FUNDAÇÃO HERMÍNIO OMETTO

Sistemas de Informação

Projeto Interdisciplinar II / 4º A/2025

Tecnologias Inovadoras para promover Energia Limpa, Sustentável e Acessível

ENERGIA PARA TODOS

01

Arthur Peixoto Lacerda	116319
Enzo Zaia Soares	116657
Guilherme Henrique Cavarsan	117017
Octavio Thim Dias	117607
Rodolfo Henrique Ribeiro Zanchetta	117179

Fase Final do Projeto Técnico

1 Proposta do Projeto

1 - Contexto e Apresentação da Proposta

O projeto consiste na criação de uma ONG que faz a intermediação da doação de créditos de energia solar: pessoas ou empresas com excedente de produção conectam-se a famílias e comunidades carentes, permitindo que recebam energia elétrica acessível e sustentável. Essa iniciativa busca suprir uma lacuna social: a falta de acesso a energia em residências vulneráveis, sem demandar infraestrutura nova, mas aproveitando créditos existentes.

2 - Justificativa e Relação com o ODS 7

2.1 - Justificativa social e ambiental

Embora o Brasil já tenha elevada participação de fontes renováveis em sua matriz energética (cerca de 48 % em 2022), ainda existem comunidades que permanecem vulneráveis e sem acesso eficiente ao recurso energético. O projeto busca diminuir essa desigualdade, transferindo recursos já disponíveis (créditos solares) para quem mais precisa.

Além da relevância ambiental, a energia elétrica impacta diretamente indicadores sociais como educação, saúde e geração de renda, componentes do IDH. O projeto visa, portanto, promover inclusão energética como vetor de desenvolvimento humano.

2.2 - Conexão com o ODS 7:

O ODS 7, da ONU, estabelece metas como:

- Acesso universal, confiável, moderno e a preços acessíveis à energia
- Aumento da participação de energias renováveis na matriz energética
- Expansão da infraestrutura e modernização tecnológica

Nossa proposta se alinha diretamente a essas metas ao:

- Proporcionar acesso à energia elétrica sustentável a populações vulneráveis
- Estimular uso efetivo de energia renovável existente
- Criar uma rede tecnológica de doação e distribuição

3 - Objetivo do Projeto e Entregáveis

3.1 - Objetivo geral

Criar uma plataforma web que conecta doadores de créditos de energia solar com famílias e comunidades em situação de vulnerabilidade, promovendo acesso à energia limpa como bem social.

3.2 - Entregáveis principais

- Plataforma Web responsiva
- Cadastro e autenticação de doadores, beneficiários e administradores
- Módulo de gestão e distribuição automática de créditos
- Painéis de transparência individual
- Relatórios de impacto (kWh distribuídos, famílias atendidas, economia gerada)

3.3 - Funcionamento do sistema

3.3.1 - Cadastro de doadores — inserção manual ou via arquivo de créditos de energia.

3.3.2 - Armazenamento dos créditos em um “banco central” virtual.

3.3.3 - Cadastro de beneficiários — priorização baseada na renda ou situação de vulnerabilidade.

3.3.4 - Fila e distribuição automática de créditos, de acordo com perfil e saldo.

3.3.5 - Painéis interativos:

- Doador: créditos doados, impacto social (kWh, famílias), economia estimada.
- Beneficiário: créditos recebidos, saldo, tempo previsto de atendimento.

3.3.6 Relatórios gerenciais para administração e prestação de contas.

3.4 - Resultados esperados

3.4.1 - Alcance rápido de famílias com necessidades energéticas.

3.4.2 - Transparência total das operações.

3.4.3 - Prova de conceito para futuras parcerias e escalabilidade nacional.

4 - Requisitos Essenciais

Esses são os obrigatórios para o sistema funcionar:

4.1 - Cadastro e autenticação de perfis: Doador, beneficiário, administrador

4.2 - Registro de créditos de energia: Entrada manual/documental, sem depender de API externa

4.3 - Fila de espera e distribuição automática de créditos: Com base em critérios sociais

4.4 - Painéis básicos e responsivos para doador e beneficiário: Saldo, histórico

4.5 - Relatórios mensais de impacto: kWh doados, famílias atendidas, economia financeira

4.6 - Segurança : LGPD + criptografia de senhas + HTTPS

4.7 - Módulo de Transparência: Usuário identificado conforme perfil escolhido pode ver estatísticas agregadas como total de kWh doados, número de famílias atendidas, impacto social em reais, posição fila, consumo médio.

4.8 - Simulação de impacto para o doador: "Uma doação de X kWh contribui para o atendimento energético de Y famílias por um período estimado de Z meses”

4.9 - Feedback dos doadores: Sistema coleta quantidade do kWh doada e retorna quantos kgs de CO_2 foi reduzido.

4.10 - Dashboard em tempo real para administradores: Monitoramento de créditos disponíveis, fila de espera e distribuição em andamento.

4.11 - Auditoria de transações: Registro de todas as operações para garantir confiabilidade e combater fraudes.

5 - Regras de Negócio

5.1 - Critérios de Priorização

5.1.1 - A prioridade na fila será definida inicialmente pela Renda Familiar, menor renda = maior prioridade

5.1.2 - Em caso de empate ou como segundo critério, será utilizado o Consumo Médio de Energia (kWh), quanto menor o consumo, maior a prioridade

5.2 - Regras de Distribuição Automática

5.2.1 - A distribuição seguirá o modelo de distribuição proporcional, levando em conta os créditos disponíveis e a posição dos beneficiários na fila. Exemplo: se um doador disponibilizar 100 kWh e houver 4 famílias na fila, a alocação será feita proporcionalmente ao peso calculado pelos critérios de prioridade.

5.3 - Condições de Permanência na Fila

5.3.1 - Caso o beneficiário já tenha sido atendido recentemente, será avaliada a sua real necessidade.

5.3.2 - Se ainda houver necessidade comprovada, ele continua recebendo créditos, caso contrário, ele será realocado para o final da fila, dando espaço para outras famílias em situação mais crítica.

5.4 - Transparência e Auditoria

5.4.1 - Cada transação de distribuição de energia deve:

- Gerar um registro no Histórico de Créditos.
- Gerar também um registro no LogAuditoria, garantindo rastreabilidade e segurança.

5.4.2 - A plataforma disponibilizará relatórios dinâmicos (mensais, semanais ou diários, conforme escolha do usuário).

5.4.3 - Os relatórios deverão apresentar:

- Total de créditos distribuídos.

- Número de famílias atendidas.
- Critérios usados para ranqueamento.

5.4.4 - Visão do Doador: acesso ao impacto de suas doações (quantidade doada, famílias beneficiadas, impacto social estimado).

5.4.5 - Visão do Beneficiário

- Saldo de créditos de energia recebidos no mês.
- Para quem está na fila de espera, o rankeamento atual e previsão de atendimento.

5.5 - Regras de Negócio Extras

5.5.1 - Flexibilidade: o administrador poderá ajustar os pesos dos critérios de priorização.

Exemplo de configuração inicial:

Renda = 50%

Número de Moradores = 30%

Tempo de Fila = 20%

5.5.2 - Reaproveitamento de Créditos Expirados

Créditos não utilizados ou vencidos retornarão automaticamente ao Banco Central de Créditos, entrando novamente no ciclo de distribuição.

2 Modelagem de Negócios

A modelagem apresentada da plataforma Energia para Todos representa de forma clara e organizada a interação entre os diferentes atores do sistema. O diagrama de atividades demonstra os principais fluxos funcionais e como cada participante contribui para o processo de doação e recebimento de benefícios.

Os atores principais são: Doador, Beneficiário, Administrativo, Sistema da ONG e Auditoria. Cada um possui responsabilidades específicas dentro do fluxo:

Doador: realiza o cadastro, escolhe o valor para doar, efetua a doação e pode consultar o histórico de doações. Após a doação, seus dados são analisados e validados pelo sistema, podendo ser aprovados ou reprovados.

Beneficiário: efetua o cadastro, aguarda na fila de contemplação, recebe notificações de aprovação e pode consultar seus benefícios e créditos mensais durante o período contratual de 12 meses.

Administrativo: acompanha e analisa relatórios, verificando o andamento das operações e o status de doadores e beneficiários.

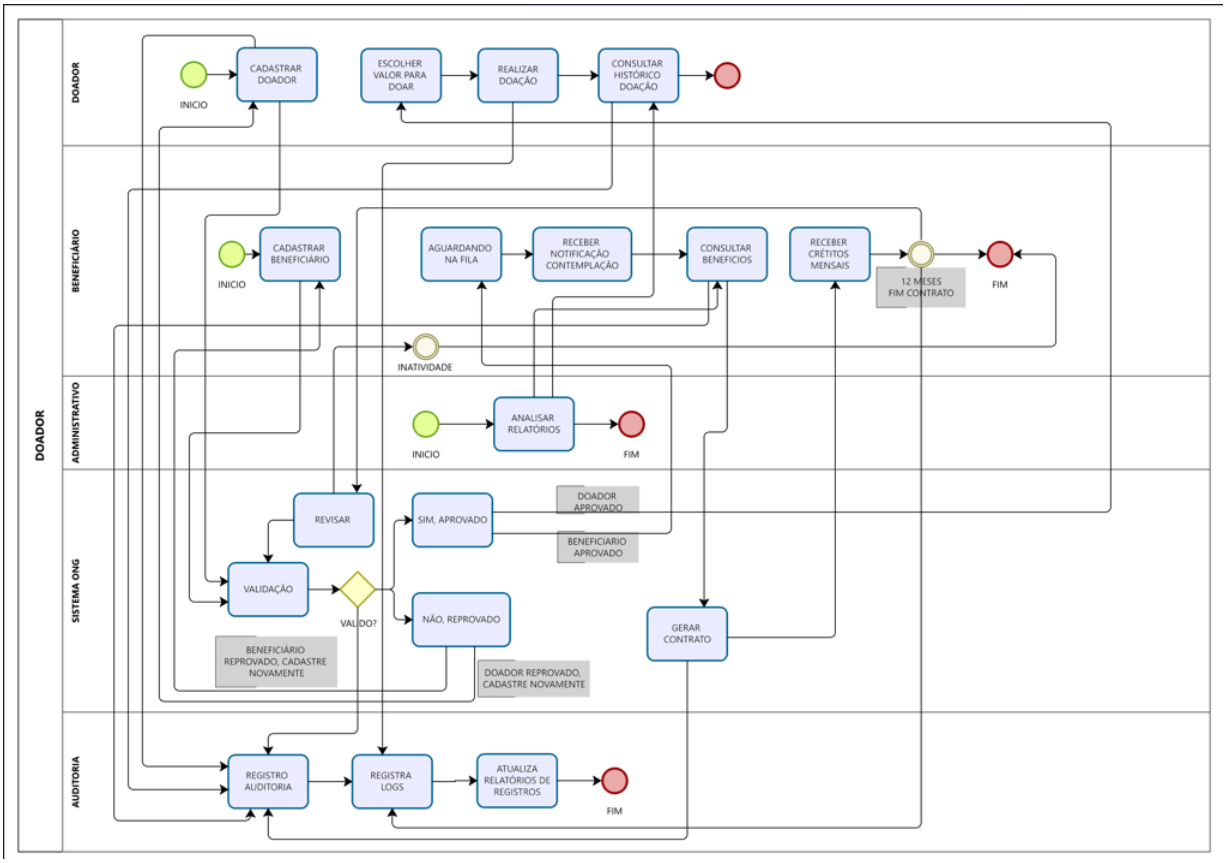
Sistema da ONG: realiza automaticamente a validação dos cadastros e das doações. Caso estejam válidos, gera o contrato correspondente; se não, orienta para o recadastro.

Auditoria: registra todas as ações do sistema, gerando logs e relatórios que asseguram a transparência e a rastreabilidade das atividades.

O fluxo geral do processo ocorre de forma integrada e cíclica, começando com o cadastro dos usuários (doadores e beneficiários), passando pela validação e análise administrativa, até a geração de contratos e o registro em auditoria.

Essa modelagem garante uma visão detalhada e coerente do funcionamento do sistema, promovendo transparência, segurança e responsabilidade social em todas as etapas do processo de doação e recebimento de benefícios.

Figura 1 – Fluxo de Processos Modelagem de Negócios



Fonte: Autor Próprio, 2025

A modelagem UML aplicada à plataforma Energia para Todos possibilita compreender de forma visual como os diferentes usuários interagem com o sistema. Os diagramas de caso de uso e atividades estruturam os principais fluxos e demonstram a colaboração entre os atores que compõem o ecossistema de doação.

O diagrama de caso de uso evidencia os papéis de Doador, Beneficiário, Administrador, Sistema da ONG e Auditoria. Cada ator realiza operações específicas dentro da plataforma:

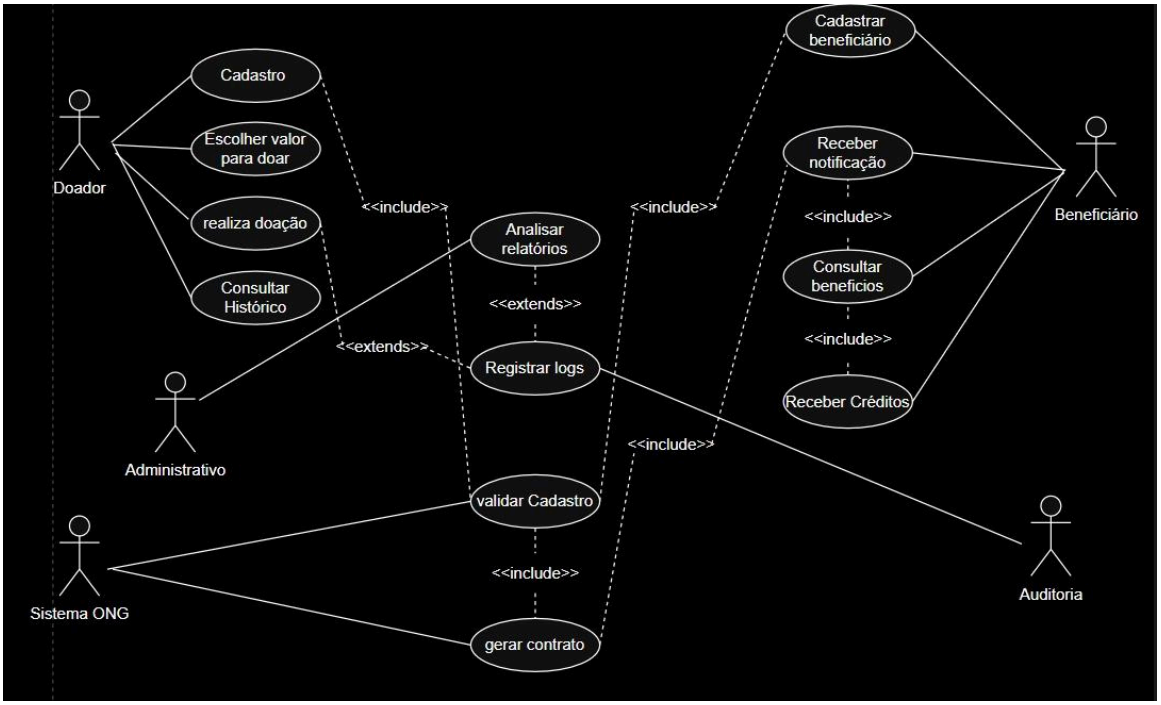
- O Doador pode se cadastrar, escolher valores para doar, consultar seu histórico e realizar doações, que passam por processos de análise e registro de logs.
- O Beneficiário realiza o cadastro, entra na fila de contemplação, recebe notificações e consulta seus benefícios.

- O Administrador é responsável por validar cadastros, analisar relatórios e redistribuir créditos, mantendo o equilíbrio e a transparência do sistema.
- O Sistema da ONG executa automaticamente funções de validação, gestão de créditos e geração de contratos, enquanto a Auditoria acompanha e registra logs de todas as operações realizadas.

O diagrama de atividades complementa essa visão ao representar os fluxos de forma sequencial e funcional. Nele, observam-se etapas como o cadastro de doadores e beneficiários, a aprovação administrativa, a distribuição de créditos e o registro em banco central de créditos. Cada ciclo — doador, beneficiário, administrativo, sistema e auditoria — é interligado, formando um processo contínuo e automatizado.

Essa modelagem garante clareza no entendimento das operações, reforça a segurança e permite o monitoramento eficiente de todas as etapas, assegurando rastreabilidade e responsabilidade social.

Figura 2 – Fluxo de Processos UML: Caso de Uso e Atividades



Fonte: Autor Próprio, 2025

A modelagem de classes do projeto Energia para Todos estrutura o sistema de forma orientada a objetos, representando as entidades centrais e seus relacionamentos. O diagrama demonstra uma arquitetura robusta e bem distribuída entre as camadas de domínio, serviço e controle.

A classe abstrata PerfilUsuario é a base do sistema, contendo atributos e métodos compartilhados entre as subclasses Doador, Beneficiário e Administrador. Essa herança garante a reutilização de código e padronização dos dados de usuários.

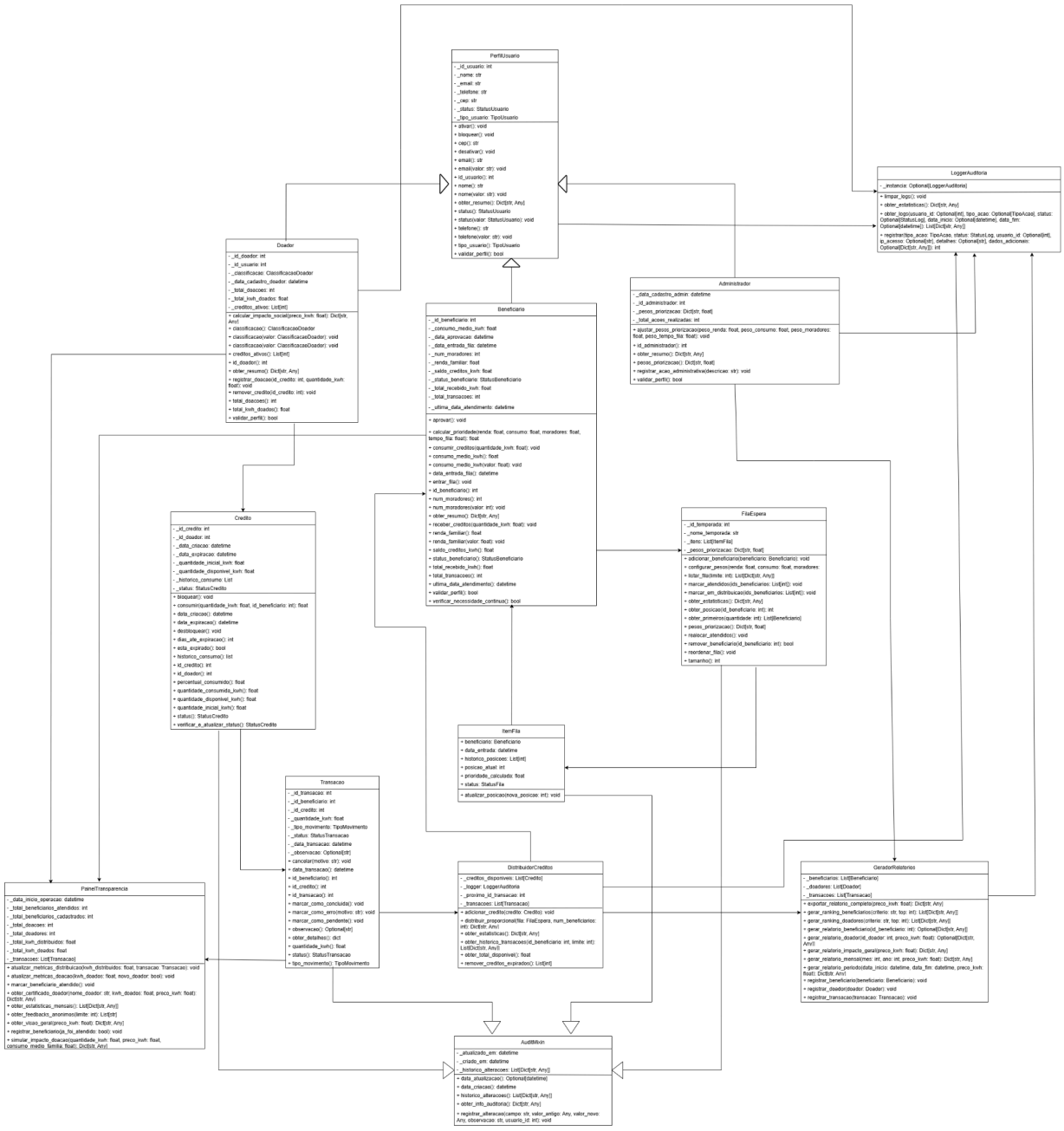
A classe Doador possui métodos para realizar doações, consultar histórico e gerenciar créditos, enquanto a classe Beneficiário inclui operações de solicitação de ajuda, atualização de status e confirmação de recebimento de créditos. Já o Administrador tem funções voltadas à validação de cadastros, análise de relatórios e distribuição de créditos entre beneficiários.

Outras classes fundamentais incluem:

- Credito, que representa os créditos energéticos e financeiros movimentados nas transações;
- Transacao, que registra cada operação com valores, datas, status e participantes;
- FilaEspera, que organiza a ordem de beneficiários aguardando contemplação;
- DistribuicaoCredito, que gerencia a alocação dos créditos disponíveis;
- Auditoria e LogRelatorio, que garantem o controle e rastreamento das atividades;
- ControleFinanceiro, responsável pelo acompanhamento de fluxo e saldo dos créditos.

Essa estrutura modular e coesa reforça a manutenção e escalabilidade do sistema, permitindo fácil expansão para novos perfis ou funcionalidades. O diagrama de classes evidencia como a abstração, herança e encapsulamento são aplicados de forma prática para garantir consistência e eficiência no projeto.

Figura 3 – Fluxo de Processos UML: Diagrama de Classes



Fonte: Autor Próprio, 2025

3 Programação Orientada a Objetos

Abaixo mostra como a estrutura do sistema foi organizada em classes que representam as principais entidades do domínio: perfis de usuário (Administrador, Doador, Beneficiário), créditos, fila de espera, transações e serviços de negócio (distribuição, relatórios e painel). Essas classes modelam o ciclo completo desde o cadastro e auditoria até a distribuição de kWh e geração de relatórios permitindo um sistema coeso, modular e alinhado à POO.

CLASSE USUÁRIO (PerfilUsuario)

Representa um usuário base do sistema, do qual derivam Administrador, Doador e Beneficiário. Centraliza dados pessoais e estado do perfil.

Atributos

- private int _id_usuario → Identificador único do usuário.
- private String _nome → Nome completo.
- private String _email → E-mail para contato/autenticação.
- private String _telefone → Telefone de contato.
- private String _cep → CEP do usuário.
- private TipoUsuario _tipo_usuario → Papel do usuário (ADMINISTRADOR, DOADOR, BENEFICIARIO).
- private StatusUsuario _status → Estado do perfil (ATIVO, INATIVO, PENDENTE, BLOQUEADO).

Construtor

- PerfilUsuario(...) (abstrato) → Utilizado pelas subclasses para inicialização padronizada.

- **Métodos**

- ativar() / desativar() / bloquear() → Gestão do ciclo de vida do perfil.
- validar_perfil() → Regras mínimas de consistência (e-mail, telefone, etc.).
- *Getters/Setters* das propriedades principais.

Conceitos Relacionais Utilizados

- **Herança**: base para Administrador, Doador e Beneficiário.
- **Abstração**: impede instância direta de PerfilUsuario.

Possibilidades de Expansão

- Validação de formato (e-mail/CEP) e *value objects* para telefone/CEP.
- Regras de antifraude e *rate limiting* para alteração de status.

CLASSE ADMINISTRADOR

Especializa PerfilUsuario com poderes de configuração e auditoria.

Atributos

- private int _id_administrador → Identificador interno.
- private datetime _data_cadastro_admin → Quando virou administrador.
- private dict _pesos_priorizacao → Pesos (0–1) usados para priorizar a fila (ex.: renda, consumo, moradores, tempo de espera).
- private int _total_acoes_realizadas → Métrica de operações administrativas executadas.

Construtor

- Administrador(id_administrador, id_usuario, nome, email, telefone, cep, data_cadastro?)
  Inicializa o perfil como ADMINISTRADOR e ativa trilhas de auditoria.

Métodos

- ajustar_pesos_priorizacao(...) → Atualiza pesos da fila.
- registrar_acao_administrativa(descricao) → Grava ação no histórico de auditoria.
- obter_resumo() → Snapshot do administrador (quantidade de ações, data de cadastro).
- validar_perfil() / __repr__().

Conceitos Relacionais Utilizados

- **Herança** (de PerfilUsuario) + **Mixin** (*AuditMixin*) para auditoria.
- **Composição** indireta com FilaEspera via pesos.

Possibilidades de Expansão

- Controle de permissões por *feature flags*.
- Histórico de versões dos pesos com rollback.

CLASSE DOADOR

Especializa PerfilUsuario e mantém o saldo/estatísticas de doação.

Atributos

- private int _id_doador → Identificador interno.
- private ClassificacaoDoador _classificacao → PESSOA_FISICA | PESSOA_JURIDICA.
- private float _total_kwh_doados → Total de kWh doados.
- private int _total_doacoes → Número de doações.
- private float _saldo_disponivel_kwh → Saldo disponível para novas distribuições.

Construtor

- Doador(id_doador, id_usuario, nome, email, telefone, cep, classificacao).

Métodos

- doar_credito(kwh, validade_dias) → Registra um novo lote (gera Credito).
- obter_resumo() / validar_perfil() / __repr__().

Conceitos Relacionais Utilizados

- **Herança** de PerfilUsuario; **Agregação** com Credito (créditos existem associados ao doador).

Possibilidades de Expansão

- Regras fiscais (recibos/nota).
- Limites dinâmicos por perfil (PF/PJ).

CLASSE BENEFICIÁRIO

Especializa PerfilUsuario e mantém dados socioenergéticos e saldo recebido.

Atributos

- private int _id_beneficiario
- private float _renda_familiar
- private int _num_moradores
- private float _consumo_medio_kwh
- private datetime _data_entrada_fila / _data_aprovacao
- private StatusBeneficiario _status_beneficiario → AGUARDANDO_APROVACAO, APROVADO, etc.
- private float _saldo_creditos_kwh / _total_recebido_kwh / _total_transacoes
- private datetime _ultima_data_atendimento

Construtor

- Beneficiario(id_beneficiario, id_usuario, nome, email, telefone, cep, renda_familiar, consumo_medio_kwh, num_moradores?).

Métodos

- entrar_fila() / aprovar()
- creditar(kwh) / debitar(kwh)
- calcular_prioridade(agora, pesos) → Score para ordenação na fila.
- obter_resumo() / validar_perfil() / __repr__().

Conceitos Relacionais Utilizados

- **Herança** de PerfilUsuario; participa de **FilaEspera** e **Transacao**.
- **Composição** lógica: seu saldo depende de transações aprovadas.

Possibilidades de Expansão

- Regras de elegibilidade (faixa de renda/região).
- Limites de consumo por período.

CLASSE CRÉDITO (Credito)

Representa um lote de kWh doados, com validade e consumo parcial.

Atributos

- private int _id_credito / private int _id_doador
- private float _quantidade_inicial_kwh / _quantidade_utilizada_kwh
- private datetime _valido_ate
- private StatusCredito _status → DISPONIVEL, PARCIALMENTE_UTILIZADO, ESGOTADO, EXPIRADO, BLOQUEADO.

Construtor

- Credito(id_credito, id_doador, quantidade_inicial_kwh, data_expiracao?, meses_validade=12).

Métodos

- usar(kwh) → Consumo parcial e atualização de status.
- bloquear() / desbloquear()
- verificar_e_atualizar_status() / __repr__().

Conceitos Relacionais Utilizados

- **Agregação** com Doador; **Associação** com Transacao.

Possibilidades de Expansão

- Políticas de expiração renovável.
- Travas por *compliance* (bloqueio cautelar).

CLASSE ITEM DE FILA (ItemFila)

Envelope de Beneficiário na fila com posição, prioridade e histórico.

Atributos

- Beneficiario beneficiario
- datetime data_entrada
- int posicao_atual
- float prioridade_calculada
- StatusFila status → AGUARDANDO, EM_DISTRIBUICAO, ATENDIDO, REMOVIDO
- List historico_posicoes

Construtor

- ItemFila(beneficiario, data_entrada?).

Métodos

- atualizar_posicao(nova_posicao) → Registra histórico.
- calcular_prioridade(agora, pesos) → Usa renda, consumo, moradores, tempo em fila.

- __repr__().

Conceitos Relacionais Utilizados

- **Composição**: o item é o “contêiner” do beneficiário na Fila.

CLASSE FILA DE ESPERA (FilaEspera)

Orquestra priorização e ordenação dos beneficiários.

Atributos

- private List _itens → Lista de ItemFila.
- private dict _pesos_priorizacao → mesmos pesos do Administrador.

Construtor

- FilaEspera(pesos_priorizacao?).

Métodos

- configurar_pesos(...)
- adicionar_beneficiario(beneficiario) → Cria ItemFila.
- ordenar_por_prioridade
- realocar_atendidos()
- obter_estatisticas() / listar_fila(limite?) / __repr__().

Conceitos Relacionais Utilizados

- **Agregação** de ItemFila; coopera com DistribuidorCreditos.

CLASSE TRANSAÇÃO (Transacao)

Registra a movimentação de kWh entre crédito e beneficiário.

Atributos

- private int _id_transacao
- private int _id_beneficiario / private int _id_credito
- private float _quantidade_kwh
- private TipoMovimento _tipo_movimento → DISTRIBUICAO, ESTORNO, AJUSTE, EXPIRACAO
- private StatusTransacao _status → CONCLUIDA, PENDENTE, CANCELADA, ERRO
- private String _motivo (opcional)
- private datetime _data_registro

Construtor

- Transacao(id_transacao, id_beneficiario, id_credito, quantidade_kwh, tipo_movimento?, observacao?).

Métodos

- concluir() / marcar_como_erro(motivo) / cancelar(motivo)
- obter_detalhes().

Conceitos Relacionais Utilizados

- **Associação** com Credito e Beneficiario; **Agregação** pelos serviços.

Possibilidades de Expansão

- Trilha de aprovação (workflow) e dupla validação.
- Integração contábil/fiscal.

CLASSE DISTRIBUIDOR DE CRÉDITOS (DistribuidorCreditos)

Serviço que executa a lógica de negócio de distribuição automática e proporcional de kWh para a Fila.

Atributos

- private List _creditos_disponiveis
- private LoggerAuditoria _logger
- Métricas internas (histórico, totais).

Construtor

- DistribuidorCreditos(logger?).

Métodos

- adicionar_credito(credito) / remover_creditos_expirados(agora)
- disponibilizar_fila(fila)
- distribuir(agora, fila) → Gera Transacao, atualiza Beneficiario e Credito.
- Consultas: obter_total_disponivel(), obter_historico_transacoes(id_beneficiario, limite?), obter_estatisticas().

Conceitos Relacionais Utilizados

- **Composição** de regras sobre Credito, FilaEspera, Transacao.
- **Observabilidade** via LoggerAuditoria.

Possibilidades de Expansão

- Estratégias pluggables (ex.: proporcional, teto por família, *fair share*).

CLASSE GERADOR DE RELATÓRIOS (GeradorRelatorios)

Gera relatórios de impacto e estatísticas (geral, por doador, por beneficiário, rankings).

Atributos

- Coleções internas de Doador, Beneficiario, Transacao.

Construtor

- GeradorRelatorios().

Métodos

- Registro: registrar_doador(d), registrar_beneficiario(b), registrar_transacao(t)
- Relatórios:
    - gerar_relatorio_impacto_geral(preco_kwh)
    - gerar_relatorio_doador(id) / gerar_relatorio_beneficiario(id)
    - gerar_ranking_beneficiarios(criterio, top)
    - exportar_relatorio_completo(preco_kwh)

Conceitos Relacionais Utilizados

- **Consulta** de dados agregados de domínio; **separação CQRS** (leitura sem mutação).

CLASSE PAINEL DE TRANSPARÊNCIA (PainelTransparencia)

Fornece visão pública agregada, sem expor dados pessoais.

Atributos

- Acumuladores de métricas: total doado/distribuído, número de doadores/beneficiários atendidos, etc.

Construtor

- PainelTransparencia().

Métodos

- Atualizações: atualizar_metricas_doacao(kwh_doados, novo_doador?), atualizar_metricas_distribuicao(kwh_distribuidos, novo_beneficiario_atendido?)
- Exposição: obter_painel_publico() / obter_feedbacks_anonimos(limite?)
- Reconhecimento: obter_certificado_doador(nome_doador, kwh_doados, preco_kwh).

Conceitos Relacionais Utilizados

- **Agregação** de dados derivados de Transacao e registros do sistema.

CLASSE LOGGER DE AUDITORIA (LoggerAuditoria)

Singleton responsável por registrar e gerenciar logs de eventos do sistema.

Atributos

- private static LoggerAuditoria _instancia
- private List _logs → histórico de eventos com data/tipo/status/detalhes/usuário.

Construtor

- __new__() garante a unicidade (Singleton).

Métodos

- registrar(tipo_acao, status, detalhes, usuario_id?)
- listar_logs(limite?), filtrar_por_tipo(tipo), filtrar_por_status(status)
- obter_estatisticas(), limpar_logs().

Conceitos Relacionais Utilizados

- **Cross-cutting concern** (transversal) usado por serviços e entidades auditáveis.

MIXIN DE AUDITORIA (AuditMixin)

Acoplar trilha de auditoria a qualquer classe de domínio (Doador/Beneficiário/Administrador).

Atributos

- private datetime _criado_em, _atualizado_em

- private List _historico_alteracoes

Construtor

- AuditMixin() → Inicializa a trilha.

Métodos

- registrar_alteracao(campo, valor_antigo, valor_novo, observacao, usuario_id)
- obter_info_auditoria()
- *Propriedades*: data_criacao, data_atualizacao, historico_alteracoes.

Conceitos Relacionais Utilizados

- **Mixin** (composição horizontal) para reutilização de comportamento.

ENUMERAÇÕES (principais)

- TipoUsuario → DOADOR, BENEFICIARIO, ADMINISTRADOR.
- StatusUsuario → ATIVO, INATIVO, PENDENTE, BLOQUEADO.
- ClassificacaoDoador → PESSOA_FISICA, PESSOA_JURIDICA.
- StatusBeneficiario → AGUARDANDO_APROVACAO, APROVADO, EM_ATENDIMENTO, SUSPENSO, INATIVO.
- StatusCredito → DISPONIVEL, PARCIALMENTE_UTILIZADO, ESGOTADO, EXPIRADO, BLOQUEADO.
- StatusFila → AGUARDANDO, EM_DISTRIBUICAO, ATENDIDO, REMOVIDO.
- TipoMovimento → DISTRIBUICAO, ESTORNO, AJUSTE, EXPIRACAO.
- StatusTransacao → CONCLUIDA, PENDENTE, CANCELADA, ERRO.
- TipoAcao (auditoria) → LOGIN, LOGOUT, CADASTRO, DOACAO, DISTRIBUICAO, ALTERACAO_PERFIL, AJUSTE_PESO, GERACAO_RELATORIO, EXCLUSAO, EXPIRACAO_CREDITO.
- StatusLog (auditoria) → SUCESSO, FALHA.

4 Relação de Artefatos de Banco de Dados II

O banco de dados do sistema Energia Para Todos foi desenvolvido com o objetivo de armazenar, organizar e dar suporte às operações do sistema de gestão de energia descentralizada, contemplando funcionalidades como cadastro de usuários, controle de transações de créditos energéticos, gerenciamento de equipamentos, histórico de uso, trocas de créditos e registros de produção e consumo. A estrutura foi planejada para garantir integridade, rastreabilidade e escalabilidade, atendendo às necessidades de clientes, fornecedores e administradores do sistema.

A estrutura contempla:

- cadastro e autenticação de usuários
- perfis diferenciados (doador, beneficiário, administrador)
- registro de créditos, transações, fila de espera
- controle socioeconômico (renda, consumo, moradores)
- auditoria completa de atividades
- segurança de credenciais
- parâmetros configuráveis do sistema

A modelagem garante integridade, rastreabilidade, governança e escalabilidade.

Legenda de Tipos de Dados:

- **INT** → Número inteiro.
- **DECIMAL(10,2)** → Número com até 10 dígitos, sendo 2 casas decimais.
- **VARCHAR(x)** → Texto com até “x” caracteres.
- **CHAR(x)** → Cadeia de texto fixa de tamanho “x”.
- **BOOLEAN** → Verdadeiro (1) ou Falso (0).
- **DATETIME** → Data e hora (YYYY-MM-DD HH:MM:SS).

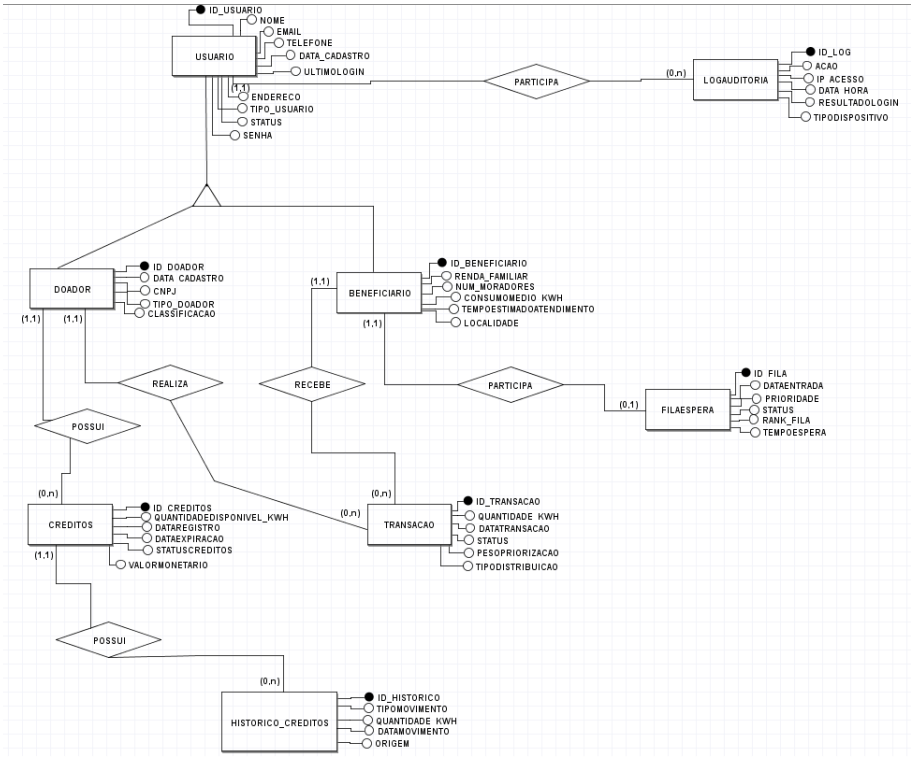
Cada entidade atribuída no banco de dados foi estruturada conforme a necessidade do sistema como um todo. Os atributos foram escolhidos e declarados com base na sua

relevância e função, garantindo a integridade dos dados, rastreabilidade, suporte a filtros de buscas, e funcionalidades como geração de nota fiscal e controle de status de operação.

Essa modelagem visa proporcionar uma base sólida e escalável, além de manter a organização e a eficiência no armazenamento e recuperação de informação.

Sendo assim, as relações entre as entidades são representadas e aplicadas na prática, conforme ilustrado na Figura 4 abaixo.

FIGURA 4 - FLUXO CONCEITUAL BANCO DE DADOS



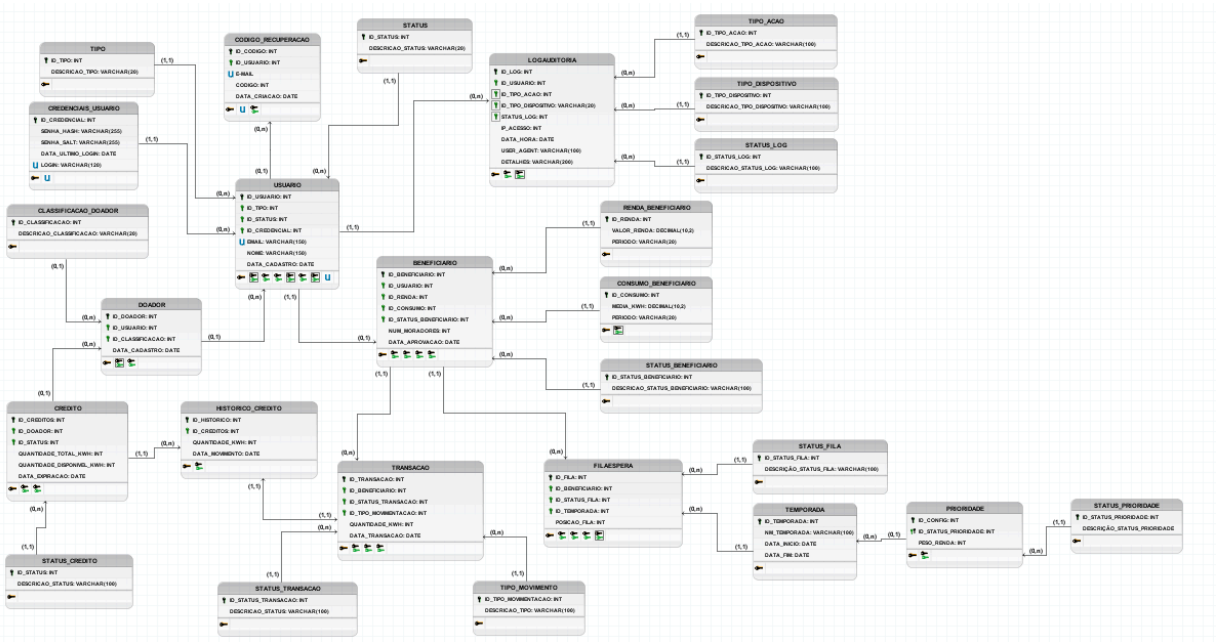
Fonte: Autor Próprio, 2025

O modelo lógico desenvolvido foi estruturado de acordo com os requisitos e funcionalidades essenciais do sistema. Cada tabela e relacionamento foi definido com base na lógica de negócio, garantindo consistência, integridade referencial e suporte às operações do sistema, como transações de compra e venda, controle de equipamentos e gerenciamento de usuários.

Essa modelagem tem como objetivo oferecer uma estrutura eficiente e bem organizada para o armazenamento e recuperação dos dados, além de facilitar a escalabilidade e manutenção futura do sistema.

Dessa forma, o modelo lógico reflete as conexões e dependências entre as entidades envolvidas, conforme representado na Figura 5 a seguir.

FIGURA 5 - FLUXO LÓGICO BANCO DE DADOS



Fonte: Autor Próprio, 2025

O modelo físico do banco de dados foi desenvolvido para traduzir a estrutura lógica em tabelas e colunas, com tipos de dados e restrições que garantem a integridade e a performance. Ele reflete o esquema final do banco de dados, otimizando o armazenamento e a recuperação de informações para o sistema de gestão de energia descentralizada, conforme representado no código fonte a seguir:

```
-- ============================================================
-- FASE 1: CRIAR TODAS AS TABELAS (SEM CHAVES ESTRANGEIRAS)
-- ============================================================
-- Tabelas de Apoio
```

```
CREATE TABLE tipo_usuario (
    id_tipo BIGSERIAL PRIMARY KEY,
    descricao_tipo VARCHAR(20) NOT NULL
);
CREATE TABLE status (
    id_status BIGSERIAL PRIMARY KEY,
    descricao_status VARCHAR(20) NOT NULL
);
CREATE TABLE classificacao_doador (
    id_classificacao BIGSERIAL PRIMARY KEY,
    descricao_classificacao VARCHAR(100) NOT NULL
);
CREATE TABLE status_credito (
    id_status_credito BIGSERIAL PRIMARY KEY,
    descricao_status VARCHAR(100) NOT NULL
);
CREATE TABLE tipo_movimento (
    id_tipo_movimentacao BIGSERIAL PRIMARY KEY,
    descricao_tipo VARCHAR(100) NOT NULL
);
```

```
CREATE TABLE status_transacao (
    id_status_transacao BIGSERIAL PRIMARY KEY,
    descricao_status VARCHAR(100) NOT NULL
);
CREATE TABLE tipo_acao (
    id_tipo_acao BIGSERIAL PRIMARY KEY,
    descricao_tipo_acao VARCHAR(100) NOT NULL
);
CREATE TABLE tipo_dispositivo (
    id_tipo_dispositivo BIGSERIAL PRIMARY KEY,
    descricao_tipo_dispositivo VARCHAR(100) NOT NULL
);
CREATE TABLE status_log (
    id_status_log BIGSERIAL PRIMARY KEY,
    descricao_status_log VARCHAR(100) NOT NULL
);
CREATE TABLE renda_beneficiario (
    id_renda BIGSERIAL PRIMARY KEY,
    valor_renda NUMERIC(10,2) NOT NULL,
    periodo VARCHAR(20)
```

```
);
CREATE TABLE consumo_beneficiario (
    id_consumo BIGSERIAL PRIMARY KEY,
    media_kwh NUMERIC(10,2) NOT NULL,
    periodo VARCHAR(20)
);
CREATE TABLE status_beneficiario (
    id_status_beneficiario BIGSERIAL PRIMARY KEY,
    descricao_status_beneficiario VARCHAR(100) NOT NULL
);
CREATE TABLE status_fila (
    id_status_fila BIGSERIAL PRIMARY KEY,
    descricao_status_fila VARCHAR(100) NOT NULL
);
CREATE TABLE temporada (
    id_temporada BIGSERIAL PRIMARY KEY,
    nm_temporada VARCHAR(100) NOT NULL,
    data_inicio DATE NOT NULL,
    data_fim DATE
);
```

Autenticação

```
CREATE TABLE credencial_usuario (
    id_credencial BIGSERIAL PRIMARY KEY,
    login VARCHAR(120) NOT NULL UNIQUE,
    senha_hash TEXT NOT NULL,
    senha_salt TEXT NOT NULL,
    data_ultimo_login TIMESTAMPTZ
);
```

Tabelas Centrais (Core)

- Tabela usuario

```
CREATE TABLE usuario (
    id_usuario BIGSERIAL PRIMARY KEY,
    nome VARCHAR(150) NOT NULL,
    email VARCHAR(150) NOT NULL UNIQUE,
    data_cadastro DATE NOT NULL DEFAULT CURRENT_DATE,
    id_tipo BIGINT NOT NULL,
    id_status BIGINT NOT NULL,
    cep VARCHAR(9) NOT NULL,
    id_credencial BIGINT NOT NULL UNIQUE
);
```

doador

```
CREATE TABLE doador (
    id_doador BIGSERIAL PRIMARY KEY,
    data_cadastro DATE NOT NULL,
    id_usuario BIGINT NOT NULL UNIQUE,
    id_classificacao BIGINT NOT NULL,
    razao_social VARCHAR(200),
    cnpj VARCHAR(18)
);
```

beneficiario

```
CREATE TABLE beneficiario (
    id_beneficiario BIGSERIAL PRIMARY KEY,
    num_moradores BIGINT,
    data_aprovacao DATE,
    id_usuario BIGINT NOT NULL UNIQUE,
    id_renda BIGINT NOT NULL,
    id_consumo BIGINT NOT NULL,
    id_status_beneficiario BIGINT NOT NULL
);
```

credito

```
CREATE TABLE credito (
    id_credito BIGSERIAL PRIMARY KEY,
    quantidade_disponivel_kwh NUMERIC NOT NULL,
    data_expiracao DATE,
    id_doador BIGINT NOT NULL,
    id_status_credito BIGINT NOT NULL
);
```

historico_credito

```
CREATE TABLE historico_credito (
    id_historico BIGSERIAL PRIMARY KEY,
    quantidade_kwh NUMERIC NOT NULL,
    data_movimento DATE NOT NULL,
    id_credito BIGINT NOT NULL
);
```

transacao

```
CREATE TABLE transacao (
    id_transacao BIGSERIAL PRIMARY KEY,
    quantidade_kwh NUMERIC NOT NULL,
    data_transacao DATE NOT NULL,
    id_beneficiario BIGINT NOT NULL,
```

```
    id_status_transacao BIGINT NOT NULL,
    id_tipo_movimentacao BIGINT NOT NULL,
    id_credito BIGINT NOT NULL
);
```

fila_espera

```
CREATE TABLE fila_espera (
    id_fila BIGSERIAL PRIMARY KEY,
    id_beneficiario BIGINT NOT NULL,
    prioridade BIGINT,
    data_entrada DATE,
    id_status_fila BIGINT NOT NULL,
    renda_familiar NUMERIC,
    consumo_medio_kwh NUMERIC,
    num_moradores BIGINT,
    tempo_espera_dias BIGINT
);
```

log_auditoria

```
CREATE TABLE log_auditoria (
    id_log BIGSERIAL PRIMARY KEY,
    ip_acesso VARCHAR(45),
```

```
    data_hora TIMESTAMPTZ NOT NULL,
    user_agent VARCHAR(255),
    detalhes VARCHAR(500),
    id_usuario BIGINT,
    id_tipo_acao BIGINT NOT NULL,
    id_tipo_dispositivo BIGINT NOT NULL,
    id_status_log BIGINT NOT NULL
);
```

codigo_recuperacao

```
CREATE TABLE codigo_recuperacao (
    id_codigo BIGSERIAL PRIMARY KEY,
    email VARCHAR(150) NOT NULL,
    codigo VARCHAR(6) NOT NULL,
    data_criacao TIMESTAMP NOT NULL
);
```

configuracao_sistema

```
CREATE TABLE configuracao_sistema (
    id_config SERIAL PRIMARY KEY,
    chave VARCHAR(100) UNIQUE NOT NULL,
    valor TEXT,
```

```
    descricao TEXT,
    data_atualizacao TIMESTAMP DEFAULT NOW()
);
-- ================================================================
    -- FASE 2: ADICIONAR TODAS AS CHAVES ESTRANGEIRAS (CONSTRAINTS)
-- ================================================================
-- USUARIO
ALTER TABLE usuario
        ADD CONSTRAINT fk_usuario_tipo FOREIGN KEY (id_tipo) REFERENCES
tipo_usuario(id_tipo);
ALTER TABLE usuario
      ADD CONSTRAINT fk_usuario_status FOREIGN KEY (id_status) REFERENCES
status(id_status);
ALTER TABLE usuario
          ADD CONSTRAINT fk_usuario_credencial FOREIGN KEY (id_credencial)
REFERENCES credencial_usuario(id_credencial);
-- DOADOR
ALTER TABLE doador
    ADD CONSTRAINT fk_doador_usuario FOREIGN KEY (id_usuario) REFERENCES
usuario(id_usuario);
ALTER TABLE doador
```

ADD CONSTRAINT fk_doador_classificacao FOREIGN KEY (id_classificacao) REFERENCES classificacao_doador(id_classificacao);

-- BENEFICIARIO

ALTER TABLE beneficiario

ADD CONSTRAINT fk_beneficiario_usuario FOREIGN KEY (id_usuario) REFERENCES usuario(id_usuario) ON DELETE CASCADE;

ALTER TABLE beneficiario

ADD CONSTRAINT fk_beneficiario_renda FOREIGN KEY (id_renda) REFERENCES renda_beneficiario(id_renda);

ALTER TABLE beneficiario

ADD CONSTRAINT fk_beneficiario_consumo FOREIGN KEY (id_consumo) REFERENCES consumo_beneficiario(id_consumo);

ALTER TABLE beneficiario

ADD CONSTRAINT fk_beneficiario_status FOREIGN KEY (id_status_beneficiario) REFERENCES status_beneficiario(id_status_beneficiario);

-- CREDITO

ALTER TABLE credito

ADD CONSTRAINT fk_credito_doador FOREIGN KEY (id_doador) REFERENCES doador(id_doador);

ALTER TABLE credito

ADD CONSTRAINT fk_credito_status FOREIGN KEY (id_status_credito) REFERENCES status_credito(id_status_credito);

-- HISTÓRICO

ALTER TABLE historico_credito

ADD CONSTRAINT fk_historico_credito FOREIGN KEY (id_credito) REFERENCES credito(id_credito);

-- TRANSACAO

ALTER TABLE transacao

ADD CONSTRAINT fk_transacao_beneficiario FOREIGN KEY (id_beneficiario) REFERENCES beneficiario(id_beneficiario);

ALTER TABLE transacao

ADD CONSTRAINT fk_transacao_status FOREIGN KEY (id_status_transacao) REFERENCES status_transacao(id_status_transacao);

ALTER TABLE transacao

ADD CONSTRAINT fk_transacao_tipo_movimento FOREIGN KEY (id_tipo_movimentacao) REFERENCES tipo_movimento(id_tipo_movimentacao);

ALTER TABLE transacao

ADD CONSTRAINT fk_transacao_credito FOREIGN KEY (id_credito) REFERENCES credito(id_credito);

-- FILA

ALTER TABLE fila_espera

ADD CONSTRAINT fk_fila_beneficiario FOREIGN KEY (id_beneficiario) REFERENCES beneficiario(id_beneficiario);

ALTER TABLE fila_espera

ADD CONSTRAINT fk_fila_status FOREIGN KEY (id_status_fila) REFERENCES status_fila(id_status_fila);

-- LOG

ALTER TABLE log_auditoria

ADD CONSTRAINT fk_log_usuario FOREIGN KEY (id_usuario) REFERENCES usuario(id_usuario) ON DELETE SET NULL;

ALTER TABLE log_auditoria

ADD CONSTRAINT fk_log_tipo_acao FOREIGN KEY (id_tipo_acao) REFERENCES tipo_acao(id_tipo_acao);

ALTER TABLE log_auditoria

ADD CONSTRAINT fk_log_tipo_dispositivo FOREIGN KEY (id_tipo_dispositivo) REFERENCES tipo_dispositivo(id_tipo_dispositivo);

ALTER TABLE log_auditoria

ADD CONSTRAINT fk_log_status FOREIGN KEY (id_status_log) REFERENCES status_log(id_status_log);

-- ==

-- FASE 3: CRIAR CRUDs (INSERT, UPDATE, DELETE, SELECT) DO SISTEMA DE LOGIN/CADASTRO DE USUÁRIO

-- ==

-- habilita bcrypt (crypt + gen_salt)

CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- índices para autenticação e unicidade

CREATE INDEX IF NOT EXISTS ix_credencial_login ON credencial_usuario(login);

CREATE INDEX IF NOT EXISTS ix_usuario_email ON usuario(email);

```
-- ============================================================
-- CREATE — Cadastrar usuário
-- ============================================================

CREATE — Cadastro de Usuário
BEGIN;
WITH cred AS (
    INSERT INTO credencial_usuario (login, senha_hash)
    VALUES ($1, crypt($2, gen_salt('bf')))
    RETURNING id_credencial
)
INSERT INTO usuario (nome, email, id_tipo, id_status, cep, id_credencial)
SELECT $3, $1, $4, $5, $6, cred.id_credencial;
COMMIT;
-- ============================================================
-- READ — Login (autenticação)
-- ============================================================
WITH auth AS (
  SELECT u.id_usuario, u.nome, u.sobrenome, u.email, c.id_credencial
  FROM credencial_usuario c
  JOIN usuario u ON u.id_credencial = c.id_credencial
  WHERE c.login = $1
    AND c.senha_hash = crypt($2, c.senha_hash)   -- valida senha
)
UPDATE credencial_usuario c
   SET ultimo_login = NOW()
  FROM auth
```

```
 WHERE c.id_credencial = auth.id_credencial
RETURNING
  auth.id_usuario, auth.nome, auth.sobrenome, auth.email, c.ultimo_login;
-- ================================================================
                -- UPDATE — Atualizar nome, sobrenome ou senha
-- ================================================================
UPDATE usuario
   SET nome = $2,
       sobrenome = $3,
       email = $4
 WHERE id_usuario = $1;
-- ================================================================
                -- DELETE — Excluir conta
-- ================================================================
BEGIN;
WITH dados AS (
  SELECT id_credencial
  FROM usuario
  WHERE id_usuario = $1
)
DELETE FROM usuario WHERE id_usuario = $1;
DELETE FROM credencial_usuario
 WHERE id_credencial IN (SELECT id_credencial FROM dados);
COMMIT;
```

Após a criação das tabelas e adição das chaves estrangeiras, foram definidos índices e extensões que aprimoram o desempenho e a segurança do banco, como:

Extensão pgcrypto: responsável pela criptografia das senhas de usuários por meio das funções crypt() e gen_salt('bf').

Índices de autenticação: criados para otimizar consultas nas colunas de login e e-mail (ix_credencial_login e ix_usuario_email).

Constraints de integridade: utilizam ON DELETE CASCADE e ON DELETE SET NULL para garantir a consistência dos dados durante exclusões.

Além das estruturas físicas, o banco conta com funções, views e triggers automáticas que asseguram a atualização dinâmica dos relacionamentos e o cumprimento das regras de negócio:

Função recalcular_posicoes_fila() — recalcula as posições dos beneficiários na fila de espera sempre que há alteração de prioridade ou atendimento.

View v_fila_priorizada — apresenta a visão consolidada da fila de espera, mostrando a posição e o status atual de cada beneficiário.

Trigger trigger_atualizar_fila — executa automaticamente após a conclusão de uma transação, atualizando o status do beneficiário na fila de “Aguardando” para “Atendido”.

Esses artefatos complementares fortalecem a consistência entre os dados e reduzem a necessidade de intervenção manual, tornando o sistema mais confiável e automatizado.

O modelo como um todo garante integridade referencial, rastreabilidade das ações e segurança dos dados, permitindo que o sistema funcione de forma estável e transparente.

A estrutura relacional adotada reflete diretamente os requisitos funcionais e não funcionais do projeto, possibilitando o controle completo do ciclo de energia — desde a doação até a distribuição e o registro em log das atividades.

Dessa maneira, o Artefato de Banco de Dados se consolida como um dos pilares do sistema, sustentando todas as operações com consistência, escalabilidade e alto nível de governança.

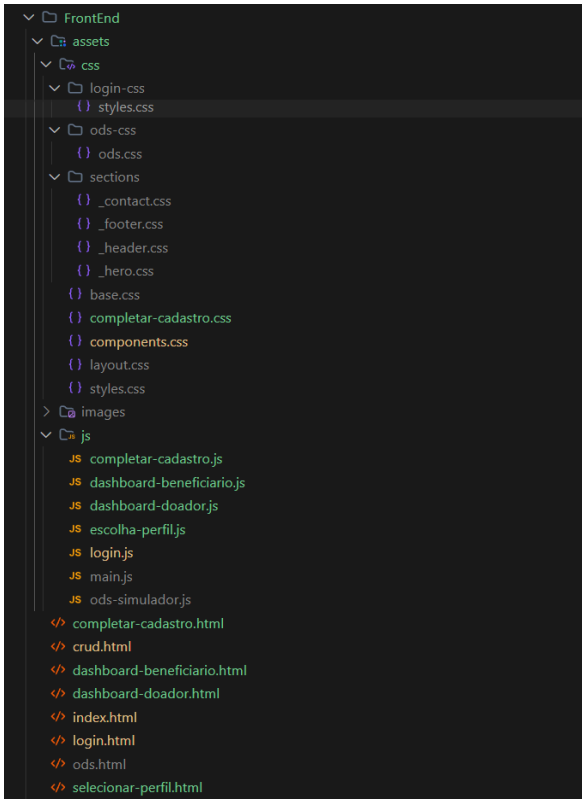
5 Relação de Artefatos de Programação para Web I

A arquitetura de front-end da plataforma Energia Para Todos foi desenvolvida utilizando HTML, CSS e JavaScript, formando a interface principal do sistema de gestão de créditos energéticos. Todo o design foi construído para reforçar uma identidade visual coesa alinhada ao tema de energia limpa, inovação social e acessibilidade digital.

A estrutura do código segue padrões modernos de desenvolvimento orientados à experiência do usuário (UX), priorizando clareza visual, responsividade, eficiência e sustentabilidade do projeto. A organização modular dos arquivos facilita a manutenção, a reutilização de componentes e a escalabilidade do sistema. Os estilos CSS estão separados em arquivos dedicados, como styles.css, dashboard.css e completar-cadastro.css, permitindo que cada página mantenha seu contexto visual isolado. Essa abordagem modular torna o sistema mais organizado e adaptável a futuras evoluções.

A Figura 6 ilustra a arquitetura de arquivos e diretórios do front-end, evidenciando a hierarquia limpa e coerente adotada para gerenciamento dos componentes visuais.

Figura 6 - Estrutura de pastas/arquivos



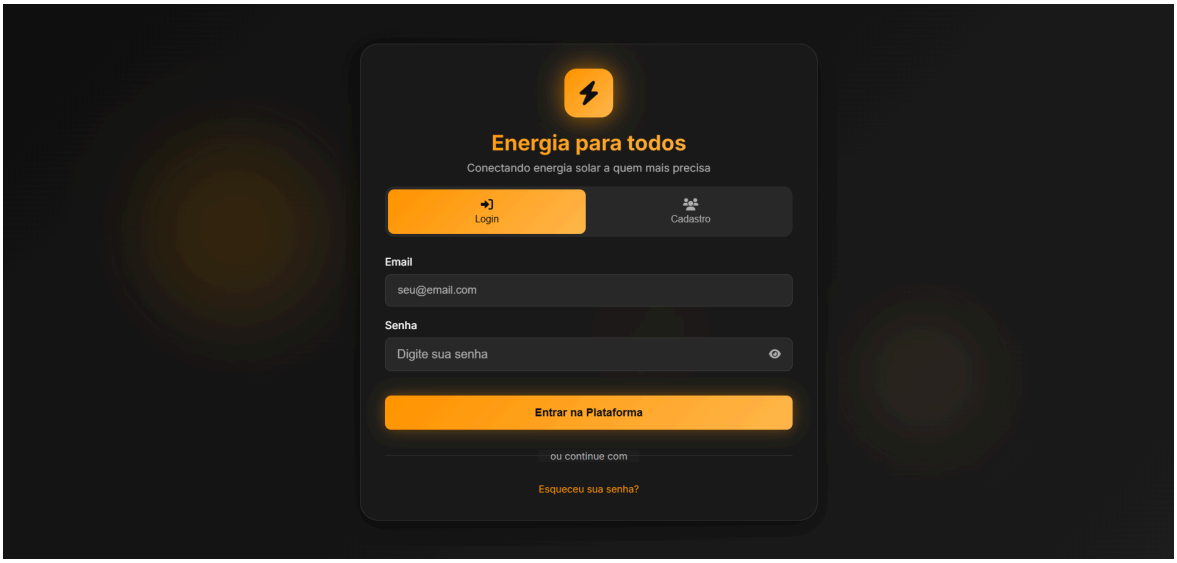
Fonte: Autor Próprio, 2025

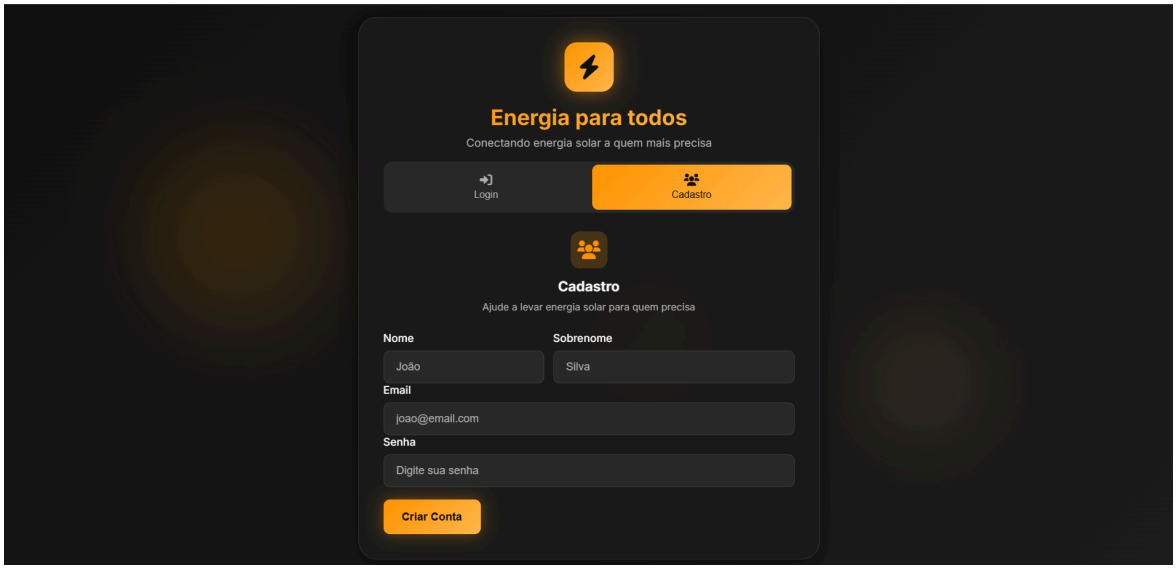
A página inicial (*index.html*) apresenta a proposta do projeto, destacando missão, impacto social e etapas de participação. Seu design utiliza seções delimitadas, foco em legibilidade e elementos visuais que conduzem intuitivamente o usuário pelo fluxo de entrada na plataforma.

A primeira interação prática ocorre por meio do arquivo *login.html*, que unifica login e cadastro em uma interface com abas dinâmicas. O uso de JavaScript permite alternar entre as seções “Entrar na Plataforma” e “Criar Conta de Doador” sem recarregar a página, representando um modelo simplificado de SPA (Single Page Application).

As Figuras 8 e 9 exemplificam o sistema de abas do *login.html*, demonstrando o fluxo contínuo entre autenticação e criação de contas.

Figura 8, 9 - Layout da página de cadastro e autenticação (login.html)





Fonte: Autor Próprio, 2025

O formulário de cadastro, composto por campos como nome completo, e-mail e senha, corresponde diretamente às exigências da camada de banco de dados, garantindo consistência entre front-end e back-end. As interações são processadas via requisições HTTP (métodos GET e POST) que conversam com as rotas definidas no *routes.py* para autenticação segura.

Após autenticação, o usuário é direcionado à tela selecionar-perfil.html, onde escolhe entre os papéis de Doador ou Beneficiário. A interface utiliza ícones e descrições orientativas, reforçando clareza no processo de escolha. O botão “Continuar” permanece inativo até que uma opção seja selecionada, comportamento controlado pelo script escolha-perfil.js, garantindo coerência e prevenindo erros de navegação.

Em seguida, o usuário acessa completar-cadastro.html, onde o formulário é construído dinamicamente. O arquivo completar-cadastro.js identifica o tipo de usuário selecionado e gera automaticamente os campos adequados dentro do elemento . Isso evita duplicação de páginas, melhora a escalabilidade do sistema e reduz riscos de inconsistência entre formulários.

Dashboard do Doador

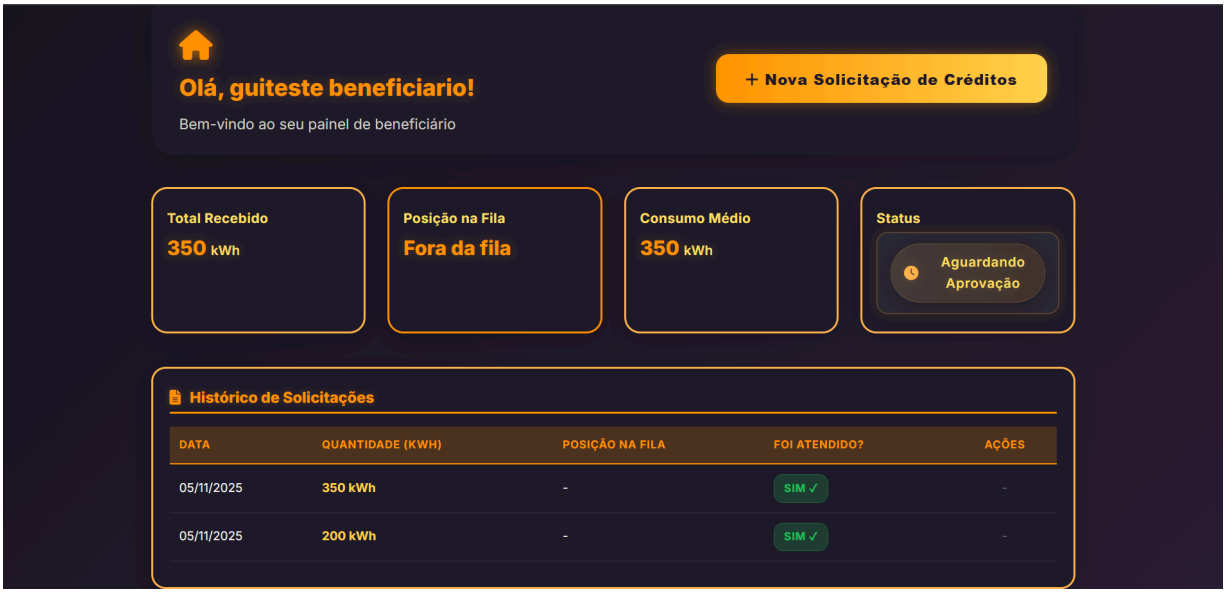
O painel do doador (Figura 10) apresenta métricas como créditos disponíveis, total doado e histórico de distribuições. O layout utiliza componentes como *dashboard-metric* a metric-*value* para organizar informações de forma limpa e responsiva. Ícones do Font Awesome e botões com efeitos modernos reforçam a estética sustentável do projeto. Os dados são carregados dinamicamente por meio do script *dashboard-doador.js*, que consome rotas específicas da API.

Figura 10 - Dashboard do Doador (dashboard-doador.html)

Fonte: Autor Próprio, 2025

O painel do beneficiário (Figura 11) apresenta informações sobre situação energética, posição na fila, média de consumo e status da solicitação. O usuário pode solicitar novos créditos através de um modal que valida limites com base no banco de dados, com lógica gerenciada pelo script *dashboard-beneficiario.js*.

Figura 11 - Dashboard do Beneficiário (dashboard-beneficiario.html)



Fonte: Autor Próprio, 2025

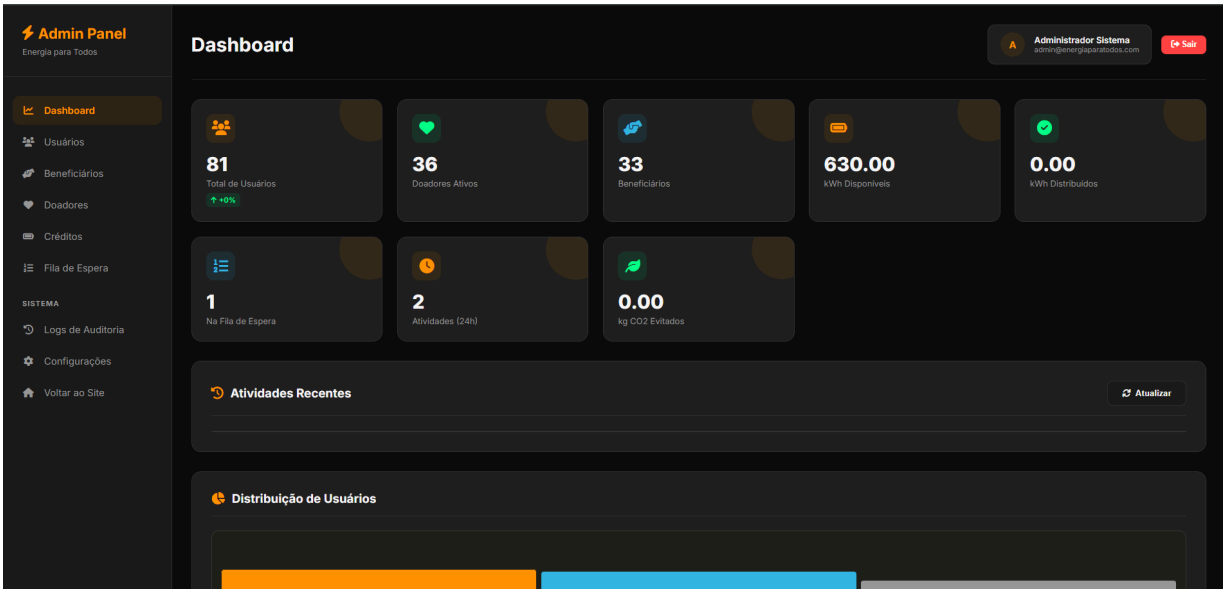
Com um modal interativo permite inserir novos créditos, fortalecendo a usabilidade. A área de “Histórico de Solicitações” acompanha atualizações em tempo real.

O arquivo crud.html funciona como o Painel Administrativo oficial do sistema. Ele centraliza todas as operações de gestão, permitindo que o administrador visualize informações, atualize registros e acompanhe métricas essenciais da plataforma.

O painel exibe dados consolidados sobre usuários, doadores, beneficiários, créditos disponíveis, créditos distribuídos e fila de espera, utilizando a view v_metricas_admin para garantir informações sempre atualizadas. Além da visualização, o administrador pode editar, atualizar ou excluir contas, realizar consultas rápidas e monitorar a atividade recente do sistema.

Dessa forma, o painel deixa de ser apenas um ambiente de testes e se torna uma ferramenta completa de gestão e controle, contribuindo para auditoria, transparência e operação segura da plataforma Energia Para Todos.

Figura 12 - Página de Operações do Administrador do Sistema (crud.html)



Fonte: Autor Próprio, 2025

Todos os artefatos do front-end enviam e recebem dados por meio de *fetch()* em JavaScript, consumindo endpoints REST em Python.

As respostas são recebidas em formato JSON, permitindo atualização instantânea sem recarregar páginas — o que configura uma arquitetura SPA simplificada.

A integração é estruturada para:

- login e autenticação
- cadastro
- listagem de créditos
- atualização de fila
- solicitação e distribuição de energia
- auditoria de ações

- sessão e cookies

A implementação do frontend reflete de forma direta a estrutura e lógica operacional da solução, alinhando-se às necessidades de transparência e eficiência do sistema.

O design responsivo, os formulários dinâmicos e a integração com o backend consolidam uma arquitetura coesa, que oferece experiência fluida para doadores e beneficiários, assegurando clareza na interação e confiabilidade no uso.

1. ACESSIBILIDADE (A11y)

O front-end considera boas práticas de acessibilidade:

- contraste adequado entre texto e fundo
- navegação por teclado em botões e formulários
- uso de *aria-label* em botões e ícones
- foco visível para inputs
- mensagens de erro legíveis
- estrutura semântica com <header>, <main> e <section>

Isso melhora a experiência de usuários com necessidades especiais e atende a recomendações de acessibilidade digital.

2. Segurança no Front-End

A aplicação implementa camadas básicas de segurança:

- sanitização de campos antes do envio
- validação dupla (front/back)
- criptografia de senhas no servidor
- proteção contra XSS com manipulação controlada do DOM
- sessões temporárias e cookies de autenticação controlados
- bloqueio de ações críticas para usuários não autenticados

3. Responsividade e Mobile-First

O sistema foi projetado para funcionar em celulares, tablets e desktops:

- uso de Flexbox e CSS Grid
- breakpoints definidos para reorganização de cards
- colunas convertidas em blocos verticais no mobile
- botões ampliados para toque
- menus recolhíveis (hambúrguer)

4. Arquitetura Visual (UX/UI)

A interface aplica conceitos modernos de design:

- paleta com tons verdes e amarelos (energia, sustentabilidade)
- uso de sombras suaves e microinterações
- ícones representativos (fa-sun, fa-home, fa-bolt)
- hierarquia tipográfica clara
- foco no conteúdo principal
- botões com feedback visual

5. Estratégia de JavaScript e Modularização

O JS utiliza:

- async/await
- fetch API
- funções puras reutilizáveis
- separação de scripts por contexto
- atualização seletiva do DOM para melhor desempenho
- tratamento centralizado de erros

Esse modelo garante escalabilidade e boa manutenção.

6. Arquitetura Front–Back–Banco

A solução segue o fluxo:

Browser → Front-end (HTML/CSS/JS) → API Python (server.py) → PostgreSQL

Isso garante:

- comunicação estruturada
- rotas organizadas
- persistência confiável
- desacoplamento entre camadas

A implementação do front-end reflete diretamente a lógica operacional da solução e está alinhada às necessidades de clareza, segurança e eficiência. O design responsivo, a construção modular, os formulários dinâmicos e a integração robusta com o back-end consolidam uma arquitetura moderna, fluida e confiável tanto para doadores quanto para beneficiários.

O conjunto dos artefatos demonstra maturidade técnica, domínio dos conceitos de programação para web e forte atenção à experiência do usuário — entregando um produto bem estruturado, escalável e pronto para evoluir.

6 Relação de Artefatos de Software e Estrutura do Projeto

A arquitetura do sistema “Energia Para Todos” foi concebida seguindo princípios modernos de Engenharia de Software, com foco em modularidade, baixa dependência entre componentes, alta coesão, segurança, escalabilidade e clareza na separação de responsabilidades.

A estrutura do projeto aplica conceitos da Arquitetura em Camadas, influenciada por princípios da Clean Architecture e pelos fundamentos SOLID e Separation of Concerns.

A solução foi dividida em três grandes camadas lógicas:

1. BackEnd (núcleo lógico + regras de negócio)
2. Conexão (API REST e controle de rotas)
3. FrontEnd (interface do usuário, experiência e interação)

Essa divisão garante independência, estabilidade e evolução contínua do sistema, permitindo que cada camada possa ser estendida ou substituída sem comprometer as demais.

6.1 BackEnd – Arquitetura, Domínio e Regras de Negócio

A camada de BackEnd foi projetada em Python com foco em clareza, organização de código e orientação a objetos (POO).
A estrutura segue o padrão de pacotes, distribuída da seguinte forma:

```
BackEnd/
├── main.py                 # Orquestra execução ponta a ponta do sistema
│
├── mixins/
│   ├── audit_mixin.py      # AuditMixin: histórico, timestamps e auditoria
│
├── models/                 # Núcleo de domínio do sistema
│   ├── administrador.py    # Classe Administrador: pesos e priorizações
│   ├── base.py             # Classe abstrata PerfilUsuario + enums
│   ├── beneficiario.py     # Entidade Beneficiário: consumo, renda, saldo
│   ├── credito.py          # Entidade Crédito: validade e controle de uso
│   ├── doador.py           # Entidade Doador: PF/PJ, histórico, doações
```

```
│   ├── fila_espera.py          # Algoritmo de ordem e prioridade
│   └── transacao.py            # Fluxo de transações energéticas
│
├── services/                   # Casos de uso / Regras de negócio
│   ├── distribuidor_creditos.py # Core da distribuição inteligente de créditos
│   ├── gerador_relatorio.py    # Métricas, PDFs e relatórios do sistema
│   └── painel_transparencia.py  # Dados públicos / governança
│
├── utils/
│   ├── database.py             # Interface com PostgreSQL
│   └── logger_auditoria.py     # Registro estruturado de atividades
│
├── extensao_sprint1.sql        # Procedures, gatilhos e funções SQL auxiliares
├── script_banco.sql            # Modelo lógico e físico completo
└── database.py                 # Conexão persistente com o banco SQL
```

Arquiteturais do BackEnd

Clean Architecture / Camadas

O sistema separa:

- Domínio (models/)
- Serviços (services/)

- Infraestrutura (utils/)
- Interface com o servidor (Conexão/)

Isso garante isolamento entre lógica de negócio e detalhes técnicos.

Princípios SOLID

- S — Classes com responsabilidade única (ex.: Credito, Beneficiario).
- O — Facilidade de extensão (novos tipos de usuário podem ser adicionados).
- L — Entidades seguem hierarquias coerentes (PerfilUsuario).
- I — Interfaces e classes abstratas organizam os comportamentos.
- D — Baixo acoplamento entre domínio e infraestrutura.

Auditoria e Rastreabilidade

O audit_mixin.py centraliza:

- timestamps
- criação
- atualização
- histórico de ações

Todos os registros sensíveis são auditados, permitindo rastreamento completo no banco.

Modularidade e Reuso

Cada serviço do diretório services/ atua como um *Caso de Uso*, podendo ser reutilizado em múltiplas rotas.

6.2 Conexão – API REST, Rotas e Comunicação Client <–> Server

A pasta Conexao/ faz a ponte entre o FrontEnd e o BackEnd, servindo como a camada de Controllers da aplicação.

Conexao/

├── routes.py # Definição de rotas REST, validações e session control

└── server.py # Servidor HTTP e registro dos endpoints

Funções principais dessa camada:

✔ Receber requisições HTTP (fetch do FrontEnd)
✔ Validar dados e sessões
✔ Chamar os serviços apropriados no BackEnd
✔ Retornar respostas em JSON
✔ Padronizar mensagens de erro, sucesso e status

Fluxo resumido da requisição:

1. Usuário realiza ação no FrontEnd (ex.: solicitar crédito).
2. O script JS chama um endpoint via fetch('/api/...').
3. server.py recebe a rota e delega para routes.py.
4. routes.py chama a lógica do BackEnd (services/).
5. O serviço acessa o domínio (models/) e o banco (utils/database.py).
6. A resposta é devolvida em JSON para o FrontEnd.

Essa arquitetura cria um pipeline claro, seguro e escalável.

6.3 FrontEnd – Arquitetura Visual, UX e Integração

O FrontEnd foi construído com HTML, CSS e JavaScript puro (Vanilla JS), aplicando boas práticas de UX, responsividade e modularização.

FrontEnd/

├── index.html

├── login.html

├── selecionar-perfil.html

```
├── completar-cadastro.html
├── dashboard-doador.html
├── dashboard-beneficiario.html
├── crud.html
│
├── assets/
│   ├── css/
│   │   ├── base.css
│   │   ├── components.css
│   │   ├── layout.css
│   │   ├── styles.css
│   │   ├── completar-cadastro.css
│   │   └── escolha-css/
│   │
│   ├── js/
│   │   ├── main.js
│   │   ├── login.js
│   │   ├── escolha-perfil.js
│   │   ├── completar-cadastro.js
│   │   ├── dashboard-doador.js
```

```
│   │   ├── dashboard-beneficiario.js
│   │   └── ods-simulador.js
│   │
│   └── images/
```

Modularização completa de CSS

- base.css → reset + tokens globais
- components.css → botões, cards, badges
- layout.css → responsividade + grids
- Arquivos específicos por página

Scripts separados por responsabilidade

Cada página tem seu próprio JS, seguindo o princípio Single Responsibility.

Integração REST Assíncrona

Os scripts usam:

- fetch()
- respostas JSON
- atualizações sem recarregar a página (SPA simplificada)
- feedbacks visuais (loaders, alertas, modais)

UX consistente

- feedback imediato
- botões com efeitos visuais
- alerts contextuais
- campos dinâmicos gerados via JS
- seleção de perfil com lógica de habilitação/desabilitação

6.4 – Fluxo Arquitetural do Sistema

O fluxo completo ocorre assim:

1. Usuário interage no FrontEnd.
2. FrontEnd envia requisição para /api/... via fetch().
3. Server/Routes tratam validação, sessão e chamam o serviço correto.
4. Serviços aplicam regras de negócio e interagem com o banco.
5. Resposta: JSON retorna ao front.
6. Interface atualizada sem recarregar a página.

Esse ecossistema cria uma arquitetura:

- reativa
- desacoplada
- escalável
- com domínio isolado da interface
- adequada para produção
- alinhada ao mercado

6.5 — Síntese Arquitetural

A arquitetura adotada apresenta maturidade técnica e alinhamento com padrões profissionais de desenvolvimento de software, garantindo modularidade, testabilidade, escalabilidade e clareza estrutural. A separação rigorosa entre domínio, infraestrutura, API e interface assegura um sistema sustentável, capaz de evoluir sem risco de regressões e preparado para operação contínua.

O uso de princípios SOLID, DRY, Separation of Concerns, modelagem relacional normalizada e comunicação REST padronizada consolida um ecossistema coerente, robusto e pronto para implantação real.

O sistema Energia Para Todos demonstra não apenas domínio técnico, mas aderência à engenharia moderna, permitindo crescimento seguro, manutenibilidade a longo prazo

e rastreabilidade completa das operações. Trata-se de uma arquitetura sólida, escalável e alinhada às melhores práticas do mercado atual.

7 Link de Acesso do GitHub

Acesse o nosso projeto no GitHub: https://github.com/oRodolfo/Energia-Para-Todos.git

8 Link de Acesso ao Cronograma do Projeto (Trello)

Acesse o nosso cronograma no Trello: https://trello.com/b/ULaBoSd6/projeto-interdisciplinar-energia-para-todos

6 Considerações Finais

O desenvolvimento desse sistema foi um aprendizado muito grande, tanto na parte técnica quanto na parte estrutural do projeto como um todo. O projeto alcançou plenamente seu objetivo acadêmico e funcional, consolidando uma solução completa, modular e alinhada às boas práticas de Engenharia de Software. Entretanto, como qualquer projeto real, também revelou desafios, pontos de atenção e oportunidades claras de melhoria, sendo eles, pontos essenciais para a maturidade do sistema e a evolução profissional do grupo inteiro.

Do ponto de vista técnico, o desenvolvimento inicial foi um dos maiores desafios do projeto: a integração entre o BackEnd e o Banco de Dados. Como nessa parte o grupo não tinha experiência prática e nem domínio técnico, foi necessário um esforço adicional para compreender conceitos como conexões persistentes, consultas parametrizadas, views, triggers e manipulação transacional. Os primeiros testes envolveram erros de sintaxe SQL, falhas de comunicação entre camadas e ajustes na modelagem lógica, o que gerou retrabalho e demandou aprofundamento técnico.

No FrontEnd, o principal obstáculo esteve relacionado à comunicação assíncrona com o BackEnd. Alguns endpoints precisaram de revisão para garantir consistência no formato JSON e padronização de respostas, o que impactou diretamente na

experiência do usuário. Além disso, a ausência de testes automatizados dificultou a detecção precoce de regressões, fazendo com que algumas falhas fossem percebidas somente após a integração final.

Apesar desse aprendizado, o sistema demonstrou grande evolução e, após a fase de ajustes e testes, alcançou eficiência na integração completa entre BackEnd, FrontEnd e Banco de Dados, aplicando uma arquitetura limpa, escalável e sustentada por princípios sólidos. As funcionalidades centrais — cadastro, autenticação, distribuição de créditos, gestão de usuários, fila de espera e dashboards — foram implementadas com consistência e coerência lógica. Além disso, a organização modular permitiu que o sistema permanecesse compreensível mesmo com o aumento da complexidade.

Dessa forma, o projeto consolidou-se como um projeto completo, socialmente relevante e tecnicamente consistente. Os erros identificados foram importantes para orientar o desenvolvimento e fortalecer a compreensão de arquitetura e boas práticas. As melhorias mapeadas pavimentam o caminho para próximos ciclos de evolução, mantendo o sistema mais seguro, eficiente, intuitivo e preparado para cenários reais.

O projeto, portanto, não apenas entrega uma solução funcional, mas também representa um marco de aprendizado, crescimento e amadurecimento técnico — servindo como base sólida para futuras versões e iniciativas profissionais ainda mais completas.

Para eventos e versões futuras, destacam-se como principais oportunidades de melhoria:

- implementação de testes automatizados (unitários e de integração) para reduzir retrabalhos;
- definição mais rígida de requisitos logo no início, evitando mudanças estruturais tardias;
- adoção de uma documentação técnica inicial mais completa (UML + diagramas de sequência + casos de uso);
- padronização mais firme de respostas da API para minimizar erros de integração;

- inclusão de métricas de desempenho e monitoramento para avaliar o comportamento do sistema sob carga;
- evolução para um modelo de deploy real (Docker/Render/Cloud) visando maior maturidade operacional;
- criação de um manual de governança do projeto, para melhorar a organização e fluxo de decisões.

Referências Bibliográficas

BRASIL. Nações Unidas no Brasil. Objetivo de Desenvolvimento Sustentável 7: Energia Acessível e Limpa. Disponível em: https://brasil.un.org/pt-br/sdgs/7. Acesso em: 27 ago. 2025.

WIKIPÉDIA. Objetivo de Desenvolvimento Sustentável 7. Disponível em: https://pt.wikipedia.org/wiki/Objetivo_de_Desenvolvimento_Sustent%C3%A1vel_7. Acesso em: 27 ago. 2025.

SANTOS, E. P.; et al. Energia e sustentabilidade humana: impacto das metas do ODS 7 no Brasil. ResearchGate, 2020. Disponível em: https://www.researchgate.net/publication/344858460_ENERGIA_E_SUSTENTABILIDADE_HUMANA_Impacto_das_metas_do_ODS_7_no_Brasil. Acesso em: 27 ago. 2025.

BRASIL. Ministério de Minas e Energia (MME). Projetos de geração distribuída de interesse social: Sumário executivo. Brasília: GIZ/MME, 2020. Disponível em: https://www.gov.br/mme/pt-br/assuntos/ee/sistemas-de-energia-do-futuro/SumrioExecutivoGDSocial_Impresso.pdf. Acesso em: 27 ago. 2025.

WIKIPÉDIA. Instituto para o Desenvolvimento de Energias Alternativas e da Auto Sustentabilidade (IDEAAS). Disponível em: https://pt.wikipedia.org/wiki/Instituto_para_o_Desenvolvimento_de_Energias_Alternativas_e_da_Auto_Sustentabilidade. Acesso em: 27 ago. 2025.