1. Visão Geral e Objetivo

Este documento detalha a estrutura, arquitetura e soluções técnicas empregadas no **Sistema de Gestão Horista (SGH)**. O objetivo principal é gerenciar o cadastro de funcionários, registro de dependentes e cálculos básicos de salários, focando na **integridade de dados** e na usabilidade através de **formulários modais (pop-ups)**.

- **Nome do Sistema:** Sistema de Gestão Horista (SGH)
- **Domínio Principal:** Gestão de Recursos Humanos (Cadastro e Folha Simples).
- **Tecnologias Core:** Python, Django, PostgreSQL/SQLite.

2. Tecnologias e Dependências Chave

2.1 Backend e Estrutura

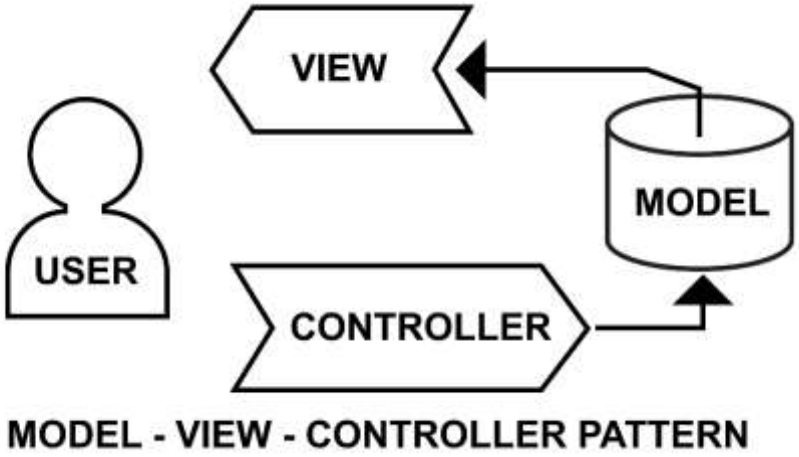
- **Framework: Django (Python)**. Base da arquitetura MVT, utilizado para ORM, roteamento, autenticação e Views genéricas (CBVs).
- **Aritmética: decimal.Decimal**. Crucialmente empregado em todos os cálculos monetários para garantir a **precisão financeira** (evitando erros de ponto flutuante).
- **Banco de Dados: PostgreSQL / SQLite**. SQLite para desenvolvimento; PostgreSQL recomendado para produção.
- **Forms: Django ModelForms** e a biblioteca crispy_forms para validação e estilização.

2.2 Frontend e Interatividade

- **Template Engine: Django Templates**.
- **Estilização: CSS (Puro)**. Estilos customizados para *layouts* e componentes (ex: .ordenar-btn, .form-container).
- **Interatividade (UX): JavaScript (Vanilla)**. Essencial para o fluxo de *pop-ups* modais (window.open) e o fechamento seguro, garantindo a atualização da página principal após o salvamento.

3. Arquitetura e Padrões de Design

A



MODEL - VIEW - CONTROLLER PATTERN

arquitetura do SGH segue o padrão **Model-View-Template (MVT)** do Django.

3.1 Camada de Domínio (Modelos)

- **Funcionario:** Modelo base, armazenando dados principais (nome_funcionario, salario_hora).
- **Salario:** Modelo detalhe.
    - Relação: ForeignKey para Funcionario.
    - Regras: Lógica de cálculo (_calc_salario_bruto, _calc_salario_liquido) centralizada no método save() do modelo.
- **DependenteElegivel:** Modelo detalhe.
    - Relação: ForeignKey para Funcionario (related_name='dependenteelegivel').
- **User:** Modelo principal de conta (autenticação).

3.2 Views (Controladores) e Fluxo de Dados

- **Padrão: Views Genéricas Baseadas em Classe (CBVs)**.

- **Reuso de Templates:** As Views (CreateView, UpdateView) utilizam o mesmo template (cadastro_att_dependente.html) injetando a variável modo ('criação' ou 'edição') via get_context_data.
- **Função de Detalhe:** A DetailView (Funcionario_detalhe) atua como painel de controle, buscando os dados relacionados (Salario e DependenteElegivel) no método get_context_data.

---

4. Soluções e Regras de Negócio Implementadas

4.1 Validação e Integridade de Dados

- **Unicidade:** O UsuarioForm.clean() utiliza consultas com .exclude(pk=self.instance.pk) para garantir que o username e email sejam únicos, tratando corretamente os cenários de criação e edição.
- **Obrigatoriedade Condicional:** Campos como username e password são definidos como obrigatórios **apenas na criação de novos usuários** (if not self.instance.pk:).
- **Elegibilidade de Dependentes (Regra Crítica):** O **DependenteElegivelForm.clean()** valida que a idade máxima permitida para cadastro é de **14 anos** (idade_data > 14), aplicando a regra de negócio para Salário Família.
- **Validação Numérica:** Checagem de valores negativos (< 0.00) e validação implícita de tipo numérico via DecimalField.

4.2 Fluxo de Formulários Contextuais (*Pop-ups*)

A manipulação da Foreign Key (FK) é gerenciada internamente:

1. **Injeção de Instância:** A View envia a instância completa do Funcionario via kwargs['funcionario_instance'] para o formulário.
2. **Ocultação da FK:** O método __init__ do Form torna o campo Foreign Key (responsavel ou funcionario) **oculto** (HiddenInput) e preenche seu initial com o PK do objeto Funcionario.
3. **Display:** Um campo *read-only* separado (funcionario_nome) é usado para exibir o nome completo do funcionário, mantendo o campo FK (funcionario) limpo para o salvamento.
4. **Fechamento Seguro:** No form_valid, a View retorna um script que executa **window.opener.location.reload(); window.close();**, garantindo o fechamento do modal e a atualização da tabela na tela principal.

4.3 Gestão e Ordenação de Listas

- **Busca Centralizada:** A função busca_centralizada utiliza o objeto **Q** para construir *QuerySets* complexos, permitindo a busca por múltiplos campos textuais usando o operador lógico **OR** (|).
- **Ordenação de Salários:** A DetailView implementa a ordenação do *QuerySet* de salários (salarios.all()) usando o método order_by('salario_liquido', 'salario_bruto') para exibir os registros do menor para o maior.