

# Avaliação de conhecimentos em Desenvolvimento de Software

---

## Sistema Serverless para Compartilhamento Seguro de Senhas

### AVALIAÇÃO TÉCNICA

- **Frontend (produção):** <https://bryansilva.vercel.app/>
- **Backend:** AWS Lambda + API Gateway + DynamoDB (TTL)
- **Design:** Canva(<https://www.canva.com>)

## 1 INTRODUÇÃO

A solução propõe um mecanismo serverless para compartilhamento seguro de senhas por meio de URL temporária, com controle de expiração temporal e limitação de visualizações. O sistema foi projetado com foco em segurança criptográfica, controle atômico de concorrência e consistência de experiência do usuário.

## 2. ARQUITETURA

A solução foi projetada utilizando arquitetura **serverless na AWS**, eliminando a necessidade de gerenciamento de infraestrutura e permitindo escalabilidade automática sob demanda.

### 2.1 VISÃO GERAL

O fluxo operacional ocorre da seguinte forma:

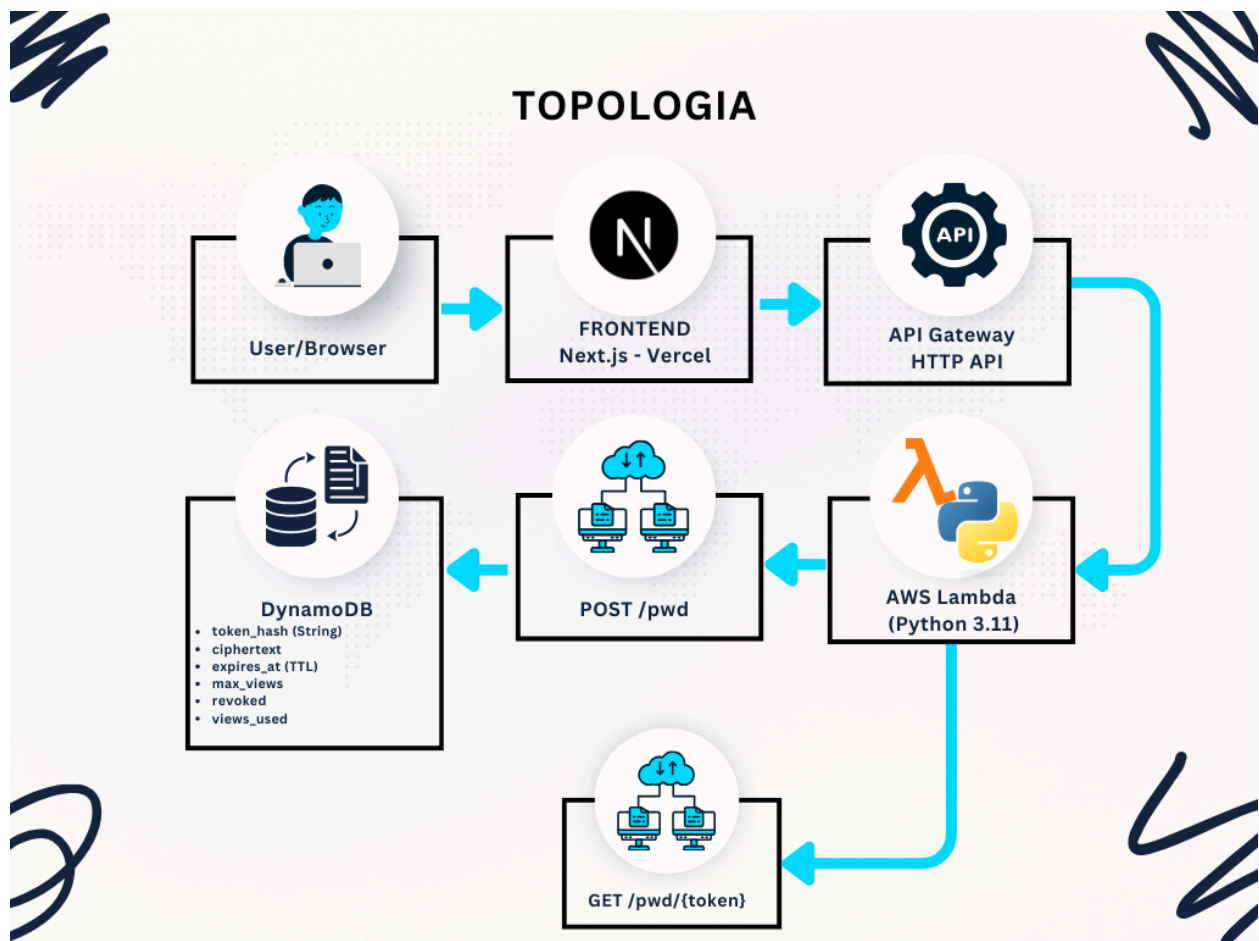
1. O usuário acessa o frontend desenvolvido em Next.js, hospedado na Vercel.
2. O frontend consome uma API HTTPS exposta pelo Amazon API Gateway (HTTP API).
3. O API Gateway invoca funções AWS Lambda (Python 3.11) responsáveis pela criação e leitura de segredos.

4. As funções interagem com uma tabela no Amazon DynamoDB para persistência segura.
5. Logs e métricas são enviados automaticamente ao Amazon CloudWatch.

**Create: Frontend** → **POST /pwd** → Lambda cria *token*, criptografa e persiste → retorna *token* para compor URL.

**View: Frontend** (**/visualizar/{token}**) → **GET /pwd/{token}** → Lambda valida (tempo/views/revogação), consome 1 view e retorna segredo (se permitido).

## 2.2 FIGURA 1 – TOPOLOGIA E SOLUÇÃO SERVERLESS



- Frontend desacoplado do backend
- API Gateway como endpoint
- Lambda como camada de lógica
- DynamoDB com TTL para expiração automática

## 2.3 Componentes

### Frontend

- Next.js (React)
- Hospedagem: Vercel
- Deploy contínuo via push (CI/CD automático)

### API

- Amazon API Gateway (HTTP API)
- Integração direta com Lambda
- Configuração de CORS

### ComputeAWS Lambda (Python 3.11)

- Funções independentes para criação e leitura
- Execução stateless

### Persistência

Tabela DynamoDB (modo on-demand):

- token\_hash (Partition Key – String)
- ciphertext (String)
- expires\_at (Number – TTL)
- max\_views (Number)
- views\_used (Number)
- revoked (Boolean)

O token original não é armazenado, apenas seu hash criptográfico.

### Expiração

- Validação síncrona na Lambda
- Limpeza automática assíncrona via DynamoDB TTL

## 3 FRONTEND (UX, CONSISTÊNCIA E REDUÇÃO DE AMBIGUIDADE)

Uma decisão no desenvolvimento de *frontend* foi priorizar a consistência da experiência do usuário, desvinculando-a dos detalhes técnicos de implementação. Para o usuário, a origem da criação da senha (seja no *frontend* ou no *backend*) é irrelevante. O fundamental é que, para o *backend*, qualquer credencial contida no campo "**sended\_password**" é processada como a senha do usuário. Essa abordagem simplifica as expectativas de comunicação e mitiga a complexidade da lógica de processamento.

### 3.1 UX DEFENSIVA: OCULTAR CUSTOMIZAÇÕES QUANDO NÃO FAZEM SENTIDO

A interface foi projetada para reagir dinamicamente ao estado do campo de senha.

- Quando o usuário digita manualmente uma senha, as opções de customização de geração (letras, números, símbolos e tamanho) são ocultadas automaticamente, pois deixam de ter efeito funcional.
- Quando o usuário opta por gerar uma senha pelo ícone de dado (🎲), a senha gerada é inserida diretamente no campo de input, passando a ser tratada como senha definida. Nesse momento, as opções de customização também são ocultadas, mantendo consistência de comportamento.
- Caso o usuário apague o conteúdo do input, a interface restaura automaticamente as opções de customização, permitindo uma nova geração configurável.

### 3.2 TRÊS CENÁRIOS SUPORTADOS DE FORMA UNIFORME

A Figura 1 detalha os três cenários de criação de senha suportados e o comportamento resultante do *backend*.

**Tabela 1** - Cenários de criação de senha e comportamento do *backend*.

Cenário	Ação do usuário	O que o frontend envia	Comportamento do backend
A) Senha digitada	Usuário digita a senha	Envia <b>sended_password</b>	Criptografa e persiste a senha

			recebida
B) Gerar no <i>frontend</i>	Usuário não digita e clica no dado	Gera localmente e envia como <code>sended_password</code>	Criptografa e persiste (igual ao cenário A)
C) Gerar no <i>backend</i>	Usuário não digita, mas configura política e salva	NÃO envia <code>sended_password</code> ; envia política	Gera a senha com base na política e persiste

Com isso, o *backend* permanece simples: ele só distingue se existe ou não "`sended_password`" no *payload*. Se existe, trata como *input* final; se não existe, aplica a política de geração e produz o segredo.

### 3.3 VALIDAÇÕES E FEEDBACK DE UI

- Se o usuário tentar gerar senha sem marcar nenhum *charset* (letras/dígitos/símbolos), a UI apresenta *feedback* imediato.
- Campos de expiração e limite de visualizações são validados antes do `POST` (evita criar *links* inválidos).
- Ao criar, a UI exibe URL pronta para copiar e opção de visualizar imediatamente.
- Na visualização, o estado de carregamento e erros (404/410) são apresentados de forma clara e orientada à ação (voltar ao início).
- Proteção contra dupla requisição no React Strict Mode (evita consumir *views* duas vezes por efeito colateral de *render*).

### 3.4 EDUCAÇÃO CONTEXTUAL: CARD “COMO FUNCIONA?”

Para reduzir ambiguidades e evitar interpretações incorretas sobre o funcionamento do sistema, foi implementado um card explicativo intitulado “**Como funciona?**” diretamente na interface principal.

Esse componente atua como mecanismo de onboarding leve e contextual, explicando:

- Os três modos possíveis de criação de senha;
- O comportamento do sistema após a criação;
- As regras de visualização limitada;
- O conceito de expiração automática do segredo.

O objetivo não é expor detalhes técnicos de implementação, mas sim esclarecer o fluxo funcional sob a ótica do usuário final.

## 4 BACKEND (CONTRATO E ATOMICIDADE)

### 4.1 POST /PWD — CRIAÇÃO DO SEGREDO E TOKEN

Responsabilidades principais: validar entrada (expiração e limite), gerar *token* público seguro, calcular *hash* SHA-256 para armazenamento, criptografar o segredo com Fernet e persistir no DynamoDB com TTL.

### 4.2 GET /PWD/{TOKEN} — CONSUMO E REVELAÇÃO CONTROLADA

A leitura do segredo executa um *update* condicional atômico no DynamoDB para consumir exatamente 1 visualização somente se o item ainda estiver válido (não expirado, não excedeu *views*). Se o limite for atingido após o consumo, o item é removido imediatamente.

#### 4.2.1 SÊMANTICA HTTP

- **200** OK: segredo retornado e *view* consumida.
- **404** Not Found: *token* inexistente (nunca criado ou já removido).
- **410** Gone: expirado por tempo, limite de *views* excedido ou revogado.

### 4.3 CONFIGURAÇÃO DE ENDPOINT E VARIÁVEIS DE AMBIENTE

O backend é exposto publicamente por meio do Amazon API Gateway, sendo acessado exclusivamente via HTTPS.

O frontend consome a API através da variável de ambiente:

- `NEXT_PUBLIC_API_BASE_URL`

Em ambiente de produção, essa variável aponta para o endpoint provisionado pelo API Gateway:

- `https://5y0n5f4ill.execute-api.us-east-1.amazonaws.com`

## 5 SEGURANÇA

As medidas de segurança implementadas foram projetadas para mitigar vazamento de dados, acesso indevido e condições de corrida (race conditions), garantindo confidencialidade e integridade do segredo.

### 5.1 Confidencialidade do Segredo

- **Criptografia em repouso (Fernet)**  
O segredo é criptografado antes da persistência. O DynamoDB nunca armazena dados em plaintext.
- **Hash do token público (SHA-256)**  
O token enviado ao usuário não é armazenado diretamente. Apenas seu hash criptográfico é persistido como chave de busca, mitigando exposição caso haja vazamento da base.
- **Transporte seguro (HTTPS)**  
Todo tráfego ocorre via TLS através do API Gateway.

### 5.2 Controle de Acesso e Consumo Seguro

- **Update condicional atômico (ConditionExpression)**  
O consumo de visualizações ocorre por meio de operação atômica no DynamoDB, garantindo que apenas uma requisição consiga consumir cada view válida.
- **Delete imediato ao atingir `max_views`**  
Após o último consumo permitido, o item é removido imediatamente, mitigando tentativas de replay.
- **Validação síncrona de expiração**  
A Lambda valida o tempo de expiração antes da leitura, não dependendo exclusivamente do mecanismo assíncrono de TTL.

### 5.3 Expiração e Minimização de Superfície de Ataque

- **DynamoDB TTL (`expires_at`)**  
Garante remoção automática de registros após o prazo definido, reduzindo retenção desnecessária de dados sensíveis.

## 6 MELHORIAS FUTURAS

Para ambientes de maior criticidade, recomenda-se:

- Secrets Manager + rotação de chave
- Alarmes e métricas
- Integrar AWS WAF para mitigação de ataques automatizados

## 7 CONCLUSÃO

A solução atende aos requisitos do desafio, permitindo ao usuário digitar ou gerar senhas com base em políticas de complexidade, definindo expiração e limite de visualizações. É gerada uma URL única para acesso ao segredo, que é removido após o uso ou prazo.

O desenvolvimento focou na organização das responsabilidades entre frontend e backend, mantendo o contrato simples. Foram aplicadas boas práticas de segurança (criptografia do segredo, armazenamento de hash do token público e controle atômico de visualizações no DynamoDB) e a interface foi pensada para clareza.

De modo geral, o projeto foi desenvolvido considerando a segurança, a consistência e a experiência do usuário.