Prompts Específicos Otimizados para Claude Pro

🎯 PROMPTS PRONTOS PARA COPIAR E COLAR NO CLAUDE PRO

PROMPT 1: ANÁLISE INICIAL DO FISIOFLOW

```
CONTEXTO: Sou desenvolvedor do FisioFlow, sistema de gestão para clínicas de fisioterapia desenvolvido em Flask/Python. O sistema está em produção em: https://fisioflow-gest-o-cl-nica-600140429116.us-west1.run.app/

SITUAÇÃO ATUAL:
✅ Funcionalidades implementadas: Dashboard, Pacientes, Exercícios, Agenda, Financeiro, Relatórios
❌ Seções incompletas: Mentoria, Protocolos Clínicos, Projetos Ativos, Gestão Operacional

ARQUITETURA ATUAL:
- Backend: Flask/Python
- Banco: SQLAlchemy
- Frontend: HTML/Bootstrap/JavaScript
- Deploy: Google Cloud Run
- Estrutura: MVC padrão

OBJETIVO: Analisar a arquitetura atual e criar plano de implementação para as 4 seções faltantes.

TAREFA:
1. Avalie a arquitetura descrita
2. Sugira melhorias estruturais se necessário
3. Crie cronograma de implementação (30 dias)
4. Defina prioridades por complexidade/valor
5. Identifique possíveis desafios técnicos
6. Recomende padrões de código a seguir

ENTREGUE:
- Análise técnica da arquitetura
- Plano de implementação detalhado
- Estimativas de tempo por seção
- Recomendações de boas práticas
- Próximos passos específicos

Seja técnico e específico. Vamos começar!
```

PROMPT 2: IMPLEMENTAÇÃO - SEÇÃO MENTORIA (DASHBOARD)

```
TAREFA ESPECÍFICA: Implementar o DASHBOARD da seção "Mentoria e Ensino" do FisioFlow.

CONTEXTO: Sistema Flask/Python para gestão de clínicas de fisioterapia. A seção Mentoria existe na sidebar mas está vazia.

FUNCIONALIDADES DO DASHBOARD:
1. Métricas de progresso educacional
2. Lista de estagiários ativos
3. Casos clínicos recentes
4. Avaliações pendentes
5. Recursos educacionais mais acessados
6. Gráficos de desenvolvimento

ESPECIFICAÇÕES TÉCNICAS:
- Usar SQLAlchemy para modelos
- Seguir padrão MVC do Flask
- Templates HTML com Bootstrap
- JavaScript para interatividade
- Integrar com sistema de usuários existente

ENTREGUE NESTA ORDEM:

1. MODELOS DE BANCO (SQLAlchemy):
```python
# Modelo para Estagiários
class Intern(db.Model):
    # Campos necessários

# Modelo para Casos Clínicos Educacionais
class EducationalCase(db.Model):
    # Campos necessários

# Modelo para Avaliações de Competência
class Competency(db.Model):
    # Campos necessários
```

1. ROTA FLASK:

```python
@app.route('/mentoria')
def mentoria_dashboard():
    # Lógica completa do dashboard
    # Consultas ao banco
    # Preparação dos dados
    # Retorno do template
```

1. TEMPLATE HTML (mentoria/dashboard.html):

2. Layout responsivo com Bootstrap

3. Cards com métricas principais

4. Gráficos usando Chart.js
```

5. Tabelas com dados

6. Botões para outras funcionalidades

7. CSS/JAVASCRIPT específico se necessário

IMPORTANTE: - Código funcional e pronto para usar - Comentários explicativos - Tratamento de erros - Dados de exemplo para testes - Consistência com design existente

Comece com os modelos de banco de dados.

```
### PROMPT 3: IMPLEMENTAÇÃO - GESTÃO DE ESTAGIÁRIOS
```

CONTINUANDO A SEÇÃO MENTORIA: Implementar funcionalidade de GESTÃO DE ESTAGIÁRIOS.

CONTEXTO: Dashboard da Mentoria já implementado. Agora preciso da funcionalidade completa de gestão de estagiários.

FUNCIONALIDADES NECESSÁRIAS: 1. Cadastro de novos estagiários 2. Edição de dados dos estagiários 3. Atribuição de supervisores 4. Cronograma de atividades 5. Acompanhamento de progresso 6. Histórico de avaliações 7. Relatórios individuais

PÁGINAS NECESSÁRIAS: - /mentoria/estagiarios (lista todos) - /mentoria/estagiario/novo (cadastro) - /mentoria/estagiario/ (perfil individual) - /mentoria/estagiario//editar (edição) - /mentoria/estagiario//cronograma (atividades)

ENTREGUE:

1. ROTAS FLASK COMPLETAS:

```
@app.route('/mentoria/estagiarios')
def listar_estagiarios():
    # Lista todos os estagiários

@app.route('/mentoria/estagiario/novo', methods=['GET', 'POST'])
def novo_estagiario():
    # Cadastro de novo estagiário

@app.route('/mentoria/estagiario/<int:id>')
def perfil_estagiario(id):
    # Perfil individual

# Outras rotas necessárias...
```

1. TEMPLATES HTML:
2. estagiarios.html (lista)
3. novo_estagiario.html (formulário)
4. perfil_estagiario.html (detalhes)
5. editar_estagiario.html (edição)
6. FORMULÁRIOS:
7. Validação de dados
8. Campos obrigatórios
9. Máscaras para telefone/CPF
10. Upload de foto (opcional)
11. FUNCIONALIDADES EXTRAS:
12. Busca e filtros
13. Paginação
14. Exportação para Excel
15. Notificações automáticas

IMPORTANTE: - Integrar com modelos já criados - Manter padrão visual do FisioFlow - Incluir validações robustas - Tratamento de erros completo - Testes básicos incluídos

Vamos implementar!

```
### PROMPT 4: IMPLEMENTAÇÃO - BIBLIOTECA DE CASOS CLÍNICOS
```

CONTINUANDO SEÇÃO MENTORIA: Implementar BIBLIOTECA DE CASOS CLÍNICOS EDUCACIONAIS.

OBJETIVO: Criar sistema para armazenar, organizar e compartilhar casos clínicos para fins educacionais.

FUNCIONALIDADES: 1. Cadastro de casos clínicos (anonimizados) 2. Categorização por especialidade/patologia 3. Anexos (imagens, documentos, vídeos) 4. Sistema de tags e busca 5. Comentários e discussões 6. Avaliações de aprendizado 7. Histórico de visualizações

ESTRUTURA DE DADOS: - Dados do paciente (anonimizados) - Histórico clínico - Exames realizados - Tratamento aplicado - Evolução do caso - Materiais de apoio - Questões para discussão

PÁGINAS NECESSÁRIAS: - /mentoria/casos (biblioteca completa) - /mentoria/caso/novo (cadastro) - /mentoria/caso/ (visualização) - /mentoria/caso//discussao (comentários)

ENTREGUE:

1. MODELO DE BANCO EXPANDIDO:

```
class ClinicalCase(db.Model):
    # Campos para caso clínico completo
    # Relacionamentos com anexos
    # Metadados educacionais

class CaseAttachment(db.Model):
    # Anexos do caso (imagens, PDFs, etc.)

class CaseComment(db.Model):
    # Comentários e discussões

class CaseView(db.Model):
    # Histórico de visualizações
```

1. ROTAS FLASK:

```
@app.route('/mentoria/casos')
def biblioteca_casos():
    # Lista todos os casos com filtros

@app.route('/mentoria/caso/novo', methods=['GET', 'POST'])
def novo_caso():
    # Cadastro de novo caso

@app.route('/mentoria/caso/<int:id>')
def visualizar_caso(id):
    # Visualização completa do caso

# Outras rotas...
```

1. TEMPLATES HTML:
2. biblioteca_casos.html (lista com filtros)
3. novo_caso.html (formulário completo)
4. visualizar_caso.html (layout educacional)
5. discussao_caso.html (comentários)

6. FUNCIONALIDADES ESPECIAIS:

7. Upload de múltiplos arquivos
8. Visualizador de imagens médicas
9. Sistema de busca avançada
10. Exportação de casos para PDF
11. Sistema de favoritos

IMPORTANTE: - Anonimização completa dos dados - Conformidade com LGPD - Interface educacional intuitiva - Sistema de permissões - Backup automático de anexos

Implemente esta funcionalidade completa.

```
### PROMPT 5: IMPLEMENTAÇÃO - PROTOCOLOS CLÍNICOS
```

NOVA SEÇÃO: Implementar completamente "PROTOCOLOS CLÍNICOS" do FisioFlow.

CONTEXTO: Seção existe na sidebar com 3 itens básicos (Reabilitação Pós-Cirúrgica, Fisioterapia Neurológica, Ortopédica) mas sem funcionalidade real.

OBJETIVO: Criar biblioteca robusta de protocolos baseados em evidências científicas.

FUNCIONALIDADES PRINCIPAIS: 1. Biblioteca de protocolos por especialidade 2. Protocolos com fases detalhadas 3. Prescrição baseada em protocolos 4. Acompanhamento de aderência 5. Analytics de efetividade 6. Sistema de evidências científicas 7. Atualizações baseadas em pesquisas

ESTRUTURA DE PROTOCOLO: - Informações gerais (nome, descrição, indicações) - Critérios de inclusão/exclusão - Fases do tratamento (aguda, subaguda, crônica) - Exercícios por fase - Critérios de progressão - Resultados esperados - Evidências científicas

ENTREGUE:

1. MODELOS DE BANCO:

```
class ClinicalProtocol(db.Model):
    # Protocolo principal

class ProtocolPhase(db.Model):
    # Fases do protocolo

class ProtocolExercise(db.Model):
    # Exercícios por fase

class ProtocolEvidence(db.Model):
    # Evidências científicas

class PatientProtocol(db.Model):
    # Protocolo aplicado ao paciente
```

1. ROTAS FLASK COMPLETAS:

```
@app.route('/protocolos')
def biblioteca_protocolos():
    # Biblioteca organizada por especialidade

@app.route('/protocolo/<int:id>')
def visualizar_protocolo(id):
    # Visualização completa do protocolo

@app.route('/protocolo/prescrever/<int:patient_id>')
def prescrever_protocolo(patient_id):
    # Prescrição para paciente específico

# Outras rotas necessárias...
```

1. TEMPLATES HTML:
2. biblioteca_protocolos.html (organizada por especialidade)
3. visualizar_protocolo.html (layout científico)
4. prescrever_protocolo.html (seleção e customização)
5. acompanhar_protocolo.html (progresso do paciente)
6. PROTOCOLOS PRÉ-CADASTRADOS:
7. 5 protocolos completos por especialidade
8. Baseados em evidências reais
9. Exercícios detalhados
10. Critérios de progressão claros

INTEGRAÇÃO NECESSÁRIA: - Conectar com biblioteca de exercícios existente - Integrar com dados de pacientes - Vincular com sistema de agenda - Conectar com relatórios

de evolução

IMPORTANTE: - Base científica sólida - Interface profissional - Sistema de busca eficiente - Personalização por paciente - Métricas de efetividade

Implemente esta seção completa.

```
### PROMPT 6: IMPLEMENTAÇÃO - PROJETOS ATIVOS
```

NOVA SEÇÃO: Implementar "PROJETOS ATIVOS" do FisioFlow.

CONTEXTO: Seção existe com 2 itens básicos (Pesquisa sobre Eletroterapia, Caso Clínico: Sra. Helena) mas sem funcionalidade.

OBJETIVO: Sistema completo de gestão de projetos para clínicas de fisioterapia.

TIPOS DE PROJETOS: 1. Pesquisas clínicas 2. Casos clínicos especiais 3. Projetos de melhoria operacional 4. Capacitação da equipe 5. Desenvolvimento de protocolos 6. Parcerias acadêmicas

FUNCIONALIDADES: 1. Kanban board estilo Trello 2. Gestão de tarefas e prazos 3. Atribuição de responsáveis 4. Upload de documentos 5. Sistema de comentários 6. Timeline de progresso 7. Relatórios de status

ENTREGUE:

1. MODELOS DE BANCO:

```
class Project(db.Model):
    # Projeto principal

class ProjectTask(db.Model):
    # Tarefas do projeto

class ProjectMember(db.Model):
    # Membros da equipe

class ProjectDocument(db.Model):
    # Documentos anexados

class ProjectComment(db.Model):
    # Comentários e atualizações
```

1. ROTAS FLASK:

```
@app.route('/projetos')
def dashboard_projetos():
    # Dashboard com Kanban board

@app.route('/projeto/novo', methods=['GET', 'POST'])
def novo_projeto():
    # Criação de novo projeto

@app.route('/projeto/<int:id>')
def detalhes_projeto(id):
    # Detalhes completos do projeto

# APIs para Kanban
@app.route('/api/projeto/<int:id>/mover-tarefa', methods=['POST'])
def mover_tarefa(id):
    # Mover tarefa entre colunas
```

1. TEMPLATES HTML:
2. dashboard_projetos.html (Kanban board)
3. novo_projeto.html (formulário com templates)
4. detalhes_projeto.html (visão completa)
5. timeline_projeto.html (cronograma Gantt)
6. JAVASCRIPT AVANÇADO:
7. Kanban drag & drop
8. Atualizações em tempo real
9. Notificações push
10. Gráficos de progresso

TEMPLATES DE PROJETOS: - Template: Pesquisa Clínica - Template: Caso Clínico Especial - Template: Melhoria Operacional - Template: Capacitação - Template: Desenvolvimento de Protocolo

INTEGRAÇÃO: - Conectar com casos clínicos da Mentoria - Integrar com protocolos clínicos - Vincular com dados de pacientes - Conectar com sistema de usuários

IMPORTANTE: - Interface moderna e intuitiva - Performance otimizada - Sistema de notificações - Backup automático - Controle de versões de documentos

Implemente esta seção completa com Kanban funcional.

```
### PROMPT 7: IMPLEMENTAÇÃO - GESTÃO OPERACIONAL
```

ÚLTIMA SEÇÃO: Implementar "GESTÃO OPERACIONAL" do FisioFlow.

CONTEXTO: Seção existe apenas com "Métricas de Qualidade" vazio. Precisa ser o centro de comando da clínica.

OBJETIVO: Dashboard executivo completo com KPIs, métricas e controle operacional em tempo real.

FUNCIONALIDADES PRINCIPAIS: 1. Dashboard executivo com KPIs 2. Métricas de qualidade em tempo real 3. Indicadores de produtividade 4. Análise financeira operacional 5. Gestão de recursos e equipamentos 6. Sistema de alertas automáticos 7. Relatórios gerenciais

KPIS PRINCIPAIS: - Taxa de ocupação da agenda - Satisfação do paciente (NPS) - Produtividade por fisioterapeuta - Receita por paciente/sessão - Taxa de cancelamentos/faltas - Tempo médio de tratamento - Efetividade dos protocolos

ENTREGUE:

1. MODELOS DE BANCO:

```
class QualityIndicator(db.Model):
    # Indicadores de qualidade

class ProductivityMetric(db.Model):
    # Métricas de produtividade

class Equipment(db.Model):
    # Gestão de equipamentos

class OperationalAlert(db.Model):
    # Alertas automáticos

class ExecutiveReport(db.Model):
    # Relatórios executivos
```

1. ROTAS FLASK:

```python
@app.route('/gestao')
def dashboard_executivo():
    # Dashboard principal com todos os KPIs

@app.route('/gestao/qualidade')
def metricas_qualidade():
    # Métricas de qualidade detalhadas

@app.route('/gestao/produtividade')
def analise_produtividade():
    # Análise de produtividade da equipe

@app.route('/gestao/equipamentos')
def gestao_equipamentos():
    # Controle de equipamentos

# APIs para dados em tempo real
@app.route('/api/kpis')
def get_kpis():
    # KPIs em JSON para gráficos
```

1. TEMPLATES HTML:
2. dashboard_executivo.html (visão 360°)
3. metricas_qualidade.html (indicadores detalhados)
4. analise_produtividade.html (produtividade da equipe)
5. gestao_equipamentos.html (controle de recursos)
6. relatorios_gerenciais.html (relatórios executivos)
7. DASHBOARD AVANÇADO:
8. Gráficos em tempo real (Chart.js)
9. Cards com métricas principais
10. Alertas visuais para problemas
11. Filtros por período/profissional
12. Exportação de relatórios

FUNCIONALIDADES ESPECIAIS: - Cálculo automático de KPIs - Sistema de alertas inteligentes - Comparação com períodos anteriores - Projeções e tendências - Integração com todos os módulos

INTEGRAÇÃO TOTAL: - Dados de pacientes → Métricas de satisfação - Dados de agenda → Taxa de ocupação - Dados financeiros → Indicadores de receita - Dados de

exercícios → Efetividade de protocolos - Dados de usuários → Produtividade da equipe

IMPORTANTE: - Dados em tempo real - Performance otimizada - Interface executiva profissional - Alertas automáticos configuráveis - Relatórios exportáveis (PDF/Excel)

Implemente esta seção como centro de comando da clínica.

```
### PROMPT 8: INTEGRAÇÃO FINAL ENTRE TODOS OS MÓDULOS
```

TAREFA FINAL: Criar INTEGRAÇÃO COMPLETA entre todos os módulos do FisioFlow.

MÓDULOS IMPLEMENTADOS: ✅ Dashboard, Pacientes, Exercícios, Agenda, Financeiro, Relatórios (existentes) ✅ Mentoria e Ensino (implementado) ✅ Protocolos Clínicos (implementado) ✅ Projetos Ativos (implementado) ✅ Gestão Operacional (implementado)

OBJETIVO: Sistema totalmente integrado onde todos os módulos se conectam e compartilham dados.

INTEGRAÇÕES NECESSÁRIAS:

1. MENTORIA ↔ OUTROS MÓDULOS:
2. Casos clínicos baseados em pacientes reais (anonimizados)
3. Estagiários acompanham casos específicos
4. Avaliações baseadas em atendimentos reais
5. Protocolos usados como material educacional
6. PROTOCOLOS ↔ OUTROS MÓDULOS:
7. Prescrição automática baseada no diagnóstico
8. Exercícios organizados por protocolo
9. Agendamento baseado na fase do protocolo
10. Métricas de efetividade por protocolo
11. PROJETOS ↔ OUTROS MÓDULOS:
12. Casos especiais viram projetos de pesquisa

13. Desenvolvimento de novos protocolos
14. Análise de dados de pacientes
15. Melhoria de processos operacionais
16. GESTÃO ↔ TODOS OS MÓDULOS:
17. KPIs consolidados de todos os sistemas
18. Alertas baseados em dados de múltiplos módulos
19. Relatórios executivos unificados
20. Dashboard 360° da clínica

ENTREGUE:

1. SISTEMA DE EVENTOS:

```
class SystemEvent(db.Model):
    # Eventos para comunicação entre módulos

class ModuleIntegration(db.Model):
    # Configurações de integração

def trigger_event(event_type, data):
    # Função para disparar eventos

def handle_event(event):
    # Função para processar eventos
```

1. APIS DE INTEGRAÇÃO:

```
# APIs internas para comunicação
@app.route('/api/integration/patient-to-case')
def patient_to_educational_case():
    # Converter paciente em caso educacional

@app.route('/api/integration/protocol-suggestion')
def suggest_protocol():
    # Sugerir protocolo baseado no diagnóstico

@app.route('/api/integration/consolidated-metrics')
def get_consolidated_metrics():
    # Métricas consolidadas de todos os módulos
```

1. DASHBOARD UNIFICADO:
2. Visão 360° da clínica
3. Métricas de todos os módulos

4. Navegação contextual
5. Alertas consolidados
6. Fluxo de trabalho otimizado
7. FUNCIONALIDADES INTEGRADAS:
8. Busca global unificada
9. Notificações cruzadas
10. Relatórios consolidados
11. Workflows automatizados
12. Sincronização de dados

FLUXOS DE TRABALHO INTEGRADOS:

FLUXO 1: Novo Paciente → Protocolo → Exercícios → Acompanhamento 1. Paciente cadastrado com diagnóstico 2. Sistema sugere protocolo automaticamente 3. Exercícios são prescritos baseados no protocolo 4. Agenda é configurada conforme protocolo 5. Evolução é monitorada automaticamente 6. Métricas são atualizadas em tempo real

FLUXO 2: Caso Complexo → Projeto → Mentoria → Protocolo 1. Caso complexo identificado 2. Caso vira projeto de pesquisa 3. Projeto gera conteúdo educacional 4. Aprendizados atualizam protocolos 5. Novos protocolos são treinados 6. Ciclo de melhoria contínua

IMPORTANTE: - Performance otimizada - Transações seguras - Logs de auditoria - Rollback em caso de erro - Testes de integração completos

Implemente esta integração total que transformará o FisioFlow em sistema unificado.
```

---

# 🎯 COMO USAR ESTES PROMPTS

## 1. ORDEM DE EXECUÇÃO:

- Use os prompts na sequência (1 → 8)
```

- Complete cada seção antes de prosseguir
- Teste cada implementação

2. PERSONALIZAÇÃO:

- Substitua [valores] pelos seus dados reais
- Adapte às suas necessidades específicas
- Adicione detalhes do seu ambiente

3. ITERAÇÃO:

- Use os resultados para refinar próximos prompts
- Peça esclarecimentos quando necessário
- Documente o que funciona melhor

4. BACKUP:

- Salve o código gerado
- Faça backup antes de cada implementação
- Documente as mudanças

PRÓXIMO PASSO: Copie o Prompt 1, cole no Claude Pro, e comece a transformação do FisioFlow! 🚀