CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

FATEC DE PRAIA GRANDE

Tecnologia de Desenvolvimento de Software Multiplataforma

Daniel de Abreu Macedo Neto
Gabriel Venancio Nunes
Guilherme Rocha Bezerra

HidroAlerta Baixada

Praia Grande - SP

2025

Daniel de Abreu Macedo Neto
Gabriel Venancio Nunes
Guilherme Rocha Bezerra

HidroAlerta Baixada

Projeto Interdisciplinar apresentado na disciplina de Laboratório de Desenvolvimento Web como requisito para conclusão do 4º. Módulo do Curso de Tecnologia de Desenvolvimento de Software Multiplataforma**.**

Orientadora: Prof. Eulaliane Aparecida Gonçalves

Praia Grande - SP
2025

SUMÁRIO

1. INTRODUÇÃO

Nos últimos anos, a região tem enfrentado uma crescente escassez de água, especialmente durante a alta temporada. Esta situação tem se agravado devido às intensas ondas de calor que se tornaram cada vez mais frequentes. Os impactos dessa instabilidade no fornecimento de água afetam diretamente a qualidade de vida dos moradores, bem como as atividades comerciais da região.

Para enfrentar este desafio, nosso projeto propõe a criação de uma plataforma onde os usuários podem marcar em um mapa as áreas afetadas pela falta ou instabilidade no fornecimento de água. Caso um usuário do sistema relate problemas em relação à distribuição de água, notificamos 5% dos usuários na cidade, se esses confirmarem o problema seguimos com a notificação geral para a cidade. Com essa ferramenta, os usuários poderão adotar medidas para se prepararem com antecedência, assim gerenciando de forma eficaz a escassez de água, minimizando os impactos negativos.

A plataforma visa atender um público diversificado, composto por moradores de todas as idades e classes sociais, que vivenciam os desafios diários decorrentes da inconsistência no fornecimento de água. Além disso, busca auxiliar comerciantes e empresários locais, cujas atividades são impactadas pela falta de água, garantindo a continuidade de seus negócios e a manutenção da economia local.

1.2. Objetivos

O objetivo principal do projeto é criar uma plataforma digital que ajude a identificar e mapear as áreas que sofrem com a falta ou irregularidade no fornecimento de água, tanto para casas quanto para empresas.

A plataforma permitirá que usuários forneçam informações sobre os problemas de água que estão enfrentando. Um sistema colaborativo permitirá que usuários relatem problemas em relação a distribuição de água, outros usuários próximos podem confirmar a reclamação e uma notificação será enviada para todos os usuários da cidade.

1.3 Problematização

A escassez de água desencadeia uma série de problemas que impactam diretamente a saúde da população. A falta desse recurso essencial impede a prática adequada de atividades básicas de higiene pessoal, como tomar banho e lavar as mãos, expondo os moradores a um maior risco de doenças infecciosas. A higiene dos alimentos também é severamente prejudicada, pois a lavagem de frutas, verduras e utensílios de cozinha torna-se impraticável, aumentando significativamente o risco de intoxicação alimentar.

A ausência de água compromete o funcionamento do sistema de esgoto, transformando banheiros com acúmulo de dejetos em vetores de contaminação. Ambientes públicos, com saneamento precário, tornam-se focos de propagação de doenças, colocando em risco a saúde de toda a população.

O aumento da incidência de doenças decorrentes da falta de água leva a uma maior procura por atendimento médico, sobrecarregando o sistema de saúde da região. Essa sobrecarga pode resultar em filas de espera mais longas, falta de leitos e dificuldade de acesso a medicamentos, agravando ainda mais a situação de saúde da população.

1.4. Justificativa

A falta da água compromete a higiene pessoal e o saneamento básico, aumentando o risco de doenças e prejudicando a qualidade de vida. Os métodos tradicionais de monitoramento e gestão da água se mostram insuficientes para lidar com essa realidade, dificultando a tomada de decisões eficazes e a comunicação entre usuários e autoridades.

Diante desse cenário, a plataforma proposta surge como uma ferramenta essencial para enfrentar a crise hídrica. Ao permitir que os usuários se preparem antecipadamente para instabilidades ou interrupções no fornecimento de água, a plataforma minimiza os impactos negativos da escassez, garantindo melhor qualidade de vida para os moradores e impulsionando a eficiência das atividades comerciais.

Além disso, a plataforma capacita os cidadãos a participarem ativamente da gestão dos recursos hídricos. Ao fornecerem dados concretos sobre a escassez na região, os usuários contribuem para o monitoramento e o registro da falta da água, permitindo que as autoridades sejam cobradas e tomem medidas proativas para garantir o abastecimento contínuo.

A plataforma não se limita a alertar sobre a escassez, mas também promove a transparência e a colaboração entre a população e as autoridades. Ao centralizar as informações sobre a situação do abastecimento, a plataforma facilita a comunicação e a tomada de decisões conjuntas, fortalecendo a gestão dos recursos hídricos e garantindo o acesso à água para todos.

1.5. Metodologia

A metodologia do projeto é baseada nos princípios das Metodologias Ágeis, que buscam responder rapidamente às mudanças e garantir entregas incrementais do projeto. As Metodologias Ágeis seguem o Manifesto Ágil (AGILE MANIFESTO, 2001), que define:

4 Valores:

1. Indivíduos e interações mais que processos e ferramentas.
2. Software em funcionamento mais que documentação abrangente.
3. Colaboração com o cliente mais que negociação de contratos.
4. Responder a mudanças mais que seguir um plano.

12 Princípios:

1. A maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2. Requisitos mudem, mesmo tarde no desenvolvimento.
3. O software deve ser entregue funcionando frequentemente, com preferência por prazos curtos.
4. Pessoas de negócios e desenvolvedores devem trabalhar juntos diariamente.
5. Construa projetos em torno de indivíduos motivados, forneça o ambiente e suporte necessários.

6. O método mais eficiente de transmitir informações para a equipe é por conversas face a face.
7. Software funcionando é a principal medida de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável.
9. Atenção contínua à excelência técnica e bom design.
10. A simplicidade é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizadas.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz.

O SCRUM será utilizado como metodologia principal para o gerenciamento e organização do desenvolvimento do projeto. Ele se baseia em ciclos iterativos e incrementais chamados Sprints, cada um com as seguintes etapas:

- **Planejamento da Sprint:** Definição de tarefas e metas para o ciclo.
- **Reuniões diárias (Daily Scrum):** Comunicação rápida para identificar bloqueios e alinhar o progresso.
- **Revisão da Sprint:** Apresentação dos resultados ao final de cada ciclo.
- **Retrospectiva da Sprint:** Reflexão sobre o que funcionou bem e o que precisa melhorar.

Para a gestão visual das tarefas, será utilizado o Kanban, que ajuda a gerenciar o fluxo de trabalho com base em um quadro dividido em colunas, como:

- **Product Backlog**: Tarefas que serão realizadas ao longo de todo o projeto.
- **N ° Sprint:** Itens que serão entregues no sprint N.
- **Fazendo:** Tarefas em andamento.
- **Concluído:** Tarefas finalizadas.

1.6. Organização do Trabalho

Em introdução descrevemos o tema geral do projeto, contextualizando o problema da interrupção na distribuição de água na baixada santista quando há um alto fluxo de turistas na cidade, introduzindo como o objetivo oferecer uma visibilidade para a população em relação a esse problema.

Na seção de descrição do sistema descrevemos a solução, dando enfoque ao público afetado pelo nosso sistema, contando suas dores e como o nosso projeto oferecer saná-las via um software de avisos em relação a problemas de abastecimento.

2. Descrição do sistema

O sistema proposto é um software colaborativo que se empodera do público para realizar amostragens que relatem a disponibilidade de água em localidades da baixada santista. Utilizando um sistema de notificação, o sistema agrega os usuários em grupos que melhor descrevem a sua localidade e interesses, notificando-os sobre uma possível falha no sistema de distribuição, engajando o usuário diretamente para obter uma confirmação sobre a falta do serviço em sua localidade.

2.1 Descrição do Problema

A baixada santista sofre um problema crônico em relação a sua infraestrutura, em específico com a incapacidade de suportar um grande fluxo de turistas, com interrupções frequentes na distribuição de água. Esse problema é potencializado na temporada de verão, em especial quando temos uma alta concentração de feriados nesse período. É um problema que não temos panorama de mudança e que a população fica à mercê da volatilidade do serviço sem visibilidade do problema.

Existem dois grupos de enfoque que o projeto melhor descreve: Turistas que desejam ter uma visibilidade sobre a condição da disponibilidade de água na região para que ele possa melhor decidir qual cidade ele deseja visitar; Moradores que desejam receber um aviso para que ele possa planejar melhor o seu dia.

O impacto do sistema estaria em oferecer uma visibilidade de uma forma centralizada e mais ágil, já que a forma de você obter essa informação hoje é através de portais de notícias que não conseguem notificar essa classe de problema de maneira rápida o suficiente. Outro impacto estaria relacionado a forma de como a distribuição de água é interrompida, não é um processo constante e uniforme, pontos de uma cidade, sejam eles bairros ou residências possuem condições diferentes, fazendo que a falta do serviço de distribuição de água seja distribuído de forma desigual. Com isso, usuários que moram em localidades que relataram o problema mas não identificaram qualquer interrupção em sua residência ou bairro, podem diminuir a sua demanda para diminuir a carga no serviço, otimizando o tempo em que a água possa estar disponível em alta pressão para os residentes

mais afetados. O sistema se torna não só um notificador como também um vetor de conscientização.

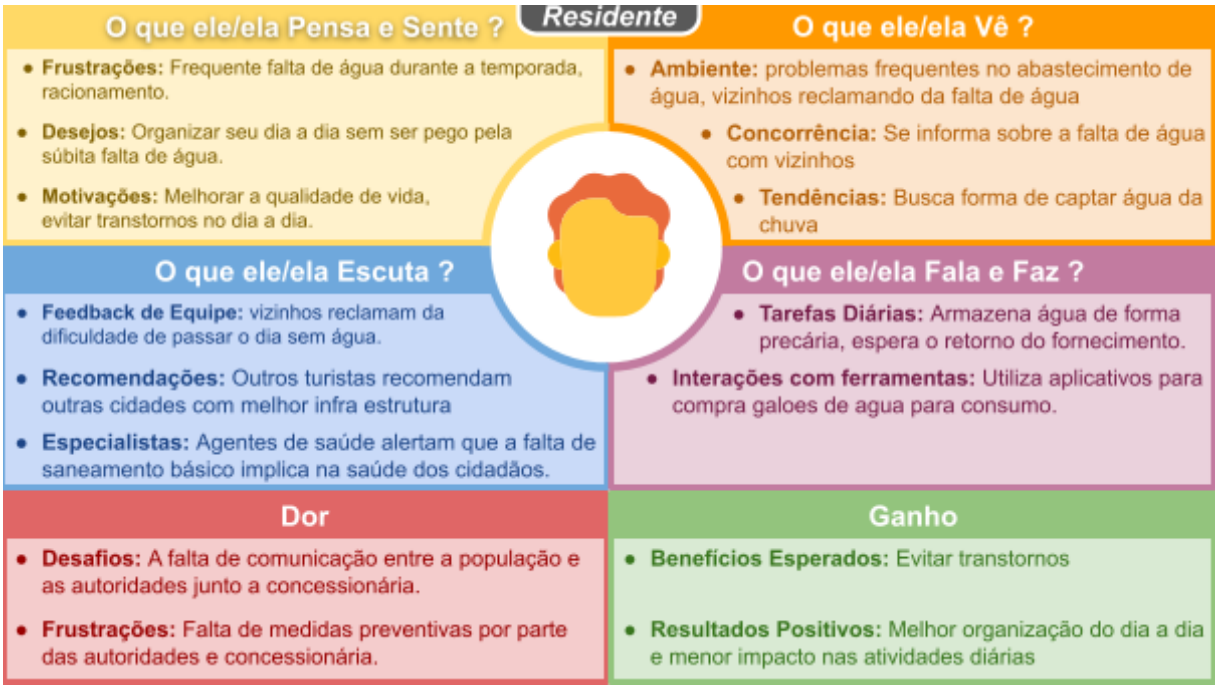
Dado a ineficácia da política pública em aprimorar a infraestrutura para ter a capacidade de receber mais turistas sem experienciar uma interrupção no serviço de distribuição de água, a melhor solução para comportar os públicos afetados seria através da transparência do problema, para que o usuário possa planejar as suas ações com a maior quantidade de informações possíveis. O nosso sistema provê exatamente isso, oferecendo uma visão mais ampla para a população.

2.2 Principais Envolvidos e suas características

O sistema é dividido em dois grupos de usuários:

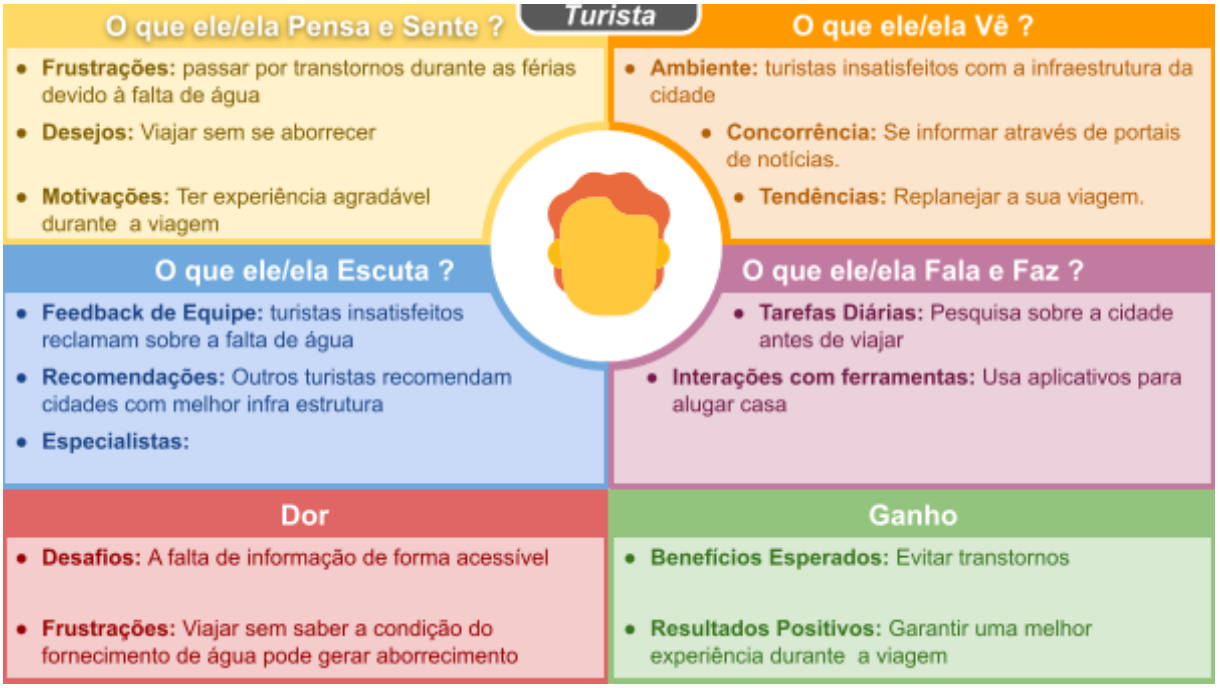
- **Residente:** Indivíduo que mora na baixada santista e deseja ter informações sobre a disponibilidade de água no local, para poder planejar o seu dia a dia, seja ele mantendo a higiene pessoal até na gerência de um comércio que esse residente possa ter.
- **Turista:** Indivíduo que não mora na região mas planeja fazer visitas ou ficar alojado em uma das cidades da baixada santista e necessita ter visibilidade do serviço para decidir o tempo em que ele ficará na região ou que cidade.

Figura 1 - Mapa de Empatia do Residente



Fonte: Do próprio autor, 2025.

Figura 2 - Mapa de Empatia do Turista



Fonte: Do próprio autor, 2025.

Os desenvolvedores do sistema e suas funções são:

- **Daniel de Abreu:** Product Manager/Documentação.
- **Gabriel Venancio Nunes:** Desenvolvedor back-end.
- **Guilherme Rocha:** Desenvolvedor front-end

2.3 Regras de negócio

1. Registro de Ocorrência
    a. Qualquer usuário autenticado pode relatar um problema de falta de água informando a localização e um breve relato opcional.
    b. Cada relato é vinculado ao município e bairro do usuário.
    c. Relatos duplicados (mesmo local, curto intervalo de tempo) são agrupados automaticamente.
2. Validação da Ocorrência
    a. Ao receber um relato, o sistema seleciona aleatoriamente 5% dos usuários ativos do mesmo município para confirmação.
    b. Esses usuários recebem uma notificação solicitando que confirmem ou neguem o problema.
    c. A validação ocorre em um tempo limite de X minutos para garantir resposta rápida.
3. Confirmação e Divulgação
    a. Se a maioria dos usuários consultados confirmar o problema, o sistema envia uma notificação geral para todos os usuários do município informando a ocorrência da falta de água.
    b. Se a maioria não confirmar ou houver baixa resposta, o relato não gera notificação ampla, mas fica registrado no sistema para análise.
4. Gerenciamento e Atualização
    a. Usuários podem marcar que o problema foi resolvido, removendo a notificação ativa.
    b. Novas confirmações podem prolongar a notificação caso o problema persista.
5. Restrições e Segurança

a. Limitações podem ser impostas para evitar spam ou uso indevido (ex.: um usuário só pode reportar uma ocorrência a cada X horas).
b. Usuários que enviam relatos falsos repetidamente podem ser temporariamente impedidos de reportar novos problemas.

3. Requisitos do Sistema

Requisitos são especificações, onde são definidas a priori, descrevendo as capacidades do sistema e como ele deve se comportar ao realizar tais funções. Os requisitos são separados em dois grupos, funcionais e não funcionais. Os requisitos funcionais tange às funções essenciais do sistema, enquanto os não funcionais incluem requisitos de desempenho, segurança, qualidade e usabilidade.

Cada requisito é dado uma prioridade que caracteriza o quão importante esse requisito é, uma prioridade alta significa algo vital ao sistema, uma prioridade média são os requisitos de que sem eles o sistema funciona porém com deficiências, sejam elas de usabilidade, segurança ou desempenho, enquanto uma prioridade baixa caracteriza um requisito opcional.

3.1 Requisitos funcionais

Identificador	Descrição	Critério de Aceitação	Prioridade
RF01 **Cadastro**	Permitir que os usuários se cadastrem, criando uma conta.	● O usuário pode preencher o formulário, com email e senha; ● Validação do nome de usuário e email; ● Os usuários podem alterar o seu próprio cadastro, incluindo excluir; ● Opção de habilitar autenticação de dois fatores utilizando TOTP.	Alta

RF02 **Login**	Permite que os usuários realizem um login utilizando email e senha após o processo de cadastramento.	● O sistema valida as credenciais de acesso. ● O sistema apresenta onde o usuário insere as informações necessárias para realizar a autenticação de dois fatores, caso o usuário tenha habilitado essa opção. ● O sistema redireciona o usuário para a página inicial.	Alta
RF03 **Recuperar senha**	Permite que o usuário recupere a senha através do email.	● Na tela de login o sistema mostra uma opção de recuperação de senha. ● Ao clicar na opção o sistema redireciona o usuário a uma tela de recuperação, pedindo pelo seu email. ● Ao fornecer o email um link de recuperação será enviado ao email. ● Ao clicar no link o usuário será redirecionado para uma tela onde ele criará uma nova senha e confirmará a mesma.	Alta
RF04 **Reportar Ocorrência**	Permite que o usuário reporte uma ocorrência sobre a interrupção do fornecimento de água.	● O usuário na tela de registro da ocorrência deve informar o município e bairro. ● O local da ocorrência fica salvo, assim na próxima ocorrência não	Alta

		será necessário colocar os dados, mas poderá ser alterado caso o usuário se mude.	
RF05 **Notificar / Validar Ocorrência**	Pede confirmação do usuário se ele está experienciando uma interrupção no sistema de distribuição de água.	● O sistema seleciona 5% dos usuários no sistema que moram em uma localidade onde foi registrado uma ocorrência. ● Uma notificação é enviada a esses usuários onde eles confirmam ou negam a interrupção do serviço.	Alta
RF06 **Resolver Ocorrência**	O usuário pode informar uma ocorrência como resolvida	● Os usuários podem marcar uma interrupção em relação ao sistema de distribuição de água como resolvida. Mudando o estado do município no mapa de ocorrência.	Média
RF07 **Informar Ocorrência**	Os usuários recebem um alerta quando o sistema confirma que há uma interrupção no sistema de distribuição de água no local onde eles moram.	● O alerta é enviado como notificação para os usuários no site; ● Além da notificação no site o alerta é enviado ao email dos usuários cadastrados que moram na localidade.	Alta
RF08 **Mapa de Ocorrência**	O sistema fornece um mapa relatando o estado do sistema de distribuição em cada município.	● Um mapa descrevendo o estado de cada município é exibido na página inicial do site, para usuários	Alta

		cadastrados e não cadastrados. ● Cada município é descrito por uma de três cores, verde se não há problema, amarelo se o sistema está validando notificações e vermelho quando o problema foi confirmado.	

3.2 Requisitos não funcionais

Identificador	Descrição	Critério de Aceitação	Prioridade
RNF01 **Período de validação**	A rodada de validação é um conjunto de notificações que são enviadas a 5% dos usuários que moram em um município que teve uma ocorrência registrada. Essa rodada acontece em um intervalo de 2 horas começando pelo momento onde teve a primeira ocorrência.	● Caso uma ocorrência não seja validada em no máximo 2 horas ou seja rejeitada pelos usuários na rodada de validação, ela é registrada como inválida. ● Notificações inválidas tem o processo de notificação interrompido.	Média
RNF02 **Redundância de ocorrência**	Relatos realizados em uma mesma localidade em um mesmo intervalo de tempo são agrupados automaticamente.	● Relatos realizados em um mesmo período em uma mesma localidade são agrupados, eliminando a necessidade de realizar uma rodada de validação caso o	Alta

		threshold de 5% já seja cumprido.	
RNF03 **Limites de interação**	O sistema limita as interações que o usuário pode realizar para evitar spam.	● O usuário é limitado por uma notificação em um espaço de 24 horas. ● O usuário precisa esperar um tempo mínimo de 4 horas para marcar uma notificação como resolvida.	Baixa
RNF04 **Suspensão de usuários**	Usuários que repetidamente marcam infrações no sistema de notificação são suspendidos temporariamente.	● Usuários que possuem mais ocorrências que foram registradas como inválidas (51%) do que válidas (50%) são suspendidos de criar novas notificações em um espaço de uma semana. ● Usuários que continuamente registram uma ocorrência como resolvida quando há uma prolongação do estado vermelho (RF06 Resolver Ocorrência) são impedidos de marcar ocorrências como resolvidas em um espaço de 1 semana.	Média
RNF05 **Criptografia**	Informações sensíveis do usuário, como senha e endereço devem ser criptografadas.	● As senhas devem ser criptografadas utilizando um algoritmo de *hash* único, através do Argon2Id.	Alta

		● Endereços devem ser criptografados utilizando AES.	
RNF06 **LGPD**	O sistema deve estar em conformidade em relação às normas descritas pela lei de garantia e proteção de dados (Lei nº 13.709/2018).	● Cumprir com cada norma relatada na lei nº 13.709/2018.	Alta

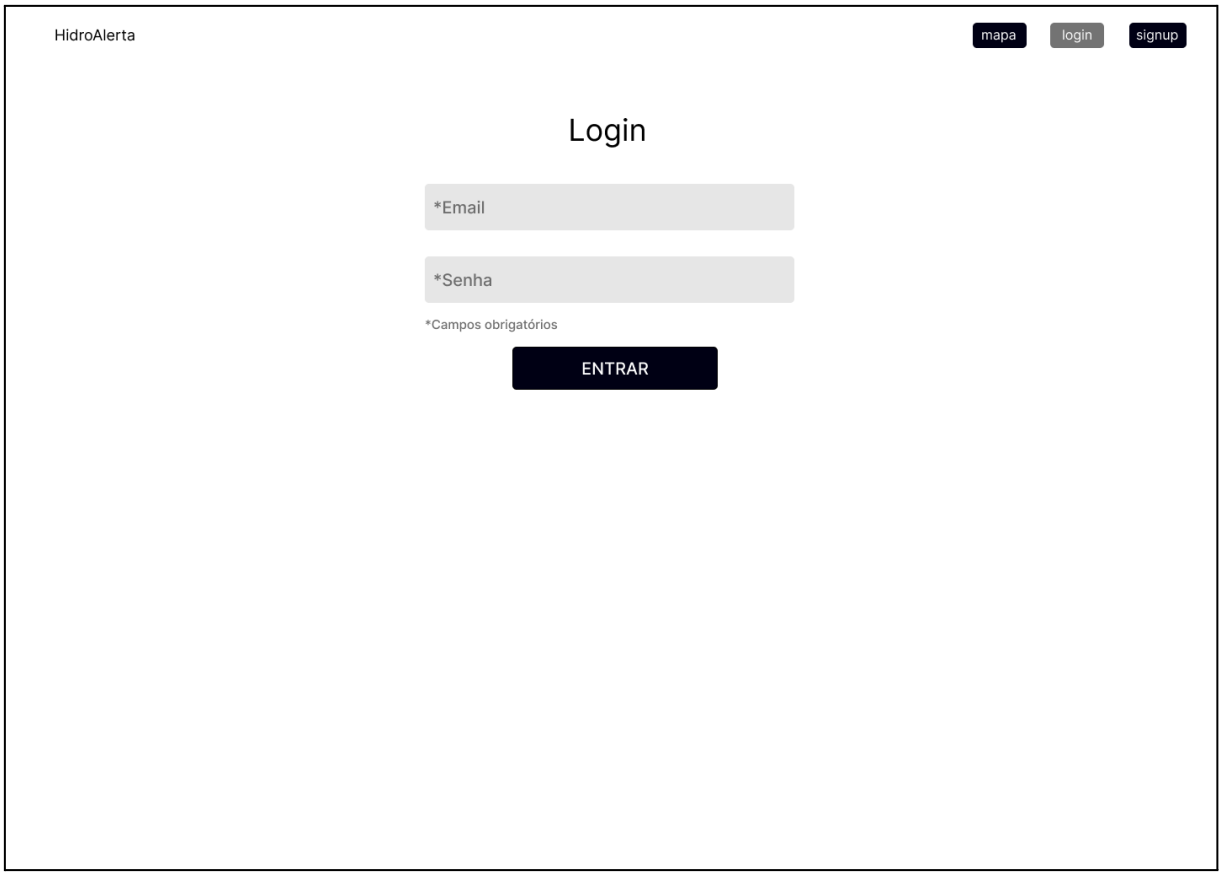
3.3 Protótipo

Figura 3 - Tela inicial

Fonte: Do próprio autor, 2025.

O objetivo da tela inicial é introduzir o usuário aos recursos da aplicação, podendo acessar as páginas de criar ou entrar à conta e visualizar o mapa de alertas.

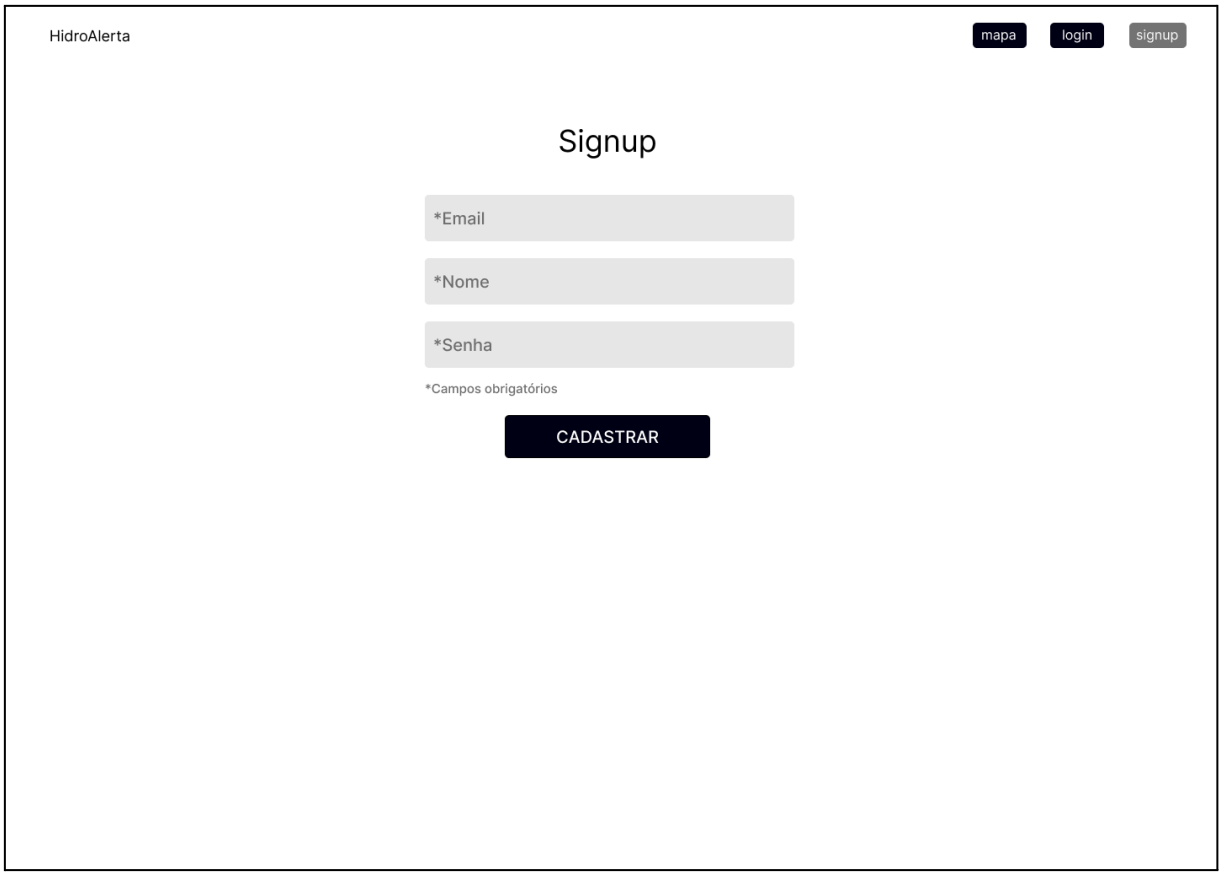
Figura 4 - Tela de Login



Fonte: Do próprio autor, 2025.

O objetivo da tela de Login é realizar o login no sistema, aqui o usuário irá preencher os campos obrigatórios, email e senha que ele utilizou durante o processo de cadastramento. Ao clicar em entrar o usuário será redirecionado para a tela de mapa de alertas.

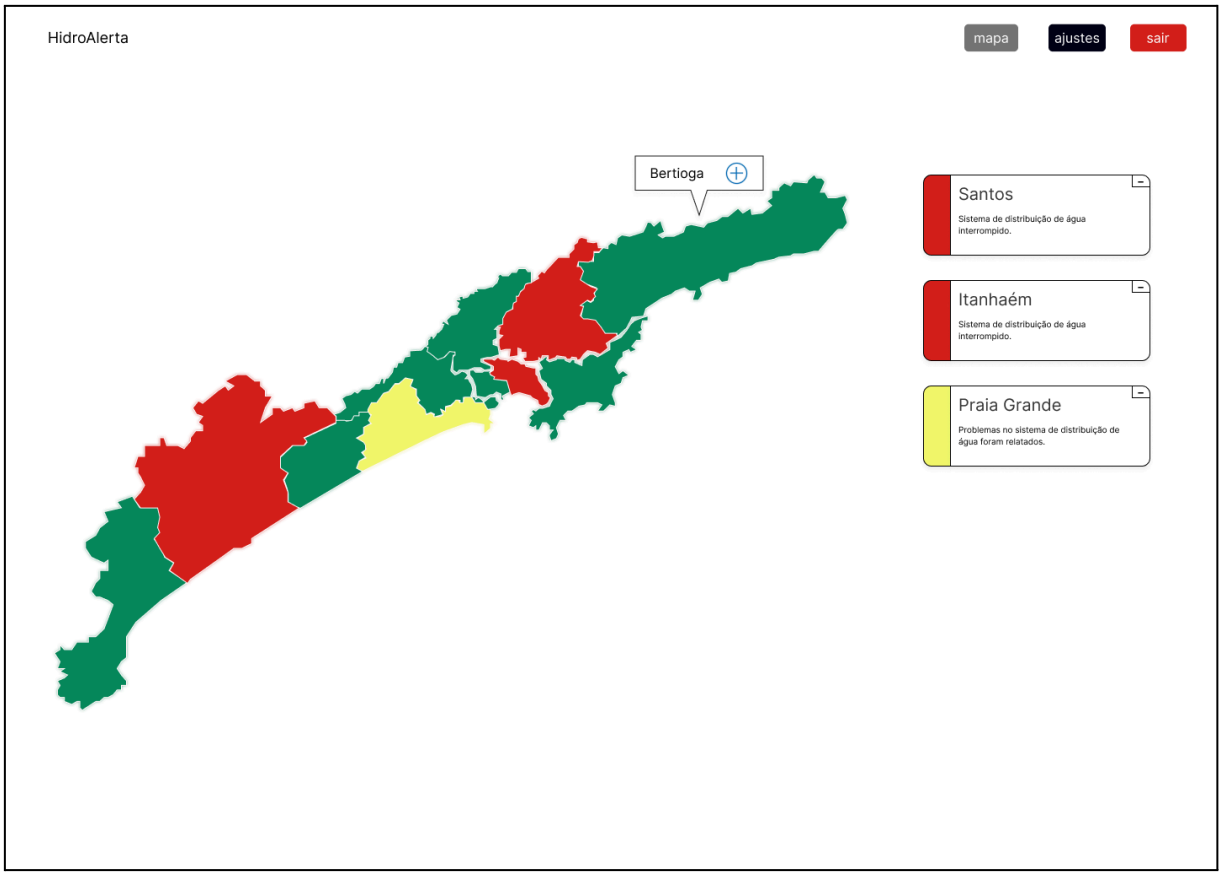
Figura 5 - Tela de SignUp



Fonte: Do próprio autor, 2025.

O objetivo da tela de SignUp é realizar o cadastro do usuário no sistema, aqui ele irá preencher os campos obrigatórios, email, nome e senha. Ao clicar em cadastrar o sistema irá validar os dados, enviando um email de confirmação ao usuário e redirecionando ele para a tela de mapa de alertas.

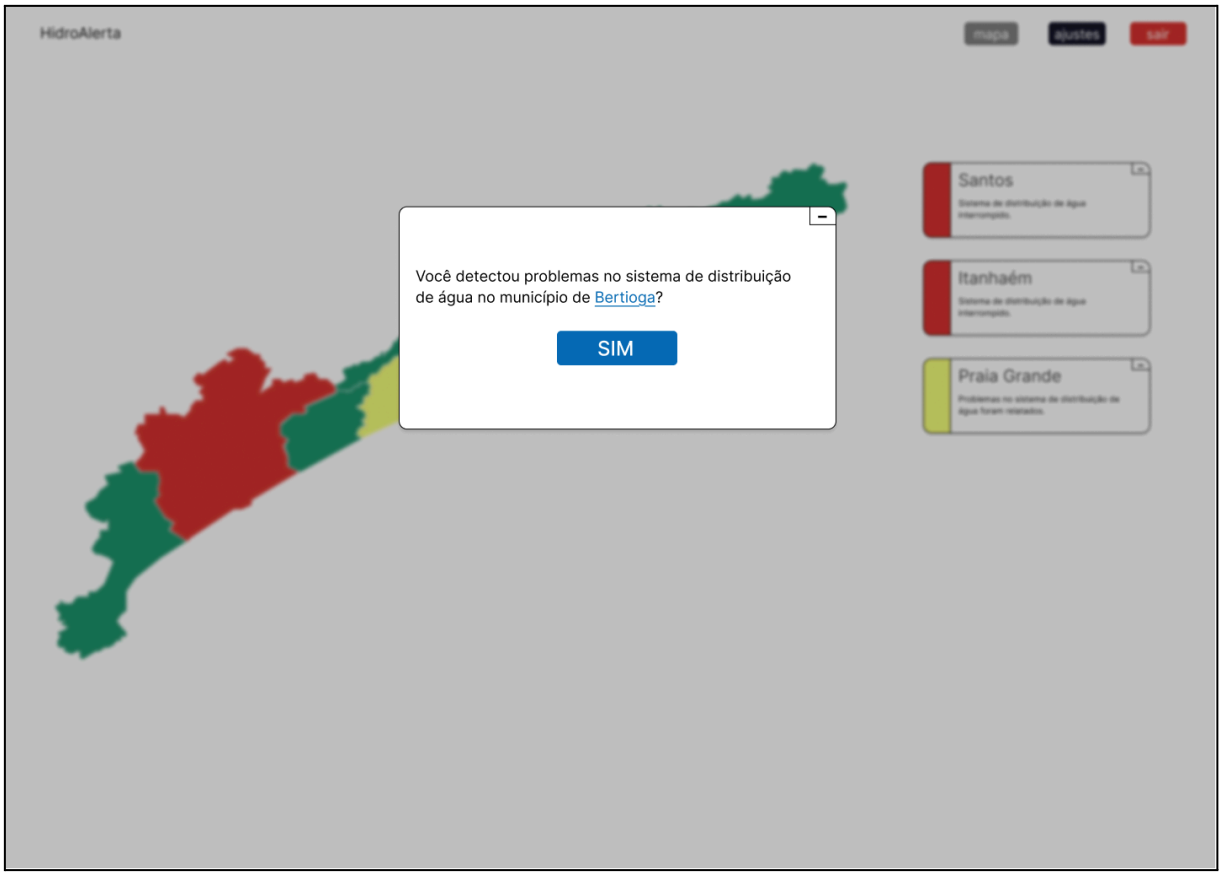
Figura 6 - Tela Mapa de Alerta



Fonte: Do próprio autor, 2025.

O objetivo do mapa de alerta é visualizar as condições atuais do sistema de abastecimento de água dos municípios, também sendo possível reportar problemas que o usuário esteja presenciando, além de acessar a página de ajustes de conta e fazer logout.

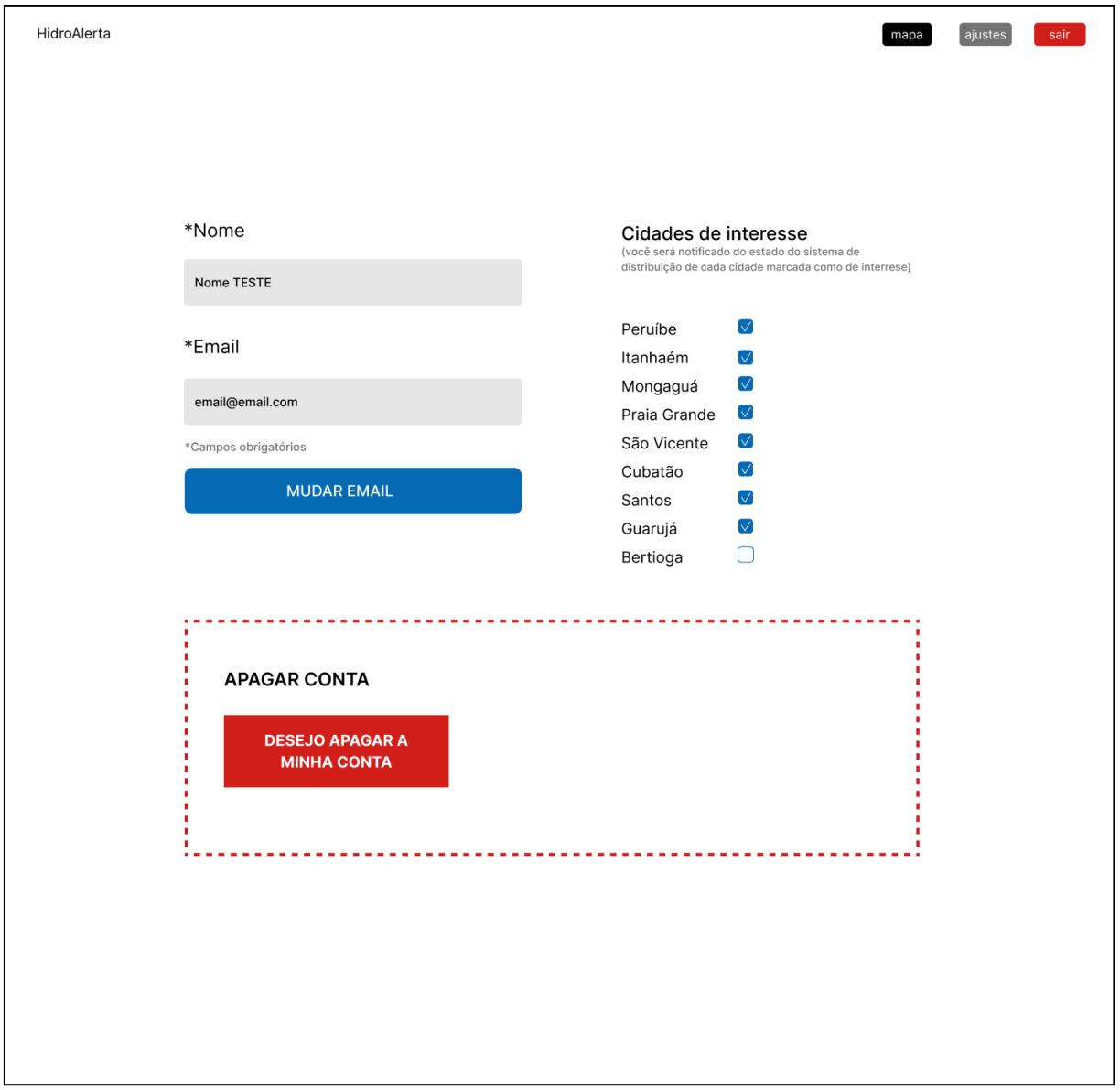
Figura 7 - Tela de relato



Fonte: Do próprio autor, 2025.

O objetivo da tela de relato é fornecer o método pelo qual o usuário irá confirmar o relato dele, ao clicar em ‘sim’ o sistema irá enviar uma requisição ao sistema que será processada de acordo com o que foi definido nos requisitos funcionais e não funcionais.

Figura 8 - Tela de ajustes



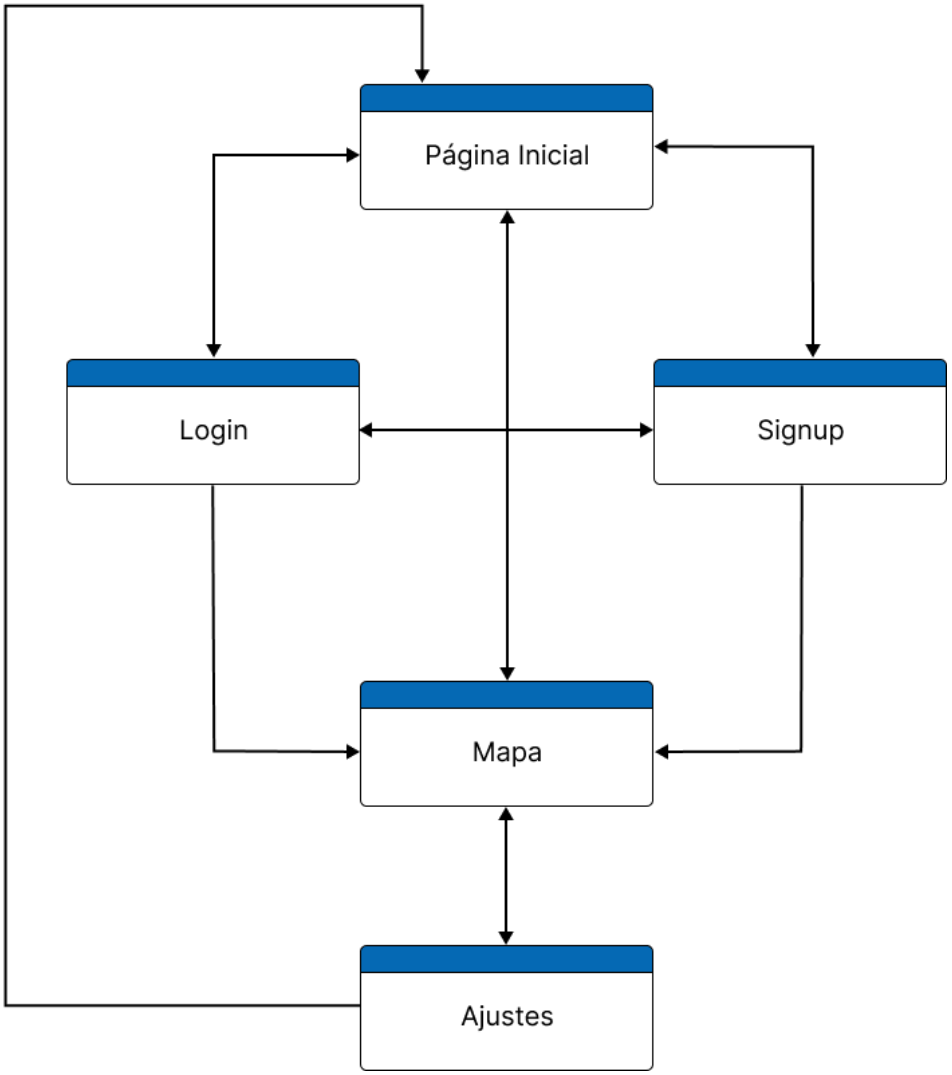
Fonte: Do próprio autor, 2025.

O objetivo da tela de ajustes é fornecer parâmetros para que o usuário modifique o seu cadastro, incluindo a exclusão dele e uma lista de checagens que permite o usuário especificar quais cidades ele tem interesse em receber uma notificação descrevendo o estado do sistema de distribuição de água de cada município. Ao clicar em uma caixa de checagem a página envia automaticamente uma requisição ao sistema indicando a alteração, o campo de nome segue a mesma lógica porém escutando pelo o evento de *input* no campo, utilizando uma técnica de *debouncing* para evitar a sobrecarga do servidor, já o campo de email o usuário

deve clicar no botão ‘mudar email’ após alterar o campo para que o sistema envie um link através do email para que ele confirme a alteração. O botão ‘DESEJO APAGAR A MINHA CONTA’ deve apresentar um *popup* pedindo uma confirmação do usuário se ele realmente deseja excluir a sua conta.

3.3.1 Diagrama de navegação

Figura 9 - Diagrama de navegação



Fonte: Do próprio autor, 2025.

4. Arquitetura do sistema

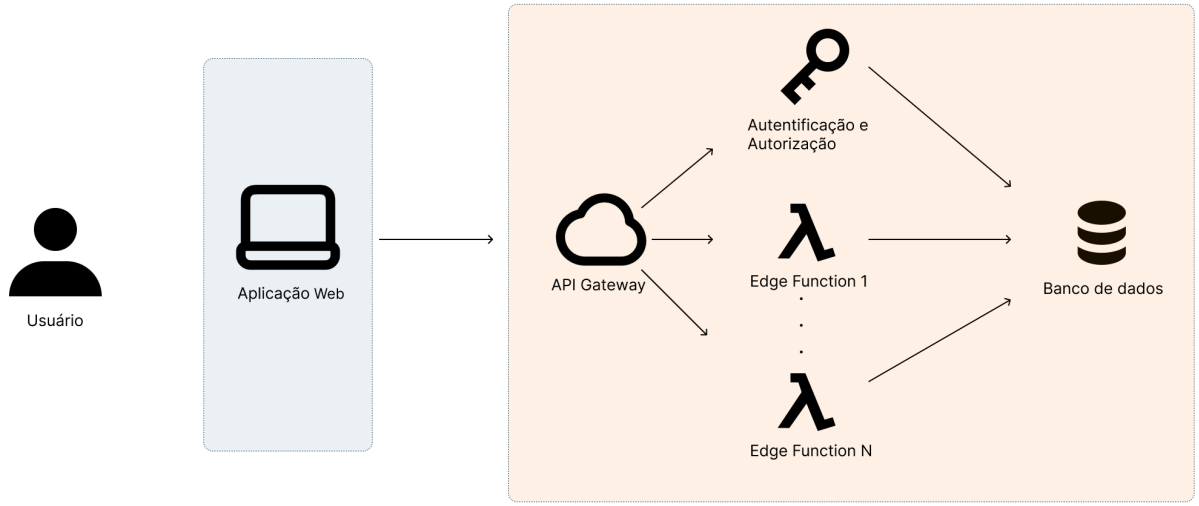
O HidroAlerta Baixada adota uma arquitetura de uma aplicação web reativa utilizando um BaSS (Backend *as a service*) e *edge functions* para o manuseamento e persistência de dados utilizando uma *pipeline* de CI/CD para garantir a integração e entrega contínua de software.

4.1 Componentes do sistema

- Front**-End**: React, Vite, Shadcn/ui.
- **API** Gateway**:** Supabase
- **Back-**End**:** Caddy, CloudFlare.
- **Banco** de **Dados:** Postgresql.

4.2 Diagrama de arquitetura

Figura 10 - Diagrama de arquitetura *serverless*



Fonte: Do próprio autor, 2025.

4.3 Segurança

A autorização e autenticação do sistema serão ambas gerenciadas pelo Supabase com tokens JWT para confirmar as autorizações do usuário. Utilizando a

feature de *Row Level Security Police* do Postgresql delimitamos o acesso e o poder de alteração dos dados.

4.4 Escalabilidade

Inicializar diversas instâncias do react sendo servidas pelo Vite e colocá-las atrás do servidor de proxy reverso Caddy, escalando o componente horizontalmente.

Por ser uma estrutura *serverless* o banco de dados escala automaticamente variando pelo uso da aplicação.

4.5 Manutenção

A *pipeline* de CI/CD permite que façamos alterações no código ao passo que os testes automatizados validam que essas alterações não quebraram *features* que já foram implementadas. Além de que o fato do backend estar desacoplado do front-end, permite que o cliente opere através de abstrações fornecidas pela api do Supabase, garantindo o funcionamento da aplicação mesmo depois de alterações no backend.

4.6 Integração

Integração com o Supabase que gere o envio de emails, uso de provedores de identidade, gerenciamento de sessões e logs, banco de dados, autenticação, *edge functions,* além de armazenamento de arquivos estáticos.

A integração com o CloudFlare permite configurar o DNS para utilizar o próprio CloudFlare como um proxy, escondendo o ip verdadeiro do nosso servidor que está atrás do domínio.

4.7 Limitações

Por utilizar um BaSS o sistema tende a ficar preso às soluções que são oferecidas pelo provedor, o que é amenizado pelo fato do Supabase ser um projeto open-source o que permite fazer o self-host de todo o stack oferecido no serviço de nuvem.

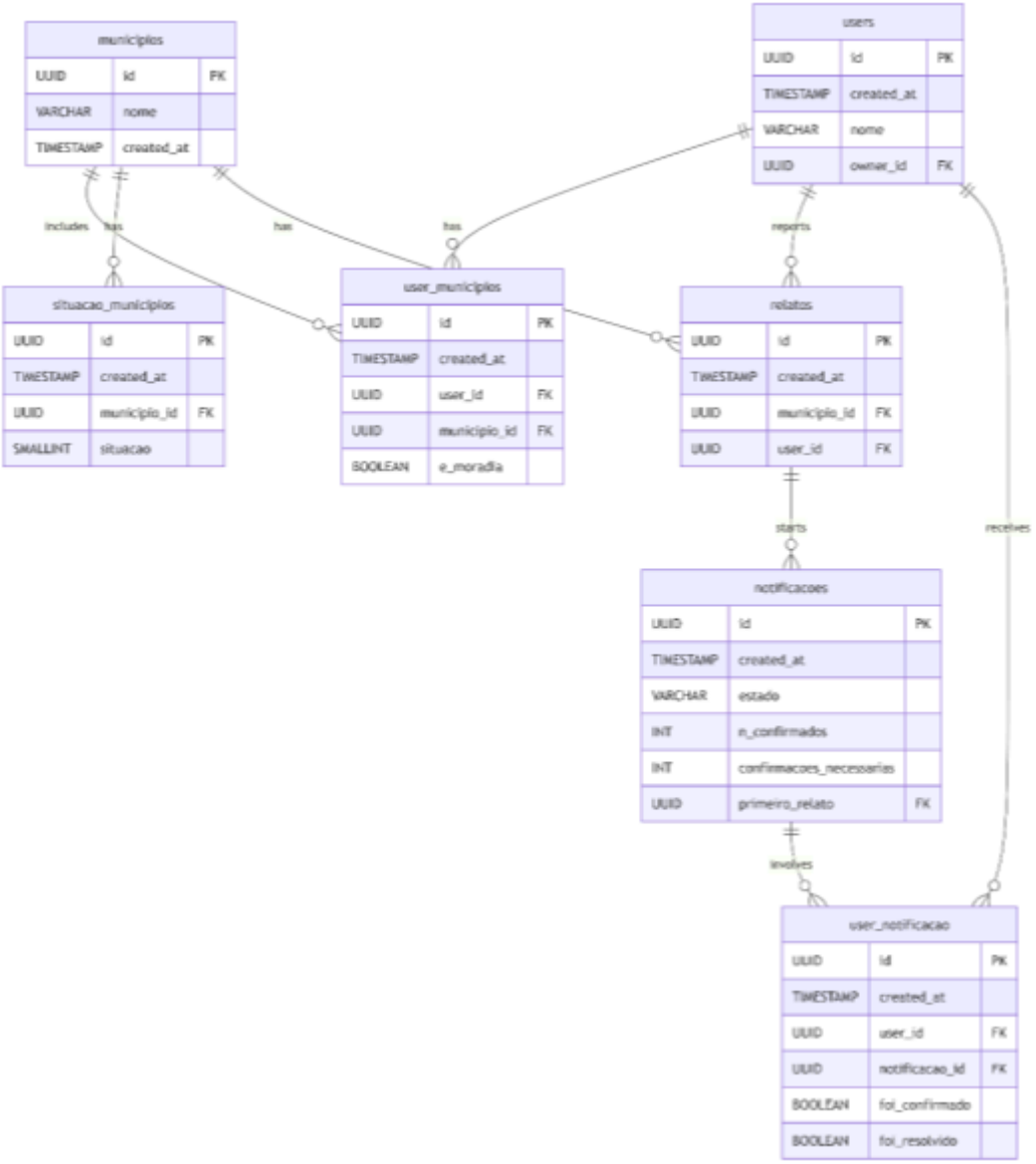
Os runtimes de JavaScript possuem um problema crônico em relação a aplicações de longa execução em detrimento das otimizações realizadas pelo JIT nas funções que estão sendo executadas, armazenando essas otimizações o que a longo prazo tende a consumir uma quantidade copiosa de memória, necessitando que a instância seja reinicializada.

5. Arquitetura de dados

O banco de dados utilizado é o PostgreSQL, um banco relacional que possui transações e a possibilidade de criar permissões de acordo com linhas em uma tabela, chamada de *Row Level Security*. O que segue é modelo lógico da base de dados, os *scripts* de criação das tabelas e permissões, além do dicionário de dados que descreve o tipo, *constraints* e o uso de cada coluna nas tabelas..

5.1 Modelo lógico da base de dados

Figura 11 - Diagrama do Modelo Lógico de Dados



Fonte: Do próprio autor, 2025.

5.2 Criação física do modelo de dados

```
create table public.users (
  id uuid not null default gen_random_uuid (),
  created_at timestamp with time zone not null default now(),
  nome character varying not null,
  owner_id uuid not null,
  constraint users_pkey primary key (id),
  constraint users_owner_id_fkey foreign KEY (owner_id) references auth.users
(id)
) TABLESPACE pg_default;

create table public.municipios (
  id uuid not null default gen_random_uuid (),
  nome character varying not null,
  created_at timestamp with time zone not null default now(),
  constraint municipios_pkey primary key (id)
) TABLESPACE pg_default;

create table public.situacao_municipios (
  id uuid not null default gen_random_uuid (),
  created_at timestamp with time zone not null default now(),
  municipio_id uuid not null,
  situacao smallint not null default '0'::smallint,
  constraint situacao_municipios_pkey primary key (id),
  constraint situacao_municipios_municipio_id_fkey foreign KEY (municipio_id)
references municipios (id)
) TABLESPACE pg_default;

create table public.user_municipios (
  id uuid not null default gen_random_uuid (),
  created_at timestamp with time zone not null default now(),
  user_id uuid not null,
  municipio_id uuid not null,
  e_moradia boolean not null,
  constraint user_municipios_pkey primary key (id),
  constraint user_municipios_municipio_id_fkey foreign KEY (municipio_id)
references municipios (id),
  constraint user_municipios_user_id_fkey foreign KEY (user_id) references
users (id)
) TABLESPACE pg_default;

create table public.relatos (
```

```
  id uuid not null default gen_random_uuid (),
  created_at timestamp with time zone not null default now(),
  municipio_id uuid not null,
  user_id uuid not null default gen_random_uuid (),
  constraint relatos_pkey primary key (id),
  constraint relatos_municipio_id_fkey foreign KEY (municipio_id) references
municipios (id),
  constraint relatos_user_id_fkey foreign KEY (user_id) references users (id)
) TABLESPACE pg_default;

create table public.notificacoes (
  id uuid not null default gen_random_uuid (),
  created_at timestamp with time zone not null default now(),
  estado character varying not null default 'em_confirmacao'::character
varying,
  n_confirmados integer not null default 0,
  confirmacoes_necessarias integer not null,
  primeiro_relato uuid not null,
  constraint notificacao_pkey primary key (id),
  constraint notificacao_primeiro_relato_fkey foreign KEY (primeiro_relato)
references relatos (id)
) TABLESPACE pg_default;

create table public.user_notificacao (
  id uuid not null default gen_random_uuid (),
  created_at timestamp with time zone not null default now(),
  user_id uuid not null,
  notificacao_id uuid not null,
  foi_confirmado boolean not null default false,
  foi_resolvido boolean null,
  constraint user_notificacao_pkey primary key (id),
  constraint user_notificacao_notificacao_id_fkey foreign KEY
(notificacao_id) references notificacoes (id),
  constraint user_notificacao_user_id_fkey foreign KEY (user_id) references
users (id)
) TABLESPACE pg_default;

create policy "Enable insert for service role only"
on "public"."municipios"
as PERMISSIVE
to service_role
with check (
  true
);
```

```
create policy "Enable read access for all users"
on "public"."municipios"
as PERMISSIVE
to public
using (
  true
);

create policy "Enable update for service role only"
on "public"."municipios"
as PERMISSIVE
to service_role
using (
  true
);

create policy "Enable insert for service role only"
on "public"."notificacoes"
as PERMISSIVE
to service_role
with check (
  true
);

create policy "Enable read access for service role users"
on "public"."notificacoes"
as PERMISSIVE
to service_role
using (
  true
);


create policy "Enable update for service role only"
on "public"."notificacoes"
as PERMISSIVE
to service_role
using (
  true
);

create policy "Enable insert for service role only"
on "public"."relatos"
as PERMISSIVE
```

```
to service_role
with check (
  true
);

create policy "Enable read access for service role"
on "public"."relatos"
as PERMISSIVE
to service_role
using (
  true
);

create policy "Enable users to view their own data only"
on "public"."relatos"
as PERMISSIVE
to authenticated
using (
  (( SELECT auth.uid() AS uid) = user_id)
);

create policy "Enable insert for service role only"
on "public"."situacao_municipios"
as PERMISSIVE
to service_role
with check (
  true
);

create policy "Enable read access for all users"
on "public"."situacao_municipios"
as PERMISSIVE
to public
using (
  true
);

create policy "Enable update for service role only"
on "public"."situacao_municipios"
as PERMISSIVE
to service_role
using (
  true
);
```

```
create policy "Enable delete for users based on user_id"
on "public"."user_municipios"
as PERMISSIVE
to authenticated
using (
  (( SELECT auth.uid() AS uid) = user_id)
);

create policy "Enable insert for authenticated users only"
on "public"."user_municipios"
as PERMISSIVE
to authenticated
with check (
  (( SELECT auth.uid() AS uid) = user_id)
);

create policy "Enable update for users based on id"
on "public"."user_municipios"
as PERMISSIVE
to authenticated
using (
  (( SELECT auth.uid() AS uid) = user_id)
with check (
  (( SELECT auth.uid() AS uid) = user_id)
);

create policy "Enable users to view their own data only"
on "public"."user_municipios"
as PERMISSIVE
to authenticated
using (
  (( SELECT auth.uid() AS uid) = user_id)
);

create policy "Enable insert for service role only"
on "public"."user_notificacao"
as PERMISSIVE
to service_role
with check (
  true
);

create policy "Enable update for service role only"
on "public"."user_notificacao"
as PERMISSIVE
```

```
to authenticated
using (
  true
);

create policy "Enable users to view their own data only"
on "public"."user_notificacao"
as PERMISSIVE
to authenticated
using (
  (( SELECT auth.uid() AS uid) = user_id)
);

create policy "Enable users to view their own data only"
on "public"."users"
as PERMISSIVE
for SELECT
to authenticated
using (
(( SELECT auth.uid() AS uid) = id)
);
```

5.3 Dicionário de dados

Tabela de usuário

Nome da coluna	Tipo de dados	Restrições	Descrição
id	uuid	NOT NULL, PRIMARY KEY	Chave primária do usuário
nome	varchar	NOT NULL	Nome do usuário
owner_id	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira que referencia o id da conta criada pelo usuário no serviço de autenticação
created_at	timestampz	NOT NULL	Data e horário que o usuário criou a conta

Tabela de municípios

Nome da coluna	Tipo de dados	Restrições	Descrição
id	uuid	NOT NULL, PRIMARY KEY	Chave primária do município
nome	varchar	NOT NULL	Nome do município
created_at	timestampz	NOT NULL	Data e horário que município foi inserido no sistema

Tabela: situacao_municipios

Nome da coluna	Tipo de dados	Restrições	Descrição
id	uuid	NOT NULL, PRIMARY KEY	Chave primária da situação do município
created_at	timestampz	NOT NULL	Data e horário do registro
municipio_id	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira que referencia o município
situacao	smallint	NOT NULL, DEFAULT 0	Situação atual do município representada como número inteiro pequeno

Tabela: situacao_municipios

Nome da coluna	Tipo de dados	Restrições	Descrição
id	uuid	NOT NULL, PRIMARY KEY	Chave primária da associação usuário-município
created_at	timestampz	NOT NULL	Data e horário do registro
user_id	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira que referencia o usuário
municipio_id	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira que referencia o município

e_moradia	boolean	NOT NULL	Indica se o município é local de moradia do usuário

Tabela: user_municipios

Nome da coluna	Tipo de dados	Restrições	Descrição
id	uuid	NOT NULL, PRIMARY KEY	Chave primária da associação usuário-município
created_at	timestampz	NOT NULL	Data e horário do registro
user_id	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira que referencia o usuário
municipio_id	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira que referencia o município
e_moradia	boolean	NOT NULL	Indica se o município é local de moradia do usuário

Tabela: relatos

Nome da coluna	Tipo de dados	Restrições	Descrição
id	uuid	NOT NULL, PRIMARY KEY	Chave primária do relato
created_at	timestampz	NOT NULL	Data e horário do relato
municipio_id	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira que referencia o município do relato
user_id	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira que referencia o usuário que fez o relato

Tabela: notificacoes

Nome da coluna	Tipo de dados	Restrições	Descrição

id	uuid	NOT NULL, PRIMARY KEY	Chave primária da notificação
primeiro_relato	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira para o primeiro relato que gerou a notificação
estado	varchar	NOT NULL	Estado da notificação, que pode estar em: ‘em_confirmacao’ que é quando o primeiro relato é criado, porém o processo de confirmação ainda não realizado; o ‘pendente_confirmacao’ é quando o processo de confirmação foi inicializado e a notificação para uma amostragem de indivíduos que moram no município fora realizada para confirmar o relato; quando o processo de confirmação foi finalizado e confirmado o estado se torna ‘pendente’ indicando que precisamos notificar todos os usuários que moram no município de interesse; por fim temos o estado ‘notificado’ quando todos os usuários que moram no município de interesse foram notificados.
confirmacoes_necessarias	int4	NOT NULL	Número de confirmações necessárias para que a notificação mude o estado para geral
n_confirmados	int4	NOT NULL	Número de confirmações de que o sistema de distribuição de água foi interrompido

created_at	timestampz	NOT NULL	Data de quando a notificação foi criada

Tabela: user_notificacao

Nome da coluna	**Tipo de dados**	**Restrições**	**Descrição**
id	uuid	NOT NULL, PRIMARY KEY	Chave primária do user_notificacao
user_id	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira que aponta para o usuário que recebeu a notificação
notificacao_id	uuid	NOT NULL, FOREIGN KEY	Chave estrangeira que aponta para a notificação
foi_confirmado	bool	NOT NULL	Valor booleano que indica se o problema relacionado a notificação foi confirmado pelo usuário
foi_resolvido	bool	NOT NULL	Valor booleano que indica se o problema relacionada a notificação já foi reselvido
created_at	timestampz	NOT NULL	Data de quando o usuário foi notificado.

6 Projeto de Software

Este capítulo apresenta o projeto interno de cada componente do sistema, detalhando suas responsabilidades, dependências, e interação com edge functions, base de dados, eventos em tempo real, autorização/autenticação, além de que forma a lógica está separada no frontend e no backend. A partir da arquitetura já definida (*serveless*), descrevemos como cada parte interage para implementar as regras de negócio, garantindo a segurança e consistência de dados.

6.1 Design dos Componentes

O sistema possui uma arquitetura *serverless* orientada a eventos, baseada em componentes desacoplados que interagem via interfaces REST, WebSocket e SQL, utilizando o supabase como backend-as-a-service, que fornece os módulos utilizados no sistema (Edge Function Runtime, Realtime, Auth, Postgres, REST API e Crojob). O controle de acesso é feito de forma declarativa, com Row-Level Security (RLS) no banco de dados e validação de JWTs emitidos pelo módulo de autenticação.

A aplicação web em React serve como orquestrador das interações do usuário com o backend. No frontend o usuário realiza o login/signup, é inscrito a canais de evento em tempo real para receber atualizações no mapa relatando o estado dos municípios, além de receber notificações, interagindo com edge functions para realizar ações mais complexas, além da API REST onde ele consegue visualizar e modificar o estado dos dados.

O Edge Function Runtime é o módulo que executa lógicas complexas, é distribuído na nuvem utilizando Cloudflare workers, no qual o frontend consegue realizar chamadas as essas funções de forma opaca. O *runtime* tem acesso à API REST para manipular qualquer dado que seja necessário.

O Supabase Auth é o componente de autenticação fornecido pelo supabase com ele podemos, realizar login, criar uma nova conta ou verificar um token JWT.

A API REST realiza uma expoxição restful das tabelas do banco de dados através do PostgresREST, dependendo do módulo de autenticação para verificar os *claims* de um token JWT, estando sujeita às políticas RLS baseadas no token.

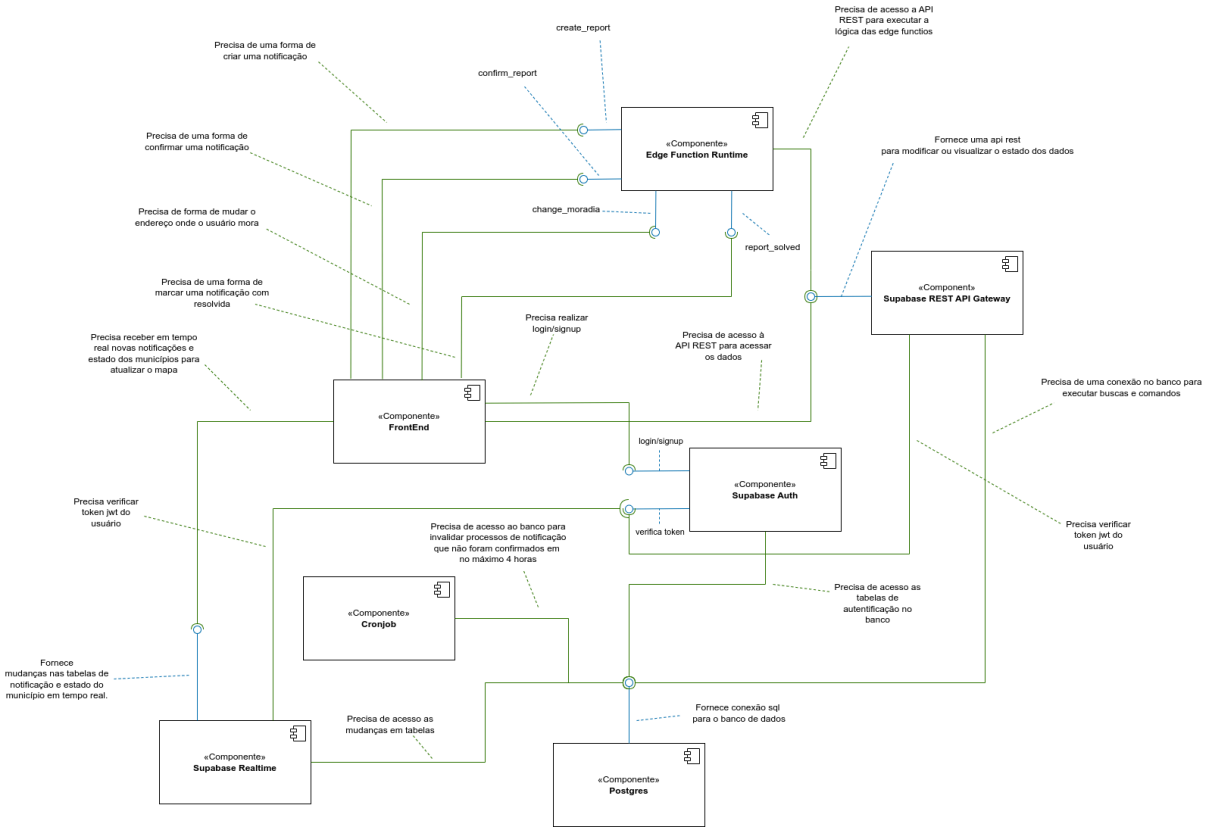
O Supabase Realtime é o módulo que sincroniza e espede eventos para os clientes conectados ao serviço, atualizando a situação de cada município além de fornecer notificações aos usuários. Depende do componente de autenticação para verificar se o usuário tem acesso a uma notificação específica, além de ser dependente das mudanças que ocorrem nas tabelas na base de dados.

A base de dados em Postgres é o núcleo de persistência do sistema, todas as operações de leitura e escrita convergem para esse módulo, protegido por RLS e fornecendo as mudanças no banco em tempo real através de repartições lógicas.

O Cronjob realiza comandos de forma periódica na base de dados para invalidar notificações que estão em um processo de confirmação por mais de 4 horas, essa ação é realizada de hora em hora.

Abaixo segue um diagrama de componente explicando as interações entre os diferentes módulos no projeto além do fluxo de dependências.

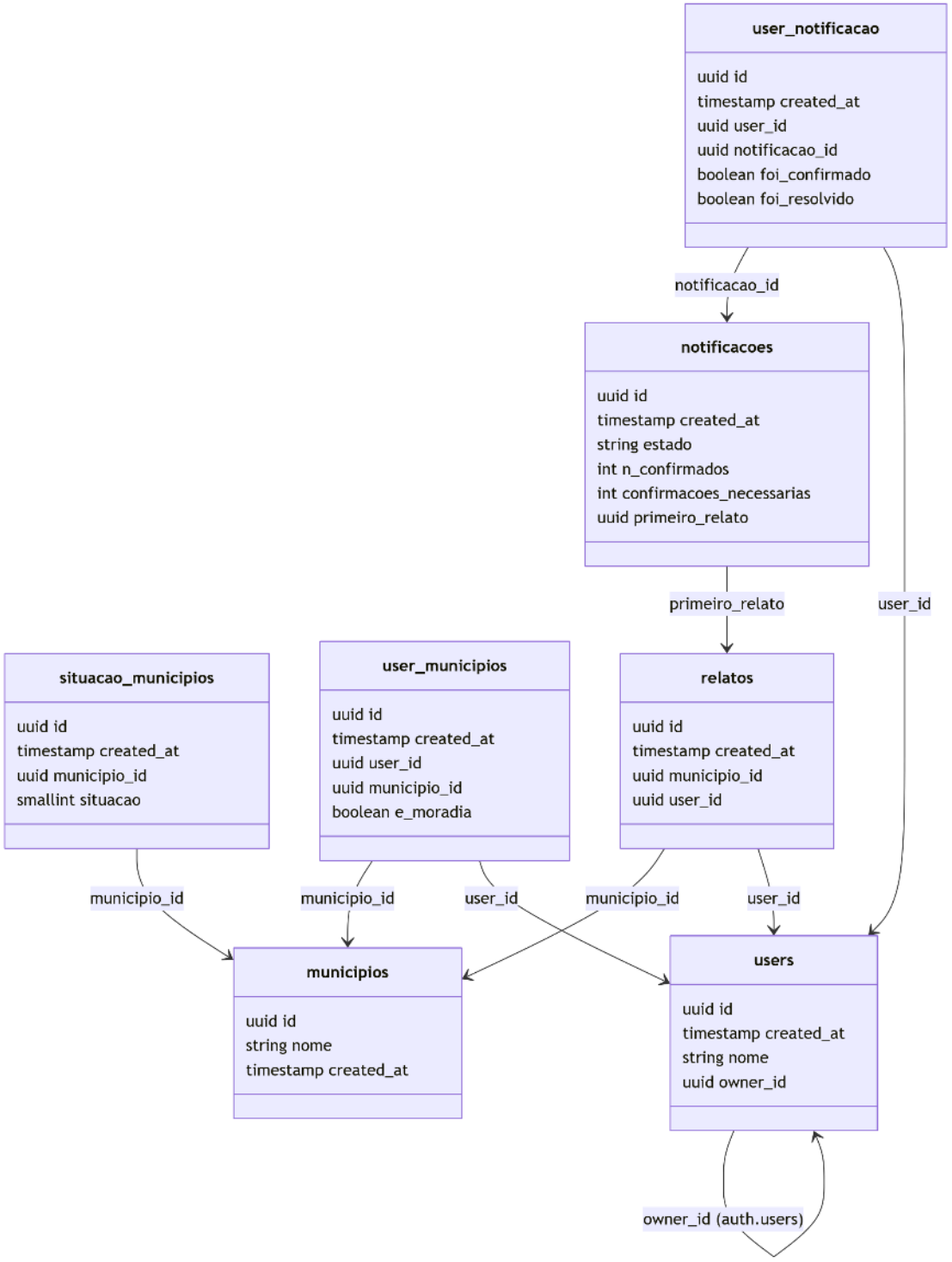
Figura 12 - Diagrama de Componente



Fonte: Do próprio autor, 2025.

6.2 Diagrama de Classe

Figura 13 - Diagrama de Classe

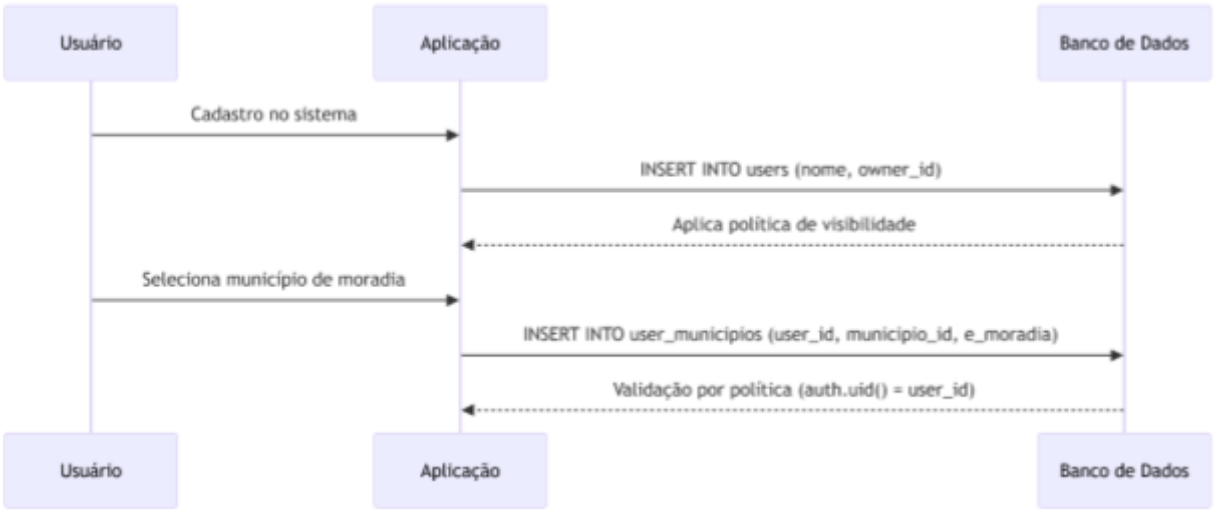


Fonte: Do próprio autor, 2025.

6.3 Diagrama de Sequência

6.3.1 Criação de Usuário e Associação a Município

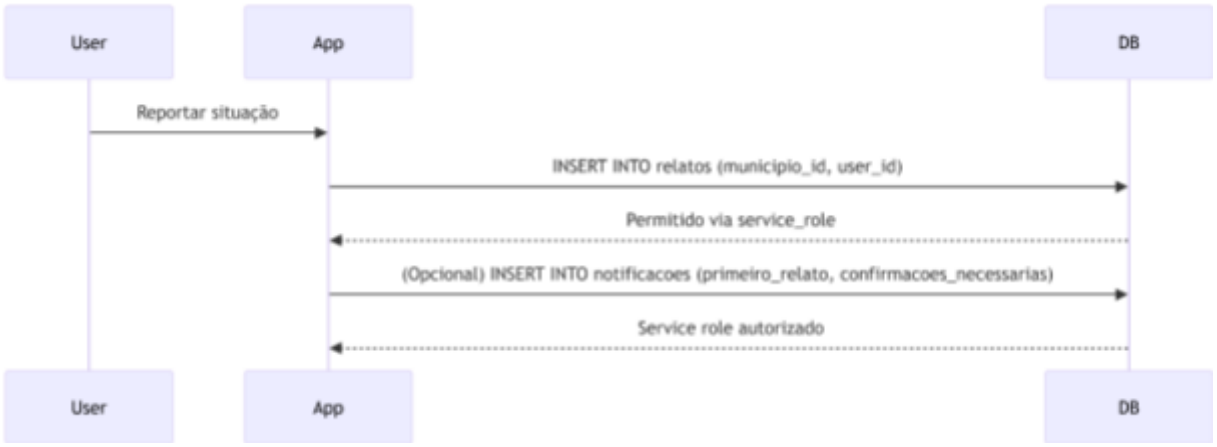
Figura 14 - Diagrama de Sequência, Criação de Usuário e Associação a Município



Fonte: Do próprio autor, 2025.

6.3.2 Usuário Cria um Relato

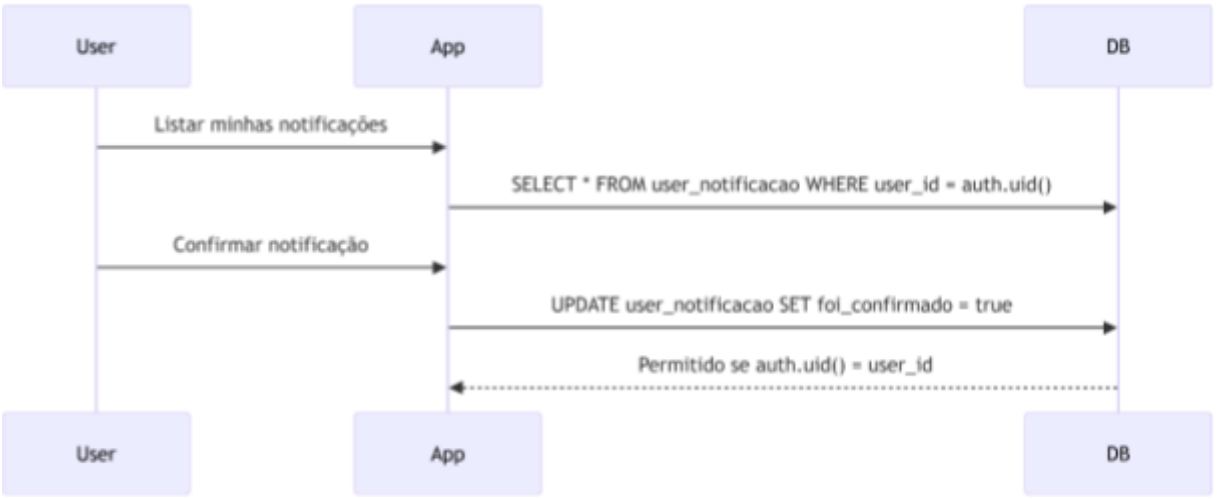
Figura 15 - Diagrama de Sequência, Cria um relato



Fonte: Do próprio autor, 2025.

6.3.3 Usuário Visualiza e Interage com Notificações

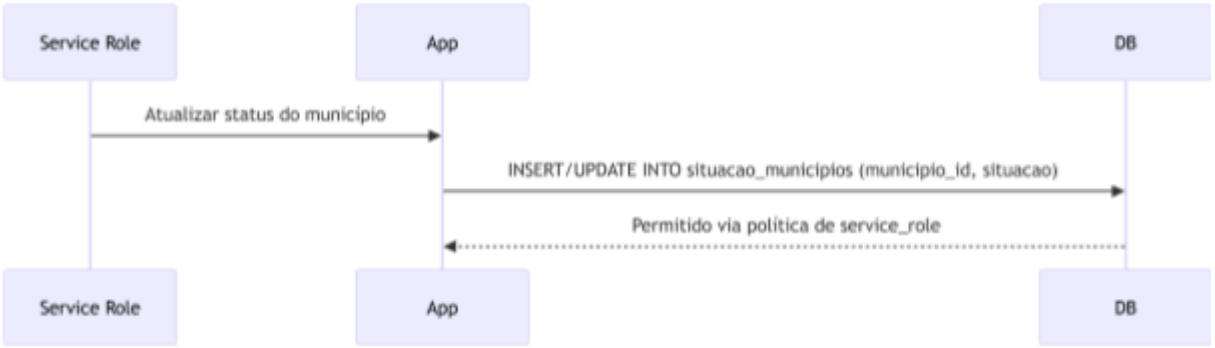
Figura 16 - Diagrama de Sequência, Usuário Visualiza e Interage com Notificações



Fonte: Do próprio autor, 2025.

6.3.4 Service Role Atualiza Situação de Município

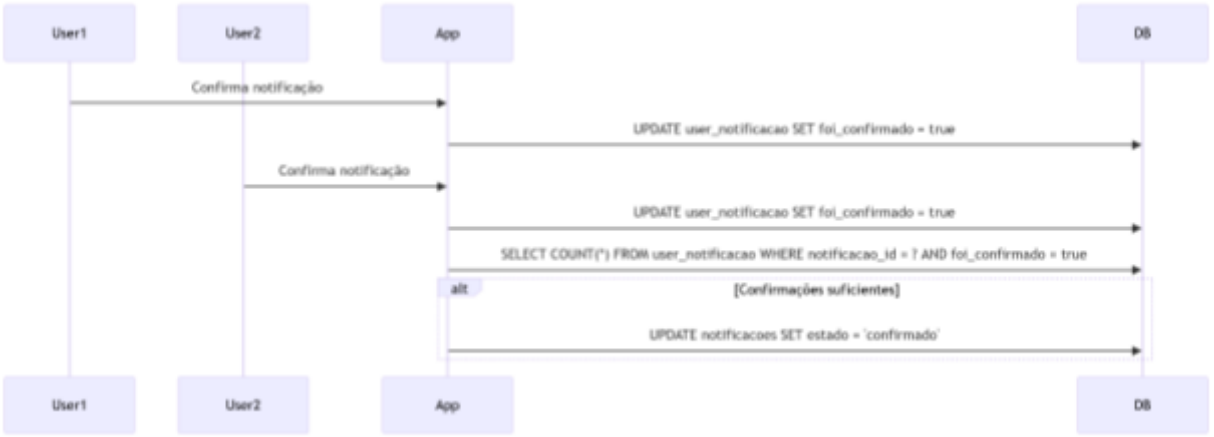
Figura 17 - Diagrama de Sequência, Service Role Atualiza Situação de Município



Fonte: Do próprio autor, 2025.

6.3.5 Confirmação de Notificação Coletiva

Figura 18 - Diagrama de Sequência, Confirmação de Notificação Coletiva



Fonte: Do próprio autor, 2025.

6.4 Regras de Negócio Detalhadas

As regras de negócio do sistema estão distribuídas de forma estratégica entre o frontend, às edge functions e o banco de dados, com forte ênfase na validação e consistência no lado do servidor. O frontend, desenvolvido em React, realiza apenas validações básicas e atua como interface com o usuário. Toda lógica crítica é centralizada em edge functions executadas no runtime serverless do supabase, garantindo atomicidade e controle. O banco de dados, por sua vez, impõe regras adicionais através de Row-Level Security (RLS), constraints e triggers.

Em relação ao fluxo de notificações, cada notificação criada precisa ser confirmada em até 4 horas. Caso contrário, ela é invalidada automaticamente por um cronjob que executa essa verificação periodicamente. Apenas usuários que receberam a notificação pode confirmá-la, e uma vez marcada como resolvida, a notificação não pode mais ser alterada. Essas garantias são aplicadas por meio de edge functions e reforçadas por políticas RLS.

No que diz respeito ao endereço de moradia do usuário, o sistema garante que apenas um endereço esteja ativo por vez. Ao solicitar uma mudança, o endereço anterior é invalidado automaticamente. Essa transição é tratada de forma atômica por uma edge function específica (change_moradia), que aplica a modificação e realiza as validações necessárias em uma única transação.

Todas as ações do sistema são autenticadas via tokens JWT emitidos pelo módulo de autenticação do supabase. As permissões do usuário são inferidas a partir das *claims* presentes nesses tokens, e aplicadas diretamente nas regras RLS configuradas no banco de dados. Isso garante que cada consulta ou modificação em dados sensíveis ocorra apenas dentro do escopo de acesso do usuário autenticado. Com o módulo de realtime do supabase só emitindo eventos de alteração para usuários que têm, segundo as regras RLS, acesso legítimo à linha modificada.

Em termos de consistência, às edge functions executam transações SQL para garantir que atualizações envolvendo múltiplas tabelas sejam aplicadas de forma coesa. O banco de dados ainda utiliza triggers para propagar automaticamente alterações de notificações para o estado do município correspondente. *Constraints* e chaves estrangeiras asseguram a integridade referencial dos dados, como, por exemplo, a relação entre notificações e seus respectivos usuários.

6.5 Estratégias de Tratamento de Erros

O sistema adota uma abordagem unificada e previsível para o tratamento de erros, com separação clara entre falhas técnicas e violações de regras de negócio. Os erros são classificados em três níveis: erros de cliente (ex: entrada inválida), erros de negócio (ex: tentativa de confirmar uma notificação expirada), e erros internos (ex: falha em conectar com o Postgres).

No frontend em React, erros são capturados no nível de chamada (try/catch) e exibidos ao usuário de forma contextualizada, evitando vazamento de mensagens técnicas. Para erros previsíveis, como validação de formulário ou restrições de acesso, o sistema exibe mensagens amigáveis e localizadas, mantendo a fluidez da experiência. Erros inesperados disparam mensagens genéricas (“Erro interno no servidor, tente novamente mais tarde”), e podem ser logados para análise posterior.

As edge functions são responsáveis por capturar falhas em lógica de negócio ou inconsistências do banco. Elas retornam respostas HTTP padronizadas com códigos de *status* adequados (400, 403, 500, etc.) e mensagens em formato JSON contendo um campo "message", que o frontend pode interpretar.

Na base de dados, falhas em transações SQL são capturadas e tratadas com *rollback* automático. Erros críticos ou de integridade (ex: violação de *constraint*) são registrados via logs estruturados para observabilidade.

No caso do Supabase Realtime, erros de conexão ou de autorização (ex: token expirado) resultam em tentativas automáticas de reconexão.

7 Implementação

Este capítulo visa documentar como as funcionalidades do sistema foram implementadas utilizando o React e supabase, detalhando a estrutura dos modelos, as interações entre componentes e a aplicação do modelo *serveless*, com foco na modularidade, reutilização de código e manutenção.

7.1 Estrutura geral do projeto

A estrutura do projeto segue o padrão de projeto de aplicações reativa integradas ao supabase utilizando o template de construção do Vite, com todos os arquivos relacionados a arquitetura serveless no diretório ‘supabase’. O projeto está organizado da seguinte forma:

```
hidro-alerta/
     ├── README.md
     ├── eslint.config.js
     ├── index.html
     ├── package.json
     ├── postcss.config.js
     ├── tailwind.config.js
     ├── vite.config.js
     ├── vite.config.ts
     ├── public/
     │    ├── manifest.json
     │    └── robots.txt
     ├── src/
     │    ├── Ajustes.jsx
     │    ├── App.css
     │    ├── App.jsx
     │    ├── App.test.js
     │    ├── index.css
     │    ├── Login.jsx
     │    ├── Main.jsx
     │    ├── Mapapage.jsx
     │    ├── reportWebVitals.js
     │    ├── setupTests.js
     │    ├── Signup.jsx
     │    ├── vite-env.d.ts
     │    ├── assets/
     │    └── components/
```

```
      │     ├── Alertcard.jsx
      │     ├── Cities.jsx
      │     ├── Header.jsx
      │     ├── Header2.jsx
      │     ├── Hero.jsx
      │     ├── Mapa.jsx
      │     └── Popup.jsx
      └── supabase/
      ├── config.toml
      ├── functions/
      │    ├── _shared/
      │    │    └── utils.ts
      │    ├── confirm_report/
      │    │    ├── deno.json
      │    │    ├── index.ts
      │    │    └── .npmrc
      │    ├── create_report/
      │    │    ├── deno.json
      │    │    ├── index.ts
      │    │    └── .npmrc
      │    ├── process_report/
      │    │    ├── deno.json
      │    │    └── index.ts
      │    └── report_solved/
      │      ├── deno.json
      │      ├── index.ts
      │      └── .npmrc
      └── migrations/
```

7.2 Paginas

A aplicação front-end do hidro-alerta é composta por 6 páginas jsx, que são elas:

7.2.1 App

A página App é a raiz da aplicação, aqui utilizamos um componente externo provido pela biblioteca React, Router, que possibilita especificar as rotas que estão presentes na nossa aplicação, além de Layouts que estarão presentes em todas as páginas.

7.2.2 MapaPage

Em MapaPage definimos o cerne da aplicação, realizando uma conexão com o supabase permitindo que os componentes nessa página realizem requisições para o banco de dados, além de renderizar o mapa e trazer os alerta em tempo real para o usuário, responsabilizadas para o componente “MapaBaixadaSantista”. Aqui também é registrado um evento relacionado a autentificação do usuário, garantindo que caso ele esteja autenticado receba as suas devidas notificações considerando o município que ele reside.

7.2.3 Ajustes

Na página ajustes, usuários que estão autenticados poderão alterar informações sobre sua conta como nome e residência, além de ter a possibilidade de excluir-lá.

7.2.4 Login e Signup

Nas respectivas telas de Login e Signup o usuário poderá autenticar em uma conta existente ou criar uma nova conta utilizando um nome, email válido e senha.

7.2.5 Signup Confirmation

Página onde usuário é direcionado no primeiro login após a criação de sua conta, aqui ele especifica se é um morador da baixada santista e se sim qual o município que ele reside.

7.2.6 PostSignup

É aqui que o usuário é imediatamente redirecionado ao criar uma nova conta, contando para o mesmo que um email de confirmação foi enviado.

7.2 Componentes

Dentre os componentes que não fazem parte do layout foi decidido a criação de 5 componentes principais, MapaBaixadaSantista que representa o estado do sistema de distribuição de água em cada município; OldNotifBar, que são as notificações já enviadas ao usuário ao qual ele pode visualizar uma marcar uma ocorrência como resolvida; AlertCard, que mostra as situações dos municípios em forma de texto ao longo que elas são atualizadas em tempo real; CalendarHeatMap, que permite o usuário acessar um calendário onde ele pode ver o histórico de cada municipio além de uma tabela de relatos por bairro naquela cidade; Popup, componente que serve para mostrar alguma notificação para o usuário ou pedir que ele realize alguma ação na interface.

REFERÊNCIAS

AGILE MANIFESTO. Manifesto for Agile Software Development. Disponível em: https://agilemanifesto.org/. Acesso em: 21 nov. 2024.