Relatório Técnico: Projeto de um Serviço de Autenticação Seguro em TypeScript

1. Introdução: Fundamentos da Autenticação Segura

A autenticação constitui a base da segurança em qualquer sistema digital, assegurando que apenas usuários devidamente autorizados possam acessar recursos e informações confidenciais. A concepção de um serviço de autenticação robusto e seguro é, portanto, imperativa para salvaguardar dados sensíveis e preservar a confiança dos usuários.

Visão Geral e Importância da Autenticação

O processo de login em plataformas online deve ser ágil e intuitivo, sem comprometer a proteção das informações pessoais da conta.[1] A segurança e a integridade dos dados dependem intrinsecamente de um fluxo de autenticação eficaz, que restringe o acesso exclusivamente a usuários autorizados.[2] O "Authentication Flow", ou fluxo de autenticação, refere-se precisamente a essa sequência de etapas que um usuário percorre para se identificar e obter acesso a um sistema ou aplicação.[2]

Princípios Essenciais de Segurança

Para além da simples combinação de nome de usuário e senha, a segurança moderna exige a incorporação de princípios mais avançados. A Autenticação Multifator (MFA) emerge como uma camada adicional de segurança, exigindo uma ou mais formas de autenticação além da senha tradicional para conceder acesso.[3] Esta abordagem

oferece uma proteção substancial contra o roubo de contas, pois, mesmo que uma senha seja comprometida, um cibercriminoso ainda necessitaria dos fatores de autenticação adicionais para efetuar o login.[3] Métodos comuns de MFA incluem o uso de aplicativos autenticadores ou a autenticação biométrica, como o Face ID.[3]

Outra inovação significativa são as chaves de acesso (Passkeys), que representam uma forma mais simplificada e segura de login, eliminando a necessidade de senhas. Elas operam utilizando o bloqueio de tela do dispositivo do usuário, como impressão digital, reconhecimento facial ou PIN. As chaves de acesso são um padrão da indústria, fundamentadas nos padrões da FIDO Alliance e do W3C, e empregam os mesmos protocolos criptográficos de chave pública que as chaves de segurança físicas, conferindo-lhes uma robusta capacidade de proteção contra ataques de phishing, preenchimento de credenciais e outras ameaças remotas.[1] O Google, por exemplo, destaca que as chaves de acesso são armazenadas na Conta Google e ficam disponíveis em todos os dispositivos sincronizados, facilitando o login de forma segura e eficiente.[1]

Os componentes fundamentais de um fluxo de autenticação abrangem a interface de login, o sistema de gerenciamento de usuários e o mecanismo de verificação de credenciais. A interface de login é o ponto de entrada onde o usuário insere suas informações. O sistema de gerenciamento de usuários é responsável por armazenar e organizar os dados dos usuários, enquanto o mecanismo de verificação de credenciais valida a correspondência das informações fornecidas com as registradas no sistema. A colaboração desses componentes assegura um fluxo de autenticação seguro e eficiente.[2]

A evolução dos paradigmas de autenticação aponta para uma clara transição para além da dependência exclusiva de senhas tradicionais. A fragilidade inerente das senhas, suscetíveis a ataques de phishing e preenchimento de credenciais, impulsiona a necessidade de camadas defensivas adicionais, como a MFA, ou de substituições mais seguras, como as chaves de acesso. Essa transição indica que, para um projeto de serviço de autenticação em TypeScript, não basta apenas implementar um sistema de senha seguro. É crucial incorporar a MFA como uma funcionalidade central desde o início e considerar uma arquitetura que permita a eventual adoção de métodos sem senha. Tal abordagem garante que o serviço de autenticação permaneça relevante e ofereça o nível de segurança esperado em sistemas modernos, mitigando os riscos associados à dependência única de senhas.

2. Gerenciamento Seguro de Senhas

O armazenamento seguro de senhas é um dos pilares mais críticos de um serviço de autenticação. Falhas nesta área podem resultar em violações de dados de proporções catastróficas.

Hashing e Salting: Por que e Como

É fundamental compreender que senhas **nunca devem ser armazenadas em texto simples**.[4] Essa prática é extremamente perigosa, pois qualquer pessoa com acesso interno ao banco de dados poderia visualizá-las.[5] Da mesma forma, armazenar senhas apenas criptografadas não é uma solução eficaz.[4] Se a chave de criptografia for comprometida, todas as senhas poderiam ser facilmente descriptografadas, anulando o propósito da segurança.[4]

A abordagem correta e amplamente aceita é o **Hashing**. Uma função de hash é unidirecional, o que significa que ela transforma a senha em uma string de comprimento fixo (o hash), mas é computacionalmente inviável reverter o hash para obter a senha original.[4] Para aprimorar ainda mais a segurança do hashing, o

Salting é essencial. Um "sal" (salt) é um dado aleatório e único, gerado para cada senha individualmente, que é adicionado à senha antes de ser hasheada.[5] O propósito principal do sal é proteger contra ataques de dicionário e tabelas arco-íris (rainbow tables), que são ataques pré-computados de hashes para senhas comuns.[6] Como o sal é diferente para cada senha, mesmo que dois usuários utilizem a mesma senha, seus hashes resultantes serão distintos, frustrando esses tipos de ataques.[6] O sal não precisa ser criptografado e pode ser armazenado em texto simples junto com o hash da senha no banco de dados. O formato de armazenamento recomendado é

hash(senha + salt).[5]

Algoritmos de Hashing Recomendados

Para o hashing de senhas, é imperativo utilizar algoritmos específicos para essa finalidade, conhecidos como Funções de Derivação de Chave (KDFs). Esses algoritmos são projetados para serem intencionalmente lentos e computacionalmente intensivos, o que os torna resistentes a ataques de força bruta.[8]

- **Bcrypt:** É uma função de hash adaptativa baseada no algoritmo Blowfish. Uma de suas características mais importantes é a incorporação de um "custo de trabalho" (também conhecido como fator de carga), que permite ajustar o custo computacional necessário para gerar o hash.[8] Essa adaptabilidade é crucial, pois permite que o algoritmo se torne mais lento ao longo do tempo, mantendo sua resistência a ataques de força bruta à medida que o poder computacional dos atacantes aumenta.[10] O Bcrypt é amplamente utilizado e considerado um dos métodos mais seguros para hashing de senhas.[10] Para ambientes Node.js/TypeScript, as bibliotecas
  bcrypt [10] ou
  bcrypt.js [11] são as escolhas comuns.
  bcrypt.js é uma implementação em JavaScript com suporte a TypeScript, compatível com a ligação C++ do bcrypt, embora seja cerca de 30% mais lenta.[11] Ambas oferecem métodos síncronos e assíncronos para geração de sal (genSaltSync, genSalt) e hashing (hashSync, hash) e comparação (compareSync, compare).[11]
- **Argon2 e Scrypt:** São alternativas robustas ao Bcrypt, também classificadas como KDFs e projetadas especificamente para a segurança de senhas.[8] Argon2, em particular, foi o vencedor do Password Hashing Competition (PHC) e é considerado altamente resistente, inclusive a ataques baseados em hardware.[8]

É crucial **evitar** algoritmos como MD5 e SHA-1 para hashing de senhas. Embora sejam rápidos, possuem vulnerabilidades conhecidas que os tornam inadequados para aplicações seguras.[8] SHA-256 e SHA-3, por sua vez, oferecem um excelente equilíbrio entre segurança e eficiência para aplicações críticas em geral, mas não são ideais para hashing de senhas devido à sua velocidade, que os torna mais suscetíveis a ataques de força bruta e tabelas arco-íris se não forem combinados com salting e alongamento de chave adequados.[8]

A segurança do hashing de senhas não é obtida pela velocidade, mas pela lentidão intencional e pelo custo computacional que impõe aos atacantes. Cada tentativa de adivinhar uma senha exige um cálculo de hash caro, e o sal aumenta ainda mais esse custo, pois impede que um atacante pré-calcule hashes para múltiplas senhas de uma vez. Ao projetar o serviço de autenticação em TypeScript, os desenvolvedores devem reconhecer que as operações de hashing de senha (durante o registro e o

login) serão inerentemente mais lentas do que outras operações de banco de dados ou API. Esta lentidão não é um problema, mas uma característica de segurança. É vital que essas operações sejam realizadas de forma assíncrona para não bloquear o *event loop* do Node.js e degradar o desempenho geral da aplicação.[11] A lentidão computacional proposital e o uso de sal aumentam exponencialmente o tempo e os recursos necessários para um ataque de força bruta, tornando-o inviável e, assim, fortalecendo a segurança.

A tabela a seguir compara os principais algoritmos de hashing relevantes para o armazenamento seguro de senhas:

Tabela 1: Comparativo de Algoritmos de Hashing para Senhas

Algoritmo	Características Chave	Resistência a Ataques	Uso Típico	Prós	Contras
Bcrypt	Adaptativo (fator de custo), Salting	Força Bruta (CPU/GPU)	Hashing de Senhas	Amplamente testado, boa resistência, adaptável ao tempo.	Mais lento que SHA-256, mas isso é intencional para senhas.
Argon2	Adaptativo (custo de tempo, memória, paralelismo), Salting	Força Bruta (CPU/GPU/Memória)	Hashing de Senhas	Vencedor do PHC, resistente a ataques baseados em hardware (ASICs, GPUs).	Mais recente, pode ter menos suporte em algumas plataformas.
Scrypt	Adaptativo (custo de tempo, memória), Salting	Força Bruta (CPU/GPU/Memória)	Hashing de Senhas	Bom para resistência a hardware, mais antigo que Argon2.	Menos flexível que Argon2 em termos de parâmetros.
SHA-256/SHA-3	Rápido, unidirecional	Colisões (para integridade de dados)	Verificação de Integridade de Dados, Assinaturas	Rápido, eficiente.	Não recomendado para senhas devido à sua

			Digitais		velocidade, vulnerável a tabelas arco-íris sem salting adequado.

Diretrizes de Políticas de Senha (NIST, OWASP)

As diretrizes modernas para políticas de senha, emitidas por órgãos como o NIST (National Institute of Standards and Technology) e o OWASP (Open Worldwide Application Security Project), passaram por uma significativa evolução. A crença histórica de que senhas precisavam de complexidade arbitrária (ex: exigir maiúsculas, números, caracteres especiais) e expiração periódica foi revisada. Estudos demonstraram que essas regras antigas, paradoxalmente, tornavam as senhas menos seguras, pois levavam os usuários a criar padrões previsíveis ou a reutilizar senhas.[12]

Atualmente, a ênfase recai sobre:

- **Comprimento da Senha:** O NIST recomenda um mínimo de 15 caracteres para senhas.[12] O OWASP sugere um mínimo de 8 caracteres, mas enfatiza que a aplicação deve permitir um comprimento máximo de pelo menos 64 caracteres para acomodar frases-senha (passphrases).[13]
- **Complexidade (Regras de Composição):** As diretrizes mais recentes do NIST e OWASP **não exigem mais regras de complexidade arbitrárias**.[12] A remoção dessas restrições, combinada com o incentivo a senhas mais longas (frases-senha) e a verificação contra listas de senhas comprometidas, resulta em senhas mais fortes e uma experiência do usuário menos frustrante.
- **Caracteres Permitidos:** É recomendado permitir o uso de todos os caracteres imprimíveis ASCII e UNICODE, incluindo espaços.[12]
- **Dicas de Senha/Perguntas de Segurança:** Não são permitidas sob as diretrizes do NIST.[12]
- **Verificação de Senhas "Ruins Conhecidas":** Novas senhas devem ser verificadas contra uma lista (blocklist) de senhas comuns ou previamente comprometidas, provenientes de dicionários, vazamentos anteriores ou padrões de teclado.[12] Serviços como "Have I Been Pwned" podem ser utilizados para essa verificação.[13]

- **Expiração de Senha:** Organizações **não devem exigir** que os usuários alterem suas senhas em intervalos definidos (ex: a cada 45, 60 ou 90 dias).[12] A alteração de senha só deve ser forçada em caso de comprometimento conhecido.[12]
- **Gerenciadores de Senha:** O uso de gerenciadores de senha deve ser permitido e facilitado, inclusive permitindo a funcionalidade de colar senhas.[12]
- **MFA:** A Autenticação Multifator deve ser incentivada para aumentar a segurança.[13] O NIST desaconselha o uso de SMS como fator de MFA, sugerindo alternativas mais robustas como senhas de uso único baseadas em tempo (TOTP) geradas por aplicativos móveis, tokens de hardware ou verificação biométrica.[12]

Melhores Práticas de Armazenamento

Além do hashing e salting, outras práticas são cruciais para o armazenamento seguro de senhas:

- Utilizar um gerenciador de senhas para armazenar e gerar senhas complexas e únicas de forma segura.[16] Gerenciadores criptografam as credenciais e facilitam a criação e o acesso.[16]
- Evitar escrever senhas em notas adesivas ou salvá-las em planilhas, pois isso aumenta o risco de vazamento de dados.[17]
- Realizar backup regularmente das senhas em um local seguro, como um dispositivo externo ou serviço de armazenamento em nuvem criptografado, para garantir acesso mesmo em caso de perda de dados.[16]
- Compartilhar senhas com extrema cautela, apenas quando estritamente necessário e utilizando métodos seguros, evitando e-mails ou mensagens de texto não criptografadas.[16]

3. Fluxos de Autenticação Essenciais

Um serviço de autenticação transcende o mero armazenamento de senhas, abrangendo a gestão de todo o ciclo de vida do usuário, desde o registro inicial até a redefinição de senha.

Fluxo de Login

O processo de login inicia-se quando o usuário acessa a interface de login e insere suas credenciais, tipicamente nome de usuário (ou e-mail) e senha.[2] No backend, o sistema verifica essas informações. Para a senha, o sistema hasheia a senha fornecida pelo usuário, utilizando o sal único armazenado para aquele usuário específico, e compara o hash resultante com o hash da senha previamente armazenado no banco de dados.[5] Se as credenciais forem validadas com sucesso, o usuário é autenticado e recebe um token de sessão, como um JSON Web Token (JWT), que concede acesso ao sistema para requisições subsequentes.[2]

É de suma importância implementar mecanismos de limitação de tentativas de login (rate limiting) para prevenir ataques de força bruta, onde atacantes tentam adivinhar senhas repetidamente.[13]

Fluxo de Registro de Usuário

No fluxo de registro, o usuário fornece um nome de usuário (ou e-mail) e uma senha. O sistema, então, deve gerar um sal único e aleatório para esta nova senha.[5] A senha, combinada com o sal, é hasheada utilizando um algoritmo robusto e adaptativo, como bcrypt ou Argon2.[5] Finalmente, o nome de usuário/e-mail, o hash da senha e o sal são armazenados no banco de dados. É altamente recomendável a verificação de e-mail para confirmar a posse e mitigar registros maliciosos. Além disso, novas senhas devem ser verificadas contra listas de senhas comprometidas para evitar o uso de credenciais fracas ou já vazadas.[12]

Fluxo de Redefinição de Senha Segura

O fluxo de redefinição de senha é um ponto de entrada sensível e, portanto, um vetor de ataque crítico. Seu design deve ser tratado com o mesmo rigor que o fluxo de

login, incorporando múltiplas camadas de segurança.

O processo começa com o usuário inserindo seu e-mail para redefinir a senha.[22] Existe um debate na comunidade de segurança sobre se o backend deve informar explicitamente se o e-mail não está presente no banco de dados. Alguns argumentam que revelar essa informação pode permitir que atores mal-intencionados enumerem e-mails válidos. Contudo, é notável que grandes plataformas como Google, Facebook e Netflix, que priorizam a experiência do usuário, retornam a mensagem "E-mail não existe" quando um e-mail aleatório é inserido.[22] Uma abordagem equilibrada prioriza a usabilidade, mas mitiga os riscos de enumeração com mecanismos robustos como rate limiting e CAPTCHA na tela de redefinição.

Se o e-mail for válido, um código secreto de uso único (OTP - One-Time Password) é gerado e enviado para o endereço de e-mail do usuário.[22] A interface do usuário então solicita a inserção desse código. O backend é responsável por verificar a validade do OTP. É crucial limitar o número de tentativas para inserir o OTP; após um número predefinido de tentativas falhas, uma nova solicitação de redefinição deve ser exigida para evitar ataques de força bruta ao OTP.[23]

Uma vez que o código é verificado com sucesso, o usuário é habilitado a inserir e confirmar a nova senha. Esta nova senha deve aderir rigorosamente às diretrizes de política de senha seguras, focando no comprimento adequado, na ausência de regras de complexidade arbitrárias e na verificação contra listas de senhas conhecidas como ruins.[12] Após a redefinição bem-sucedida, o usuário é redirecionado para a tela de login.[22]

Para contas que possuem Autenticação Multifator (MFA) habilitada, o processo de redefinição de senha deve obrigatoriamente exigir uma verificação de identidade adicional via MFA.[13] Isso pode envolver o envio de múltiplos nonces (números usados uma única vez): um para notificação no e-mail atual do usuário, alertando sobre a tentativa de mudança, e outro para confirmação no novo e-mail (ou via outro fator MFA), instruindo o usuário a confirmar a alteração.[13] A ausência de múltiplas camadas de segurança neste fluxo o transforma em um ponto de vulnerabilidade significativo, permitindo que atacantes contornem senhas fortes e MFA, levando à tomada de conta. A implementação dessas camadas adicionais é, portanto, essencial para a segurança global do sistema.

4. Mecanismos de Autenticação Avançados

Além do login/senha básico, sistemas modernos incorporam mecanismos avançados para aumentar a segurança e a usabilidade, refletindo uma convergência para a autenticação sem senha e multifator como o padrão de segurança.

Autenticação Multifator (MFA): Tipos e Integração

A Autenticação Multifator (MFA) é um requisito de segurança contemporâneo, exigindo duas ou mais formas de autenticação para acessar uma conta, adicionando uma camada extra de proteção.[3] Essa abordagem torna significativamente mais difícil para um atacante obter acesso, mesmo que a senha primária seja comprometida.[24]

A importância da MFA reside em sua capacidade de proteger contra vulnerabilidades comuns como senhas fracas, reutilizadas, ataques de phishing e infecções por malware, que são causas frequentes de comprometimento de contas.[24] Em um cenário onde violações de dados são cada vez mais comuns, a MFA atua como a barreira final para impedir o acesso não autorizado, mesmo que as credenciais primárias sejam vazadas.[25]

Os fatores de autenticação são classificados em três categorias principais [24]:

- **Algo que você sabe (Conhecimento):** Inclui senhas, PINs ou respostas a perguntas de segurança.
- **Algo que você tem (Posse):** Refere-se a um objeto físico ou digital que o usuário possui, como um smartphone (para aplicativos autenticadores ou OTPs via SMS), tokens de hardware ou cartões inteligentes. É importante notar que o NIST desaconselha o uso de SMS para MFA devido a vulnerabilidades conhecidas.[12]
- **Algo que você é (Inerência):** Envolve características biométricas únicas do usuário, como impressão digital, reconhecimento facial ou reconhecimento de voz.
- **Localização:** Baseia-se na localização geográfica do usuário no momento da tentativa de autenticação.

Os tipos mais comuns de MFA incluem aplicativos autenticadores (por exemplo, Google Authenticator, Authy), biometria (Face ID, impressão digital), e chaves de segurança de hardware (baseadas em padrões como FIDO U2F/WebAuthn). Embora OTPs via SMS/e-mail sejam utilizados, são considerados menos seguros e são

desaconselhados pelo NIST para MFA robusta.[3]

Chaves de Acesso (Passkeys) e Autenticação Sem Senha

As chaves de acesso representam o avanço mais significativo em autenticação sem senha. São um padrão da indústria (FIDO Alliance e W3C) que permite um login seguro e simples, eliminando a necessidade de memorizar e digitar senhas.[1] O funcionamento baseia-se no uso do bloqueio de tela do dispositivo (impressão digital, reconhecimento facial, PIN) para autenticação. Uma vez criadas, as chaves de acesso são armazenadas de forma segura (por exemplo, na Conta Google) e ficam disponíveis em todos os dispositivos sincronizados do usuário.[1]

Os benefícios de segurança das chaves de acesso são notáveis: oferecem proteção de última geração contra phishing, preenchimento de credenciais e outros ataques remotos, utilizando os mesmos protocolos criptográficos de chave pública que as chaves de segurança físicas. Além disso, proporcionam uma experiência de login significativamente mais rápida do que as senhas tradicionais.[1]

OAuth 2.0 e OpenID Connect (OIDC)

Para cenários de integração e Single Sign-On (SSO), protocolos como OAuth 2.0 e OpenID Connect são fundamentais.

- **OAuth 2.0:** É um protocolo de **autorização**, e não de autenticação direta. Ele delega o acesso a um recurso protegido (como uma API ou aplicativo web) através da emissão de tokens de acesso com escopo definido. O OAuth 2.0 permite que um aplicativo obtenha permissão para acessar recursos em nome do usuário, sem que o usuário precise compartilhar suas credenciais diretamente com o aplicativo.[26]
- **OpenID Connect (OIDC):** É uma camada de **autenticação** construída sobre o OAuth 2.0. O OIDC permite a autenticação do usuário e fornece funcionalidade de Single Sign-On (SSO). Ele define um tipo de token específico, o ID Token, que contém informações verificáveis sobre o usuário (conhecidas como "claims"), e padroniza áreas que o OAuth 2.0 deixa em aberto, como escopos e descoberta

de endpoints.[26]

Os componentes-chave nesses protocolos incluem [26]:

- **Cliente:** O aplicativo que busca acessar dados.
- **Servidor de Recurso:** A API ou aplicativo que detém os dados.
- **Proprietário do Recurso:** O usuário que é o proprietário dos dados.
- **Servidor de Autorização (ou Provedor OpenID):** A entidade que gerencia o acesso e emite os tokens (access token, refresh token, ID token).

Existem diversos fluxos (grants) no OAuth 2.0, adequados para diferentes tipos de clientes. Por exemplo, o Authorization Code Flow com PKCE é recomendado para clientes públicos (como aplicativos móveis), enquanto o Resource Owner Password Flow é mais restrito a clientes de primeira parte e onde MFA não é uma exigência, sendo geralmente desaconselhado para a maioria dos casos.[26] Dada a complexidade e as implicações de segurança, é fortemente encorajado o uso de bibliotecas ou serviços pré-escritos (como Google Identity Services) para implementar OAuth 2.0 e OIDC.[27]

JSON Web Tokens (JWT) para Sessão e Autorização

JSON Web Token (JWT), pronunciado "jot", é um padrão aberto (RFC 7519) para a transmissão segura e compacta de informações entre partes como um objeto JSON. A integridade e a confiança das informações são garantidas por uma assinatura digital.[28]

A estrutura de um JWT é composta por três partes separadas por pontos (.): o Header (cabeçalho), o Payload (carga útil) e a Signature (assinatura).[29]

- O **Payload** contém as "claims" (declarações), que são afirmações sobre uma entidade (tipicamente o usuário) e dados adicionais. As claims podem ser registradas (predefinidas, como iss para emissor, exp para tempo de expiração, sub para assunto), públicas ou privadas (personalizadas).[29]
- A **Assinatura** é criada a partir do cabeçalho codificado, do payload codificado, de um segredo e do algoritmo especificado no cabeçalho. Sua função é verificar se a mensagem não foi alterada durante o trânsito e, no caso de tokens assinados com chave privada, pode também verificar a identidade do remetente.[29]

Os benefícios dos JWTs incluem sua **compactação** (o que os torna adequados para ambientes HTML e HTTP), sua **segurança** (devido à assinatura digital) e sua **comunidade** (parsers JSON são suportados pela maioria das linguagens de programação).[28]

Os JWTs são amplamente utilizados em:

- **Autenticação:** Após um login bem-sucedido, um ID token (que é um JWT) é retornado ao cliente, conforme especificado pelo OpenID Connect.[28]
- **Autorização:** Um Access Token (que também pode ser um JWT) é incluído em cada requisição subsequente para acessar rotas ou recursos protegidos (como APIs) em nome do usuário.[28]
- **Troca de Informações:** Permitem a transmissão segura de informações entre diferentes partes de um sistema.[28]

Considerações de segurança são cruciais ao usar JWTs:

- **Nunca armazene informações sensíveis dentro de um JWT**, pois o conteúdo do payload é apenas codificado (base64url), não criptografado, e é visível para qualquer um que possua o token.[28]
- Em produção, JWTs devem ser **assinados com uma chave privada e verificados com uma chave pública**.[30]
- Sempre transmita JWTs **exclusivamente sobre HTTPS** para prevenir ataques man-in-the-middle.[28]
- A assinatura do JWT deve ser validada antes de seu uso.[28]
- Mantenha os tokens válidos pelo tempo mínimo necessário para reduzir a janela de oportunidade de um ataque.[29]

Para implementação em TypeScript/Node.js, bibliotecas como jsonwebtoken (para o servidor Node.js) [30] e

jwt-decode (para o cliente React) [31] são comumente empregadas. Frameworks como NestJS oferecem integração com JWT através de módulos como

@nestjs/jwt.[18]

A orquestração de protocolos e tokens, com o JWT atuando como credencial de autorização pós-autenticação, é um ponto chave. O JWT não é um método de autenticação por si só no sentido de verificar a identidade inicial do usuário (isso é feito por senha, MFA ou OIDC). Em vez disso, ele serve como um mecanismo de sessão e autorização que é emitido após a autenticação bem-sucedida. Ele funciona

como uma credencial de curta duração que atesta que o usuário já foi autenticado e possui permissão para acessar determinados recursos. Assim, o serviço de autenticação em TypeScript, após verificar as credenciais iniciais (e MFA, se aplicável), deve gerar um JWT. Este token será então utilizado pelo cliente para fazer requisições subsequentes à API, provando sua identidade e autorização sem a necessidade de uma reautenticação completa a cada requisição. A compreensão de que o JWT não deve conter informações sensíveis e deve ser transmitido via HTTPS é vital para um design de sistema mais escalável, sem estado e seguro para APIs.

5. Prevenção de Ataques Comuns

A concepção de um serviço de autenticação seguro exige uma compreensão aprofundada e a mitigação proativa de vulnerabilidades comuns, muitas das quais são categorizadas e detalhadas pelo OWASP Top 10.

OWASP Top 10 e sua Relevância

O OWASP Top 10 é um relatório fundamental, atualizado regularmente, que sumariza os dez riscos de segurança mais críticos para aplicações web, oferecendo orientações valiosas para desenvolvedores e profissionais de segurança.[33] Ele serve como uma ferramenta prática para a implementação de medidas de segurança, abrangendo técnicas de exploração, práticas de codificação segura e estratégias de mitigação.[33]

Para um serviço de autenticação, os seguintes riscos do OWASP Top 10 são particularmente relevantes:

- **A01:2021 - Broken Access Control (Controle de Acesso Quebrado):** Permite que invasores ignorem as verificações de autorização e executem tarefas como se fossem usuários privilegiados. A proteção pode ser alcançada garantindo o uso adequado de tokens de autorização e a imposição de controles rigorosos sobre eles.[34]
- **A02:2021 - Cryptographic Failures (Falhas Criptográficas):** Ocorre quando dados sensíveis, como senhas e informações financeiras, não são protegidos

adequadamente por criptografia. Para senhas, a mitigação envolve o uso de funções de hashing adaptativas e salgadas (como Argon2, bcrypt ou PBKDF2) e a evitação de métodos obsoletos como MD5 e SHA-1.[33]

- **A07:2021 - Identification and Authentication Failures (Falhas de Identificação e Autenticação):** Anteriormente conhecido como A2:2017, este risco está diretamente relacionado a implementações falhas de autenticação e gerenciamento de sessão que podem permitir que atacantes comprometam identidades de usuário.
- **A09:2021 - Security Logging and Monitoring Failures (Falhas de Logging e Monitoramento de Segurança):** A ausência de logs de segurança adequados ou um monitoramento ineficaz impede a detecção precoce de ataques, aumentando o tempo de resposta e o impacto potencial de uma violação.[36]

Ataques de Força Bruta e Limitação de Tentativas (Rate Limiting)

Um ataque de força bruta envolve tentativas repetidas e sistemáticas de adivinhar senhas, IDs de sessão ou outras credenciais.[20] Sistemas de autenticação que não implementam mecanismos de bloqueio ou limitação são particularmente vulneráveis, permitindo que atacantes tentem adivinhar senhas sem restrições.[20]

A prevenção eficaz inclui:

- **Limitação de Tentativas de Login (Rate Limiting):** Impor um limite no número de tentativas de login permitidas dentro de um determinado período de tempo.[13]
- **Bloqueio de Conta (Account Lockout):** Após um número predefinido de tentativas falhas (limiar de bloqueio), a conta é temporariamente bloqueada por um período específico (duração do bloqueio).[13]
- **Rastreamento:** Utilizar colunas no banco de dados, como failed_login_attempts e last_failed_login_time, para controlar o número de tentativas e redefinir o contador em caso de login bem-sucedido.[21]
- **MFA:** A Autenticação Multifator adiciona uma camada de segurança que dificulta o acesso mesmo que a senha seja comprometida.[19]

A defesa em profundidade para autenticação vai além de simplesmente exigir senhas fortes. Embora senhas robustas sejam a primeira linha de defesa [20], elas não são suficientes por si só. A ausência de limitação de tentativas de login é explicitamente identificada como um vetor de entrada para ataques de força bruta.[20] Portanto, a

implementação de MFA, o rastreamento de tentativas falhas no banco de dados e as políticas de bloqueio de conta são medidas essenciais. A falha em incorporar rate limiting e bloqueio de conta torna um sistema com senhas fortes um alvo fácil para ataques automatizados, enquanto a implementação dessas medidas aumenta a resiliência geral do sistema.

Prevenção de Injeção SQL

A Injeção SQL é um ataque malicioso onde código SQL é inserido em campos de entrada de usuário, explorando vulnerabilidades para expor, corromper ou deletar dados, ou obter acesso não autorizado.[38] Ocorre quando uma aplicação Node.js manipula incorretamente a entrada do usuário, utilizando-a diretamente em queries SQL sem validação, sanitização ou construção adequada.[38]

A prevenção mais eficaz envolve:

- **Queries Parametrizadas/Prepared Statements:** Esta é a forma mais robusta de prevenir Injeção SQL. Ela separa o código SQL dos dados de entrada, garantindo que a entrada do usuário seja tratada como dados literais e não como parte da lógica da query.[39]
- **ORMs (Object-Relational Mappers):** Muitos ORMs populares em Node.js/TypeScript (como TypeORM, Sequelize) utilizam prepared statements por padrão, abstraindo essa complexidade do desenvolvedor.
- **Validação e Sanitização de Entrada:** Validar e limpar (sanitizar) todas as entradas do usuário antes de utilizá-las em queries ou exibi-las em qualquer parte da aplicação.

Prevenção de Cross-Site Scripting (XSS)

O Cross-Site Scripting (XSS) é uma vulnerabilidade web crítica na qual atacantes injetam scripts maliciosos em páginas web, que são então executados no navegador de outros usuários.[40]

Para prevenir XSS:

- **Sanitização de HTML:** Nunca insira diretamente conteúdo gerado pelo usuário

em HTML usando innerHTML ou métodos similares. Em vez disso, utilize bibliotecas de sanitização (como o módulo xss do npm [41]) ou recursos de frameworks (como
DomSanitizer do Angular [40]) que escapam ou removem tags e atributos perigosos.
- **Content Security Policy (CSP):** Implementar uma Política de Segurança de Conteúdo que restringe as fontes de onde scripts, estilos e mídias podem ser carregados, mitigando significativamente o XSS.[40]

Prevenção de Cross-Site Request Forgery (CSRF)

O Cross-Site Request Forgery (CSRF) é um tipo de ataque onde comandos não autorizados são enviados de um usuário confiável para uma aplicação web, explorando a sessão ativa do usuário sem seu conhecimento ou consentimento.[40]

As principais técnicas de prevenção incluem:

- **Synchronizer Token Pattern (STP):** Gerar um token único para cada sessão de usuário (CSRF token) e incorporá-lo em todas as submissões de formulário ou requisições AJAX.[40] O servidor valida este token em cada requisição para garantir que a solicitação é legítima.[43] Em Node.js/TypeScript, pacotes como
csrf-csrf (para Express) ou @fastify/csrf-protection (para Fastify) podem ser utilizados.[42]
- **SameSite Cookies:** Definir o atributo SameSite nos cookies de sessão (com valores como strict ou lax) para que o navegador só os envie com requisições originadas do mesmo domínio.[40] Além disso, os cookies de sessão devem ser configurados com os atributos
httpOnly (para prevenir acesso via JavaScript no cliente) e secure (para garantir que sejam enviados apenas sobre HTTPS).[40]
- **Cabeçalhos de Segurança HTTP:** Implementar cabeçalhos HTTP de segurança como X-Frame-Options: DENY (previne clickjacking), X-XSS-Protection: 1; mode=block (habilita o filtro XSS do navegador) e X-Content-Type-Options: nosniff (impede o sniffing de tipo MIME).[40]

A validação de entrada é um pilar fundamental da segurança web. Um tema recorrente em diversas vulnerabilidades, incluindo Injeção SQL, XSS e, indiretamente, CSRF, é a manipulação ou o uso indevido de dados fornecidos pelo usuário. A raiz do

problema reside na confiança implícita na entrada. A falha em validar e sanitizar adequadamente a entrada do usuário abre portas para uma ampla gama de ataques, transformando dados aparentemente inofensivos em vetores de exploração. Portanto, o serviço de autenticação, e de fato qualquer parte da aplicação, deve implementar validação rigorosa de entrada (para tipos de dados, formatos, comprimentos), sanitização (para remover caracteres perigosos ou tags HTML) e escaping (para exibir dados de forma segura).

A tabela a seguir resume as melhores práticas de segurança do OWASP aplicáveis à autenticação:

Tabela 2: Melhores Práticas de Segurança OWASP para Autenticação

Categoria de Risco OWASP	Relevância para Autenticação	Estratégias de Mitigação	Referências
A02:2021 - Falhas Criptográficas	Armazenamento inseguro de senhas, uso de algoritmos fracos.	Usar funções de hashing adaptativas e salgadas (Bcrypt, Argon2, Scrypt). Evitar MD5, SHA-1.	4
A07:2021 - Falhas de Identificação e Autenticação	Implementação fraca de login, redefinição de senha, MFA.	Implementar MFA. Limitar tentativas de login (rate limiting, bloqueio de conta). Políticas de senha modernas (NIST/OWASP).	3
A03:2021 - Injeção	Injeção SQL através de campos de login/registro.	Usar queries parametrizadas ou ORMs. Validar e sanitizar todas as entradas do usuário.	38
A03:2021 - Injeção (XSS)	Injeção de scripts maliciosos em páginas de login/registro.	Sanitizar HTML de entrada do usuário. Implementar Content Security Policy (CSP).	40

A07:2021 - Falhas de Identificação e Autenticação (CSRF)	Ataques que exploram sessões de usuário para executar ações não autorizadas.	Usar Synchronizer Token Pattern (CSRF tokens). Configurar cookies SameSite, httpOnly, secure. Implementar cabeçalhos de segurança HTTP.	[40]
A09:2021 - Falhas de Logging e Monitoramento de Segurança	Falha em registrar ou monitorar eventos de segurança cruciais.	Implementar logging abrangente (tentativas de login, alterações de permissão). Centralizar logs. Monitoramento contínuo com alertas.	[36]

6. Considerações Arquiteturais

A escolha da arquitetura para o serviço de autenticação exerce uma influência direta sobre sua escalabilidade, manutenção e, crucialmente, sua segurança.

Serviço de Autenticação em Arquiteturas Monolíticas vs. Microsserviços

A decisão entre uma arquitetura monolítica e uma baseada em microsserviços tem implicações significativas para o design de um serviço de autenticação.

- **Arquitetura Monolítica:**
  - Representa um modelo tradicional de desenvolvimento de software onde todos os componentes são construídos como uma única unidade e são interdependentes.[46]
  - É geralmente mais fácil de iniciar, pois não exige um planejamento prévio extensivo.[46]
  - No entanto, com o tempo, a aplicação pode se tornar complexa e difícil de atualizar ou alterar.[46] Pequenas modificações podem ter um efeito cascata,

afetando grandes áreas da base de código.[46]

- **Arquitetura de Microsserviços:**
  - É uma abordagem arquitetural que decompõe o software em pequenos componentes ou serviços independentes, cada um executando uma única função bem definida.[46]
  - Cada microsserviço se comunica com outros serviços através de interfaces bem definidas, tipicamente APIs.[46]
  - **Vantagens:**
    - **Implantação Independente:** Os serviços podem ser implantados, atualizados, modificados ou escalados individualmente, o que facilita entregas contínuas mais rápidas.[46]
    - **Flexibilidade Tecnológica:** Permite que cada microsserviço seja implementado em uma linguagem de programação ou plataforma diferente, otimizando a escolha da tecnologia para a tarefa específica.[47]
    - **Escalabilidade:** Componentes específicos, como o serviço de autenticação, podem ser escalados independentemente para lidar com picos de demanda, sem a necessidade de escalar todo o sistema.[46]
    - **Resiliência:** Uma falha em um microsserviço tende a ser isolada, minimizando o impacto sobre outros serviços.
    - **Isolamento de Segurança:** O serviço de autenticação pode ser isolado em seu próprio ambiente, com suas próprias medidas de segurança e controle de acesso, o que minimiza o raio de explosão de uma potencial violação em outras partes do sistema.
  - **Desvantagens:**
    - **Complexidade Operacional:** Exige mais planejamento e design inicial, e adiciona complexidade no gerenciamento, monitoramento e coordenação de múltiplos serviços.[46]
    - **Consistência de Dados:** Pode apresentar desafios na manutenção da consistência eventual de dados entre os serviços.[48]
    - **Comunicação:** A comunicação entre serviços via APIs pode introduzir uma sobrecarga de rede e latência.[46]

A experiência de grandes empresas como Amazon e Netflix, que utilizam microsserviços para gerenciar aspectos críticos como a autenticação [47], sugere que projetar o serviço de autenticação como um microsserviço dedicado, mesmo que o restante da aplicação adote uma arquitetura monolítica, pode trazer benefícios substanciais a longo prazo em termos de segurança, escalabilidade e manutenibilidade. O serviço de autenticação, sendo uma funcionalidade central e de alta sensibilidade, é um candidato ideal para ser isolado. Se ele for parte de um

monólito, uma falha ou vulnerabilidade em qualquer parte do sistema pode comprometer a autenticação. Ao isolá-lo como um microsserviço, ele pode ser protegido com suas próprias políticas de segurança, escalado independentemente para lidar com picos de tráfego e atualizado sem afetar outras partes da aplicação. Essa abordagem aumenta a resiliência do sistema, limita o raio de explosão de uma violação e permite uma gestão de segurança mais focada e eficaz.

7. Design do Banco de Dados para Autenticação

O esquema do banco de dados para armazenar informações de usuário e credenciais é um componente crítico da segurança de qualquer serviço de autenticação. O design do esquema não é apenas uma questão de eficiência de dados, mas atua como uma camada de defesa ativa. A escolha de tipos de dados, a imposição de restrições (como tamanho máximo de campo e unicidade), e a configuração de controles de acesso ao banco de dados são fundamentais para a integridade e segurança.

Esquema de Tabela de Usuários

Um esquema de banco de dados define a organização lógica dos dados, incluindo nomes de tabelas, campos, tipos de dados e os relacionamentos entre essas entidades. Ele serve como a "planta" do banco de dados.[50] A tabela de usuários é a base para a autenticação e autorização, armazenando informações de login e permissões de acesso.[51]

Os campos essenciais para uma tabela de usuários segura incluem:

- id: Uma chave primária única para identificar cada usuário.
- email ou username: A credencial principal de login, que deve ser única para cada usuário.
- password_hash: O hash da senha do usuário, nunca a senha em texto simples.[4]
- password_salt: O sal único e aleatório gerado para o hash da senha.[5]
- failed_login_attempts: Um contador para rastrear tentativas de login falhas, crucial para a implementação de rate limiting.[21]

- last_failed_login_time: Um timestamp da última tentativa de login falha, usado em conjunto com failed_login_attempts.[21]
- is_locked_out: Um booleano para indicar se a conta está temporariamente bloqueada.
- lockout_until: Um timestamp que indica até quando a conta permanecerá bloqueada.
- created_at, updated_at: Timestamps para rastrear a criação e as últimas modificações dos registros de usuário.
- roles / groups: Campos ou tabelas de relacionamento para gerenciar permissões e autorizações dos usuários.[51]
- mfa_secret / mfa_enabled: Campos para armazenar segredos de MFA (se for TOTP) e o status de habilitação da MFA para o usuário.

Cada campo deve ter um tipo de dado apropriado, e regras de validação podem ser aplicadas no nível do banco de dados para garantir a integridade dos dados, como restrições de tamanho ou formato.[52]

Armazenamento Seguro de Credenciais

Conforme detalhado anteriormente, a senha deve ser armazenada no formato hash(senha + salt).[5] O sal deve ser único e aleatório para cada senha e pode ser armazenado em texto simples junto com o hash, pois sua função é garantir a unicidade do hash, não a confidencialidade do sal em si.[5]

É vital que os **controles de acesso ao banco de dados** sejam rigorosamente configurados e auditados. Bancos de dados possuem mecanismos de restrição de acesso em seus logins, como a restrição por endereço IP.[53] Infelizmente, esses mecanismos são frequentemente ignorados em projetos apressados, o que pode deixar as tabelas de credenciais vulneráveis.[53] Garantir que apenas usuários e aplicações autorizados possam acessar as tabelas de credenciais é uma camada de segurança essencial que não deve ser negligenciada.

8. Logging, Monitoramento e Tratamento de Erros

A implementação de logging abrangente, monitoramento contínuo e tratamento de erros eficaz são componentes indispensáveis para a segurança e a resiliência de um serviço de autenticação.

Security Logging e Monitoramento

O "security-logging" é um componente essencial na proteção de sistemas contra ameaças cibernéticas. Ele envolve o registro sistemático de eventos relacionados à segurança em um sistema de informação, permitindo que administradores e especialistas em segurança possam monitorar, auditar e responder a incidentes de segurança.[44]

Os benefícios do security-logging incluem a detecção precoce de atividades suspeitas, a facilitação de auditorias para conformidade regulatória e o fornecimento de evidências valiosas para a investigação de incidentes.[44] A falha em registrar ou monitorar adequadamente os eventos de segurança é uma vulnerabilidade significativa (OWASP A09:2021), pois impede a identificação de ataques em andamento, como tentativas de tomada de conta ou ataques de preenchimento de credenciais.[36]

Tipos de logs cruciais para um serviço de autenticação:

- **Logs de Autenticação:** Registram todas as tentativas de login, tanto bem-sucedidas quanto falhas. Um aumento no número de tentativas de login falhas pode indicar um ataque de força bruta.[45] O monitoramento desses logs permite a detecção de padrões suspeitos e a implementação de medidas preventivas, como o bloqueio temporário de contas.[45]
- **Logs de Autorização:** Monitoram alterações nos níveis de permissão dos usuários. São críticos para garantir que as políticas de controle de acesso sejam seguidas e para identificar mudanças não autorizadas que possam indicar uma violação de segurança interna.[45]
- **Logs de Sistema:** Capturam erros críticos no sistema operacional ou componentes de hardware que podem afetar a segurança ou disponibilidade do serviço.[45]

Melhores práticas para security-logging:

- **Centralização de Logs:** Utilizar um sistema centralizado para coletar e analisar

logs de segurança de múltiplas fontes facilita a correlação de eventos e a detecção de padrões.[44]

- **Retenção de Dados:** Manter os logs por períodos adequados, em conformidade com políticas de retenção de dados e regulamentações, é essencial para investigações forenses.[44]
- **Monitoramento Contínuo:** Implementar soluções de monitoramento que possam alertar sobre padrões suspeitos em tempo real permite uma resposta rápida a ameaças.[44]
- **Detecção de Atividades Suspeitas:** Sistemas como o Google já utilizam alertas por e-mail para administradores sobre atividades de login incomuns, como logins de locais atípicos ou em contas suspensas.[54] A capacidade de investigar essas atividades e redefinir senhas de contas comprometidas é fundamental.[54]

Tratamento de Erros

O tratamento de erros é uma prática essencial na programação e administração de sistemas, visando identificar, gerenciar e corrigir falhas durante a execução de um sistema ou aplicação.[56] É fundamental para garantir a continuidade dos serviços, a segurança dos dados e uma boa experiência do usuário.[56]

Estratégias eficazes de tratamento de erros incluem:

- **Logs de Erro:** Ferramentas valiosas que permitem monitorar e analisar problemas, identificando a causa raiz.[56]
- **Mensagens de Erro Amigáveis:** Mensagens claras e informativas para os usuários ajudam a compreender o que ocorreu e como proceder, sem revelar detalhes sensíveis que possam ser explorados por atacantes.[56]
- **Rotinas de Recuperação:** Procedimentos automatizados para restaurar o sistema a um estado funcional após uma falha.[56]
- **Autenticação Forte e Contextual:** Uma solução de autenticação robusta deve ser flexível, permitindo diferentes métodos de autenticação com base em diferentes níveis de risco. A autenticação baseada em contexto, que utiliza informações contextuais para verificar a autenticidade da identidade do usuário, é um complemento recomendado a outras tecnologias de autenticação forte.[57]

Conclusões e Recomendações

O projeto de um serviço de autenticação seguro em TypeScript para um sistema moderno transcende a mera implementação de um formulário de login/senha. Ele exige uma abordagem holística que abranja desde os fundamentos do armazenamento de credenciais até a integração de mecanismos avançados de segurança e a adoção de práticas arquiteturais robustas.

A análise demonstra uma clara evolução nos paradigmas de segurança de autenticação. A dependência exclusiva de senhas tradicionais é insuficiente diante da sofisticação dos ataques cibernéticos atuais. A adoção da **Autenticação Multifator (MFA)** e a consideração de **Chaves de Acesso (Passkeys)** para uma experiência sem senha são imperativos, não apenas como melhorias, mas como padrões de segurança essenciais. O projeto deve ser arquitetado para facilitar a integração dessas tecnologias, garantindo que o sistema permaneça resiliente a ameaças como phishing e preenchimento de credenciais.

No gerenciamento de senhas, a mudança das diretrizes do NIST e OWASP é fundamental: a prioridade deve ser o **comprimento da senha e a verificação contra listas de senhas comprometidas**, em vez de regras de complexidade arbitrárias que, ironicamente, podem comprometer a segurança. A lentidão inerente a algoritmos de hashing como Bcrypt e Argon2 não é uma falha, mas uma característica de segurança que dificulta ataques de força bruta, exigindo que as operações de hashing sejam tratadas de forma assíncrona para manter a responsividade do sistema.

Os fluxos de autenticação, especialmente o de **redefinição de senha**, são vetores de ataque críticos e devem ser protegidos com múltiplas camadas de segurança, incluindo rate limiting rigoroso, verificação de posse (OTPs) e, se aplicável, a exigência de um fator MFA. A decisão de revelar ou não a existência de um e-mail durante a redefinição de senha deve equilibrar usabilidade e segurança, com mitigações adicionais como CAPTCHAs e monitoramento de atividades suspeitas.

Do ponto de vista arquitetural, o serviço de autenticação é um **candidato ideal para ser implementado como um microsserviço**. O isolamento dessa funcionalidade crítica permite uma gestão de segurança mais focada, maior escalabilidade e resiliência, limitando o impacto de potenciais violações a um componente bem definido.

Finalmente, a **validação de entrada rigorosa** é um pilar de segurança transversal a todas as interações do usuário, prevenindo ataques como Injeção SQL e XSS. Complementarmente, um sistema abrangente de **logging e monitoramento de segurança**, com centralização de logs e alertas em tempo real, é crucial para detectar e responder rapidamente a atividades suspeitas. O tratamento de erros deve ser robusto, fornecendo mensagens claras ao usuário sem expor detalhes internos que possam ser explorados.

Em suma, um serviço de autenticação seguro em TypeScript deve ser concebido com uma mentalidade de defesa em profundidade, incorporando as melhores práticas da indústria, priorizando a experiência do usuário sem comprometer a segurança e mantendo-se adaptável às evoluções das ameaças cibernéticas.

Trabalhos citados

1. Ferramentas de autenticação para login seguro – Central de segurança do Google, acesso a agosto 13, 2025, https://safety.google/intl/pt-BR_br/authentication/
2. O Que é Authentication Flow: Entenda O Processo - LBO Dev, acesso a agosto 13, 2025, https://lbodev.com.br/glossario/o-que-e-authentication-flow/
3. Você conhece esses princípios básicos de segurança cibernética? - Keeper Security, acesso a agosto 13, 2025, https://www.keepersecurity.com/blog/pt-br/2023/09/14/do-you-know-these-cybersecurity-basics/
4. Como armazenar corretamente as senhas dos usuários - Kaspersky, acesso a agosto 13, 2025, https://www.kaspersky.com.br/blog/how-to-store-passwords/21802/
5. Boas práticas de criação, armazenamento e validação de senhas em .NET, acesso a agosto 13, 2025, https://codigosimples.net/2024/07/09/boas-praticas-de-criacao-armazenamento-e-validacao-de-senhas-em-net/
6. Sal (criptografia) – Wikipédia, a enciclopédia livre, acesso a agosto 13, 2025, https://pt.wikipedia.org/wiki/Sal_(criptografia)
7. Como o salting aprimora a segurança da senha e por que é importante usar funções de hash mais fortes? - EITCA Academy, acesso a agosto 13, 2025, https://pt.eitca.org/c%C3%ADber-seguran%C3%A7a/eitc-%C3%A9-wasf-fundamentos-de-seguran%C3%A7a-de-aplicativos-da-web/autentica%C3%A7%C3%A3o-eitc-is-wasf-fundamentos-de-seguran%C3%A7a-de-aplica%C3%A7%C3%B5es-web/introdu%C3%A7%C3%A3o-%C3%A0-autentica%C3%A7%C3%A3o/revis%C3%A3o-de-exame-introdu%C3%A7%C3%A3o-%C3%A0-autentica%C3%A7%C3%A3o/como-o-salting-aumenta-a-seguran%C3%A7a-da-senha-e-por-que-%C3%A9-importante-usar-fun%C3%A7%C3%B5es-de-hash-mais-fortes/
8. Integridade e Autenticidade: Como Algoritmos de Hash Moldam a Segurança Digital, acesso a agosto 13, 2025, https://h41stur.com/posts/hashes/

9. Qual é o melhor algoritmo para criptografar senhas? : r/golang - Reddit, acesso a agosto 13, 2025, https://www.reddit.com/r/golang/comments/m57sxf/what_is_the_best_algorithm_for_hashing_passwords/?tl=pt-br
10. NPM bcrypt - GeeksforGeeks, acesso a agosto 13, 2025, https://www.geeksforgeeks.org/node-js/npm-bcrypt/
11. bcryptjs - NPM, acesso a agosto 13, 2025, https://www.npmjs.com/package/bcryptjs
12. NIST Password Policy Guidelines 2024: What You Need to Know - Linford & Company LLP, acesso a agosto 13, 2025, https://linfordco.com/blog/nist-password-policy-guidelines/
13. Authentication - OWASP Cheat Sheet Series, acesso a agosto 13, 2025, https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
14. O NIST não recomenda mais requisitos de senha como caracteres especiais. Por que todos ainda estão presos a esse requisito obsoleto? - Reddit, acesso a agosto 13, 2025, https://www.reddit.com/r/UXDesign/comments/167e9hj/the_nist_no_longer_recommends_password/?tl=pt-br
15. www.nist.gov, acesso a agosto 13, 2025, https://www.nist.gov/cybersecurity/how-do-i-create-good-password#:~:text=NIST%20guidance%20recommends%20that%20a,combinations%20of%2015%20lowercase%20letters.
16. Gerenciamento de Senhas: As Melhores Práticas para Manter suas Credenciais Seguras, acesso a agosto 13, 2025, https://www.reitec.net.br/gerenciamento-de-senhas-seguras/
17. 8 práticas recomendadas de gerenciamento de senhas para funcionários - Keeper Security, acesso a agosto 13, 2025, https://www.keepersecurity.com/blog/pt-br/2025/06/06/password-management-best-practices-for-employees/
18. Authentication | NestJS - A progressive Node.js framework, acesso a agosto 13, 2025, https://docs.nestjs.com/security/authentication
19. Como evitar ataques de força bruta - MetaCompliance, acesso a agosto 13, 2025, https://www.metacompliance.com/pt/blog/cyber-security-awareness/how-to-avoid-brute-force-attacks
20. O que é um ataque de força bruta? - F5, acesso a agosto 13, 2025, https://www.f5.com/pt_br/glossary/brute-force-attack
21. Best way to limit (and record) login attempts - Stack Overflow, acesso a agosto 13, 2025, https://stackoverflow.com/questions/580534/best-way-to-limit-and-record-login-attempts
22. Melhores práticas para o fluxo "esqueci minha senha"? : r/webdev - Reddit, acesso a agosto 13, 2025, https://www.reddit.com/r/webdev/comments/1bgs7bi/best_practices_for_forgot_

password_flow/?tl=pt-br
23. APIs de identidade autônoma: Fluxo Esqueci minha senha autônoma para clientes e parceiros - Salesforce Help, acesso a agosto 13, 2025, https://help.salesforce.com/s/articleView?id=sf.remoteaccess_headless_forgot_password_flow.htm&language=pt_BR&type=5
24. O que é Autenticação Multifatorial (MFA)? - Software Check Point, acesso a agosto 13, 2025, https://www.checkpoint.com/pt/cyber-hub/network-security/what-is-multi-factor-authentication-mfa/
25. Tipos de autenticação multifator (MFA) - Keeper Security, acesso a agosto 13, 2025, https://www.keepersecurity.com/blog/pt-br/2023/06/27/types-of-multi-factor-authentication-mfa/
26. OAuth 2.0 and OpenID Connect overview - Okta Developer, acesso a agosto 13, 2025, https://developer.okta.com/docs/concepts/oauth-openid/
27. OpenID Connect | Sign in with Google, acesso a agosto 13, 2025, https://developers.google.com/identity/openid-connect/openid-connect
28. JSON Web Tokens - Auth0, acesso a agosto 13, 2025, https://auth0.com/docs/secure/tokens/json-web-tokens
29. JSON Web Token Introduction - jwt.io, acesso a agosto 13, 2025, https://jwt.io/introduction
30. Jsonwebtoken NPM Nodejs. JSON Web Token (JWT) is a standard for... | by Barath Kumar, acesso a agosto 13, 2025, https://medium.com/@barathkumark/jsonwebtoken-npm-nodejs-db80fcf070f8
31. Introdução a JWT: autenticação segura em aplicações Typescript (React e NodeJS API) | by Robson Fernando Trasel de Souza | Medium, acesso a agosto 13, 2025, https://medium.com/@robson.trasel/introdu%C3%A7%C3%A3o-a-jwt-autentica%C3%A7%C3%A3o-segura-em-aplica%C3%A7%C3%B5es-typescript-react-e-nodejs-api-477ce1e4208d
32. jwt npm - GeeksforGeeks, acesso a agosto 13, 2025, https://www.geeksforgeeks.org/node-js/jwt-npm/
33. OWASP Top 10: Cheat Sheet of Cheat Sheets - Oligo Security, acesso a agosto 13, 2025, https://www.oligo.security/academy/owasp-top-10-cheat-sheet-of-cheat-sheets
34. O que é OWASP? | OWASP Top 10 | Cloudflare, acesso a agosto 13, 2025, https://www.cloudflare.com/pt-br/learning/security/threats/owasp-top-10/
35. O que é OWASP? As 10 principais vulnerabilidades e riscos da OWASP - F5, acesso a agosto 13, 2025, https://www.f5.com/pt_br/glossary/owasp
36. OWASP Top 10: Security Logging and Monitoring Failures - IONIX, acesso a agosto 13, 2025, https://www.ionix.io/guides/owasp-top-10/security-logging-and-monitoring-failures/
37. OWASP Top 10 Security Logging and Monitoring Failures Explained, acesso a agosto 13, 2025,

https://www.securityjourney.com/post/owasp-top-10-security-logging-and-monitoring-failures-explained
38. Preventing SQL injection attacks in Node.js - Snyk, acesso a agosto 13, 2025, https://snyk.io/blog/preventing-sql-injection-attacks-node-js/
39. Preventing SQL injection in Node.js - Stack Overflow, acesso a agosto 13, 2025, https://stackoverflow.com/questions/15778572/preventing-sql-injection-in-node-js
40. Security Best Practices in Angular 19 and Node.js with TypeScript | by Amir Saeed - Medium, acesso a agosto 13, 2025, https://amir-saeed.medium.com/security-best-practices-in-angular-19-and-node-js-with-typescript-e654bbb09bcf
41. xss - NPM, acesso a agosto 13, 2025, https://www.npmjs.com/package/xss
42. CSRF | NestJS - A progressive Node.js framework, acesso a agosto 13, 2025, https://docs.nestjs.com/security/csrf
43. How to protect Node.js apps from CSRF attacks - Snyk, acesso a agosto 13, 2025, https://snyk.io/blog/how-to-protect-node-js-apps-from-csrf-attacks/
44. security logging - Definição e Como Funciona | DevLingo, acesso a agosto 13, 2025, https://www.devlingo.com.br/termos/security-logging
45. A Importância dos Logs na Segurança da Informação: Tipos e Benefícios para a Cibersegurança - GRC | IT Compliance | Auditoria de TI | Offensive Security - Tiago Souza, acesso a agosto 13, 2025, https://tiagosouza.com/importancia-logs-seguranca-informacao/
46. Monolítico x microsserviços — Diferença entre arquiteturas de desenvolvimento de software, acesso a agosto 13, 2025, https://aws.amazon.com/pt/compare/the-difference-between-monolithic-and-microservices-architecture/
47. Microsserviço — Vantagens e “desvantagens”!!! | by Edu Silveira - Medium, acesso a agosto 13, 2025, https://medium.com/xp-inc/microsservi%C3%A7o-vantagens-e-desvantagens-c89f227ef1f9
48. Vantagens e desvantagens de uma arquitetura microservices - SlideShare, acesso a agosto 13, 2025, https://pt.slideshare.net/slideshow/vantagens-e-desvantagens-de-uma-arquitetura-microservices/52206383
49. O que é a arquitetura de microsserviços? - Google Cloud, acesso a agosto 13, 2025, https://cloud.google.com/learn/what-is-microservices-architecture?hl=pt-BR
50. O que é um esquema de banco de dados? - IBM, acesso a agosto 13, 2025, https://www.ibm.com/br-pt/think/topics/database-schema
51. Qual é a tabela de usuários? - ServiceNow Community, acesso a agosto 13, 2025, https://www.servicenow.com/community/brazil-snug/qual-%C3%A9-a-tabela-de-usu%C3%A1rios/ba-p/2280585
52. Criar uma regra de validação para validar dados em um campo - Suporte da Microsoft, acesso a agosto 13, 2025, https://support.microsoft.com/pt-br/topic/criar-uma-regra-de-valida%C3%A7%C

3%A3o-para-validar-dados-em-um-campo-b91c6b15-bcd3-42c1-90bf-e3a0272e988d
53. Segurança de banco de dados (autenticação e permissões) - DBA, acesso a agosto 13, 2025, https://dba.com.br/2021/07/segurancabd-usuarios
54. Sobre os alertas do administrador que indicam atividade de login suspeita, acesso a agosto 13, 2025, https://support.google.com/a/answer/7102416?hl=pt-BR
55. Investigar atividades suspeitas na sua conta - Ajuda da Conta do Google, acesso a agosto 13, 2025, https://support.google.com/accounts/answer/140921?hl=pt
56. O que é Tratamento de erros - H2IT Soluções de TI, acesso a agosto 13, 2025, https://h2it.com.br/glossario/o-que-e-tratamento-de-erros-seguranca-redes-ti/
57. Melhores Práticas de Autenticação - Thales CPL, acesso a agosto 13, 2025, https://cpl.thalesgroup.com/pt-pt/access-management/strong-authentication-best-practices