, o que o `Number` do JavaScript não faz sozinho, aceitando vírgula ou ponto e mostrando até 16 dígitos significativos; (b) guardar o último resultado em `ans`, que pode ser usado na conta seguinte; (c) recusar divisão por zero, raiz de negativo, expoente fracionário e texto que não é número com um erro próprio (de `name` `"ErroDeConta"`) e uma mensagem clara, **sem encerrar o programa**. Escreva o núcleo num módulo ES que não sabe nada de entrada e saída, e reaproveite-o em duas interfaces: uma de **terminal** com Node.js (comandos `hist` e `sair`) e uma página **no navegador**, com teclado na tela, visor e histórico. O artigo #10, "Mini Projeto: Calculadora no Console", é o ponto de partida.

Sua solução (arquitetura, decisões e código principal):

Bom estudo e bons exercícios!

Dominando o JavaScript em 1 Ano · theknowledgeisle.com