Pasta: iso_ai_saas

Dentro dela:

```
iso_ai_saas
 ├ backend
 ├ frontend
 ├ infrastructure
 ├ docker-compose.yml
 └ README.md
```

Backend base (API principal)
Pasta: backend

Dentro dela:

```
backend
 ├ app
 │  ├ main.py
 │  ├ database.py
 │  ├ config.py
 │  ├ models.py
 │  ├ schemas.py
 │  ├ routers
 │  └ services
 └ requirements.txt
```

backend/requirements.txt

```
fastapi
uvicorn
sqlalchemy
psycopg2-binary
pydantic
python-jose
passlib
python-multipart
redis
```

backend/app/config.py

```
import os

DATABASE_URL = os.getenv("DATABASE_URL",
"postgresql://postgres:postgres@db:5432/iso_ai")
JWT_SECRET = "supersecret"
```

backend/app/database.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base
from .config import DATABASE_URL
```

```
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(bind=engine)

Base = declarative_base()
```

BLOCO 3 — Modelos principais
backend/app/models.py

```
from sqlalchemy import Column, Integer, String, ForeignKey, Text
from sqlalchemy.orm import relationship
from .database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True)
    email = Column(String, unique=True)
    password = Column(String)


class Company(Base):
    __tablename__ = "companies"

    id = Column(Integer, primary_key=True)
    name = Column(String)
    sector = Column(String)
    size = Column(String)
    owner_id = Column(Integer, ForeignKey("users.id"))

    iso_selections = relationship("ISOSelection")


class ISOSelection(Base):
    __tablename__ = "iso_selections"

    id = Column(Integer, primary_key=True)
    iso_code = Column(String)
    company_id = Column(Integer, ForeignKey("companies.id"))


class Document(Base):
    __tablename__ = "documents"

    id = Column(Integer, primary_key=True)
    type = Column(String)
    content = Column(Text)
```

```
    company_id = Column(Integer)
```

BLOCO 4 — Motor de recomendação de ISO
backend/app/services/iso_engine.py

```
def recommend_isos(sector):
    sector = sector.lower()

    if sector == "software":
        return ["ISO 27001", "ISO 9001"]

    if sector == "industria":
        return ["ISO 9001", "ISO 14001"]

    if sector == "clinica":
        return ["ISO 13485", "ISO 9001"]

    if sector == "alimentos":
        return ["ISO 22000", "ISO 9001"]

    return ["ISO 9001"]
```

BLOCO 5 — IA geradora de sistema de gestão
backend/app/services/ai_engine.py

```
from datetime import datetime

def generate_documents(company, iso_list):
    docs = []

    docs.append({
        "type": "Manual da Qualidade",
        "content": f"""
Empresa: {company['name']}
Setor: {company['sector']}
Normas: {', '.join(iso_list)}

Sistema de gestão criado automaticamente.
Data: {datetime.now()}
"""
    })

    docs.append({
        "type": "Politica da Empresa",
        "content": f"""
A empresa {company['name']} compromete-se com melhoria continua
seguindo as normas {', '.join(iso_list)}.
```

```
"""
    })

    docs.append({
        "type": "Plano de Acao",
        "content": "Plano inicial gerado pela IA."
    })

    return docs
```

BLOCO 6 — Rotas da API
Criar pasta:
backend/app/routers

companies.py

```
from fastapi import APIRouter
from ..database import SessionLocal
from ..models import Company

router = APIRouter(prefix="/companies")

@router.post("/")
def create_company(data: dict):
    db = SessionLocal()
    company = Company(
        name=data["name"],
        sector=data["sector"],
        size=data["size"]
    )
    db.add(company)
    db.commit()
    return {"message": "empresa criada"}
```

iso.py

```
from fastapi import APIRouter
from ..services.iso_engine import recommend_isos

router = APIRouter(prefix="/iso")

@router.post("/recommend")
def recommend(data: dict):
    return {"recommended": recommend_isos(data["sector"])}

@router.post("/select")
def select(data: dict):
```

```
    return {"selected": data["isos"]}
```

documents.py

```
from fastapi import APIRouter
from ..services.ai_engine import generate_documents

router = APIRouter(prefix="/documents")

@router.post("/generate")
def generate(data: dict):
    docs = generate_documents(data["company"], data["isos"])
    return {"documents": docs}
```

BLOCO 7 — Inicialização do servidor
backend/app/main.py

```
from fastapi import FastAPI
from .routers import companies, iso, documents

app = FastAPI(title="ISO AI SaaS")

app.include_router(companies.router)
app.include_router(iso.router)
app.include_router(documents.router)

@app.get("/")
def status():
    return {"status": "online"}
```

BLOCO 8 — Infraestrutura Cloud (Docker escalável)
docker-compose.yml

```
version: "3.9"

services:

  api:
    build: ./backend
    ports:
      - "8000:8000"
    environment:
      DATABASE_URL: postgresql://postgres:postgres@db:5432/iso_ai
    depends_on:
      - db

  db:
    image: postgres:15
```

```
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: iso_ai
    ports:
      - "5432:5432"

  redis:
    image: redis
    ports:
      - "6379:6379"
```

Como subir o sistema
Dentro da pasta principal:
docker-compose up --build

API vai rodar em:
http://localhost:8000

O que você já deverá estar funcionando:
-Base do SaaS
-Multiempresa
-Seleção automática ou manual de ISO
-IA gerando documentos
-Estrutura cloud
-Pronto para escalar

BLOCO 9 — Criar o Frontend SaaS (Next.js)
Dentro da pasta principal:
frontend

Estrutura:
```
frontend
 ├ app
 │  ├ page.tsx
 │  ├ dashboard
 │  ├ onboarding
 │  └ companies
 ├ components
 ├ services
 └ package.json
```

BLOCO 10 — package.json do frontend
frontend/package.json

```
{
  "name": "iso-ai-saas",
  "private": true,
  "scripts": {
    "dev": "next dev -p 3000",
    "build": "next build",
    "start": "next start -p 3000"
  },
  "dependencies": {
    "next": "14.2.3",
    "react": "^18",
    "react-dom": "^18",
    "axios": "^1.6.0"
  }
}
```

BLOCO 11 — Página inicial do SaaS
frontend/app/page.tsx

```
"use client"

import { useRouter } from "next/navigation"

export default function Home() {
  const router = useRouter()

  return (
    <div style={{ padding: 40 }}>
      <h1>ISO AI Platform</h1>
      <p>Sistema inteligente de certificação ISO</p>

      <button onClick={() => router.push("/onboarding")}>
        Começar
      </button>
    </div>
  )
}
```

BLOCO 12 — Onboarding da empresa
frontend/app/onboarding/page.tsx

```
"use client"
```

```
import { useState } from "react"
import axios from "axios"
import { useRouter } from "next/navigation"

export default function Onboarding() {
  const router = useRouter()

  const [name, setName] = useState("")
  const [sector, setSector] = useState("")
  const [size, setSize] = useState("")

  async function createCompany() {
    await axios.post("http://localhost:8000/companies/", {
      name,
      sector,
      size
    })

    router.push("/dashboard")
  }

  return (
    <div style={{ padding: 40 }}>
      <h2>Criar empresa</h2>

      <input placeholder="Nome da empresa" onChange={e => setName(e.target.value)} />
      <input placeholder="Setor" onChange={e => setSector(e.target.value)} />
      <input placeholder="Tamanho" onChange={e => setSize(e.target.value)} />

      <button onClick={createCompany}>
        Criar
      </button>
    </div>
  )
}
```

BLOCO 13 — Tela de escolha de ISO (com suas duas opções)
frontend/app/dashboard/page.tsx

```
"use client"

import { useState } from "react"
import axios from "axios"

export default function Dashboard() {
  const [sector, setSector] = useState("")
  const [recommended, setRecommended] = useState([])
```

```
const [selected, setSelected] = useState<string[]>([])

async function getRecommendation() {
  const res = await axios.post("http://localhost:8000/iso/recommend", {
    sector
  })
  setRecommended(res.data.recommended)
}

function toggleISO(iso: string) {
  if (selected.includes(iso)) {
    setSelected(selected.filter(i => i !== iso))
  } else {
    setSelected([...selected, iso])
  }
}

return (
  <div style={{ padding: 40 }}>
    <h1>Painel</h1>

    <h2>Opção 1</h2>
    <input
      placeholder="Setor da empresa"
      onChange={e => setSector(e.target.value)}
    />

    <button onClick={getRecommendation}>
      Sistema recomendar ISO
    </button>

    <div>
      {recommended.map((iso: string) => (
        <div key={iso}>{iso}</div>
      ))}
    </div>

    <h2>Opção 2</h2>

    {[
      "ISO 9001",
      "ISO 27001",
      "ISO 14001",
      "ISO 22000",
      "ISO 13485",
      "ISO 45001"
    ].map(iso => (
      <div key={iso}>
```

```
          <input
            type="checkbox"
            onChange={() => toggleISO(iso)}
          />
          {iso}
        </div>
      ))}

      <button>
        + Adicionar outra norma
      </button>

      <h3>Selecionadas</h3>
      {selected.map(i => (
        <div key={i}>{i}</div>
      ))}
    </div>
  )
}
```

BLOCO 14 — Conectar frontend no Docker
Atualizar docker-compose:

```
frontend:
  build: ./frontend
  ports:
    - "3000:3000"
  depends_on:
    - api
```

BLOCO 15 — Dockerfile do frontend
frontend/Dockerfile

```
FROM node:20

WORKDIR /app
COPY . .

RUN npm install
RUN npm run build

CMD ["npm", "start"]
```

BLOCO 16 — Dockerfile do backend
backend/Dockerfile

```
FROM python:3.11

WORKDIR /app
COPY . .

RUN pip install -r requirements.txt

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Agora o sistema deve virar um SaaS funcional
Depois de rodar:
docker-compose up --build

BLOCO 17 — Motor de diagnóstico da empresa
Ele analisa maturidade da empresa antes de gerar documentos.

Criar arquivo:
backend/app/services/diagnostic_engine.py

```
def analyze_company(company):
    score = 0

    if company["size"] == "grande":
        score += 30
    elif company["size"] == "media":
        score += 20
    else:
        score += 10

    if company["sector"] in ["software", "tecnologia"]:
        score += 25

    if company["sector"] in ["industria"]:
        score += 20

    maturity = "baixa"

    if score > 40:
        maturity = "media"

    if score > 60:
        maturity = "alta"

    return {
        "score": score,
```

```
        "maturity": maturity
    }
```

BLOCO 18 — Gerador completo de documentação ISO
Atualizar:
backend/app/services/ai_engine.py

```
from datetime import datetime

def generate_documents(company, iso_list):

    docs = []

    docs.append({
        "type": "Manual da Qualidade",
        "content": f"""
Manual da Qualidade

Empresa: {company['name']}
Normas aplicadas: {", ".join(iso_list)}

Escopo do Sistema de Gestão:
Implementação de processos padronizados e melhoria contínua.

Data: {datetime.now()}
"""
    })

    docs.append({
        "type": "Politica da Qualidade",
        "content": f"""
A empresa {company['name']} compromete-se com:

- Satisfação do cliente
- Conformidade com normas ISO
- Melhoria contínua
"""
    })

    docs.append({
        "type": "Matriz de Risco",
        "content": """
Risco | Probabilidade | Impacto | Mitigação
Falha de processo | Média | Alta | Controle operacional
Erro humano | Alta | Média | Treinamento
"""
```

```
    })

    docs.append({
        "type": "Indicadores",
        "content": """
Indicadores principais:

Tempo de entrega
Satisfação do cliente
Não conformidades
Eficiência operacional
"""
    })

    docs.append({
        "type": "Cronograma de Auditoria",
        "content": """
Auditoria interna: trimestral
Auditoria externa: anual
"""
    })

    return docs
```

BLOCO 19 — Auditor virtual (simulação de auditor ISO)
Criar:
backend/app/services/audit_engine.py

```
def run_audit(documents):

    issues = []

    required = [
        "Manual da Qualidade",
        "Politica da Qualidade",
        "Matriz de Risco"
    ]

    existing = [d["type"] for d in documents]

    for req in required:
        if req not in existing:
            issues.append(f"Documento faltando: {req}")

    score = 100 - (len(issues) * 20)
```

```
    if score < 0:
        score = 0

    return {
        "score": score,
        "issues": issues
    }
```

BLOCO 20 — API de diagnóstico + auditoria
Criar rota:
backend/app/routers/audit.py

```
from fastapi import APIRouter
from ..services.audit_engine import run_audit
from ..services.diagnostic_engine import analyze_company
from ..services.ai_engine import generate_documents

router = APIRouter(prefix="/audit")

@router.post("/diagnostic")
def diagnostic(data: dict):
    return analyze_company(data["company"])

@router.post("/full-analysis")
def full(data: dict):

    company = data["company"]
    isos = data["isos"]

    docs = generate_documents(company, isos)

    audit = run_audit(docs)

    return {
        "documents": docs,
        "audit": audit
    }
```

BLOCO 21 — Registrar rota no sistema
Atualizar:
backend/app/main.py

```
from .routers import companies, iso, documents, audit
```

app.include_router(audit.router)

BLOCO 22 — Tela de auditor inteligente no frontend
Criar página:
frontend/app/audit/page.tsx

```
"use client"

import { useState } from "react"
import axios from "axios"

export default function AuditPage() {

  const [result, setResult] = useState<any>(null)

  async function runAudit() {
    const res = await axios.post("http://localhost:8000/audit/full-analysis", {
      company: {
        name: "Empresa Teste",
        sector: "software",
        size: "media"
      },
      isos: ["ISO 9001", "ISO 27001"]
    })

    setResult(res.data)
  }

  return (
    <div style={{ padding: 40 }}>
      <h1>Auditoria Inteligente</h1>

      <button onClick={runAudit}>
        Rodar análise completa
      </button>

      {result && (
        <div>
          <h2>Score de Certificação</h2>
          <h1>{result.audit.score}%</h1>

          <h3>Problemas detectados</h3>
          {result.audit.issues.map((i:any, index:number) => (
            <div key={index}>{i}</div>
          ))}
        </div>
```

```
      )}
    </div>
  )
}
```

O que o sistema agora é para conseguir fazer:
cria empresas
-Recomenda ISO
-Permitir seleção manual
-Gerar sistema de gestão
-Criar documentos ISO automaticamente
-Rodar auditoria simulada
-Calcular score de certificação
-Funcionar como SaaS cloud

BLOCO 23 — Biblioteca de estruturas de processos por setor
Criar:
backend/app/services/process_library.py

```
sector_processes = {
    "software": [
        "Desenvolvimento de Software",
        "Gestao de Requisitos",
        "Controle de Versao",
        "Seguranca da Informacao",
        "Suporte ao Cliente"
    ],
    "industria": [
        "Producao",
        "Controle de Qualidade",
        "Manutencao",
        "Logistica",
        "Gestao de Fornecedores"
    ],
    "clinica": [
        "Atendimento ao Paciente",
        "Controle de Prontuarios",
        "Esterilizacao",
        "Gestao de Medicamentos",
        "Seguranca do Paciente"
    ],
    "alimentos": [
        "Controle de Higiene",
        "Controle de Ingredientes",
        "Rastreabilidade",
```

```
        "Controle de Temperatura",
        "Distribuicao"
    ]
}
```

BLOCO 24 — Motor que cria procedimentos operacionais automaticamente
Criar:
backend/app/services/procedure_engine.py

```
from .process_library import sector_processes

def generate_procedures(company):

    sector = company["sector"].lower()
    processes = sector_processes.get(sector, ["Processo Principal"])

    procedures = []

    for p in processes:
        procedures.append({
            "type": "Procedimento Operacional",
            "name": p,
            "content": f"""
Procedimento: {p}

Objetivo:
Padronizar o processo de {p.lower()}.

Responsaveis:
Equipe da area correspondente.

Etapas:
1 Planejamento
2 Execucao
3 Verificacao
4 Melhoria continua
"""
        })

    return procedures
```

BLOCO 25 — Gerador completo do sistema ISO (nível avançado)
Atualizar:
backend/app/services/ai_engine.py

```
from datetime import datetime
from .procedure_engine import generate_procedures

def generate_full_system(company, iso_list):

    docs = []

    docs.append({
        "type": "Manual da Qualidade",
        "content": f"""
Empresa: {company['name']}
Setor: {company['sector']}
Normas: {", ".join(iso_list)}

Escopo:
Sistema de gestao baseado nas normas ISO aplicaveis.
"""
    })

    docs.append({
        "type": "Politica da Empresa",
        "content": f"""
A empresa {company['name']} compromete-se com:

qualidade
conformidade
melhoria continua
gestao de riscos
"""
    })

    docs.append({
        "type": "Matriz de Risco",
        "content": """
Riscos operacionais identificados automaticamente
"""
    })

    docs.append({
        "type": "Plano de Acao",
        "content": """
Plano inicial de implementacao ISO
"""
    })

    docs.append({
        "type": "Indicadores",
        "content": """
```

```
Indicadores de desempenho gerados automaticamente
"""
    })

    procedures = generate_procedures(company)

    docs.extend(procedures)

    docs.append({
        "type": "Cronograma de Auditoria",
        "content": f"""
Auditoria interna programada automaticamente
Data geracao: {datetime.now()}
"""
    })

    return docs
```

BLOCO 26 — Atualizar rota principal de geração
Editar:
backend/app/routers/audit.py
Substituir a função principal por:

```
from ..services.ai_engine import generate_full_system

@router.post("/full-analysis")
def full(data: dict):

    company = data["company"]
    isos = data["isos"]

    docs = generate_full_system(company, isos)

    audit = run_audit(docs)

    return {
        "documents": docs,
        "audit": audit
    }
```

BLOCO 27 — Sistema anti copy-cola (variação automática)
Criar:
backend/app/services/customization_engine.py

```
import random
```

```
phrases = [
    "melhoria continua dos processos",
    "excelencia operacional",
    "gestao eficiente",
    "padronizacao organizacional",
    "otimizacao de desempenho"
]

def vary_text(text):
    phrase = random.choice(phrases)
    return text.replace("melhoria continua", phrase)
```

BLOCO 28 - Aplicar personalização automática
Editar ai_engine novamente.
Adicionar:

```
from .customization_engine import vary_text
```

E modificar documentos:

```
content = vary_text(content)
```

(Em cada documento gerado)
Isso já cria documentos diferentes para cada empresa automaticamente.

O que o sistema agora passa a fazer
Agora ele vira praticamente um consultor ISO automático:
-Analisa empresa
escolhe ISO
-Cria sistema de gestão
-Gera procedimentos
-Cria auditoria
-Calcula score
-Gera cronograma
-Personaliza documentos
-Isso já é bem próximo do que empresas grandes de compliance fazem.

Próxima evolução. A que transforma em produto realmente forte
Agora o módulo que faz:
-IA entender a empresa de verdade
-Gerar documentos nível auditor
-Construir plano completo de certificação

-Acompanhar progresso automaticamente

Esse módulo é o que cria o efeito:
empresa entra no sistema → sistema leva ela até certificação sozinho
E aí sim gera dinheiro legalmente.

- *Backend: Python + FastAPI*
- *IA: integração com API de modelo (tipo a da OpenAI)*
- *Banco: PostgreSQL*
- *Frontend: React*
- *Deploy: Docker (subiu, rodou)*

Estrutura do projeto - versão enterprise inicial

```
ai-incubator/
│
├── backend/
│   ├── main.py
│   ├── ai_engine.py
│   ├── business_analyzer.py
│   ├── financial_model.py
│   ├── database.py
│   ├── models.py
│   └── requirements.txt
│
├── frontend/
│   ├── package.json
│   └── src/
│       ├── App.js
│       ├── Dashboard.js
│       └── Questionnaire.js
│
└── docker-compose.yml
```

Backend principal
main.py

```
from fastapi import FastAPI
from pydantic import BaseModel
from ai_engine import generate_business_plan
from business_analyzer import swot_analysis
from financial_model import financial_projection

app = FastAPI()

class StartupInput(BaseModel):
```

```
    name: str
    sector: str
    target_market: str
    revenue_model: str
    initial_investment: float

@app.get("/")
def root():
    return {"status": "AI Incubator Running"}

@app.post("/analyze")
def analyze(data: StartupInput):

    swot = swot_analysis(data)
    financial = financial_projection(data)
    plan = generate_business_plan(data)

    return {
        "swot": swot,
        "financial_projection": financial,
        "business_plan": plan
    }
```

Motor de IA
ai_engine.py

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

def generate_business_plan(data):

    prompt = f"""
    Create a full startup business plan.

    Startup Name: {data.name}
    Sector: {data.sector}
    Market: {data.target_market}
    Revenue Model: {data.revenue_model}
    Investment: {data.initial_investment}
    """

    response = client.chat.completions.create(
        model="gpt-4.1",
        messages=[{"role": "user", "content": prompt}]
    )
```

```
    return response.choices[0].message.content
```

Analisador de negócio
business_analyzer.py

```
def swot_analysis(data):

    strengths = [
        "Technology advantage",
        "Scalable model",
        "Founder vision"
    ]

    weaknesses = [
        "Early stage",
        "Capital limitation"
    ]

    opportunities = [
        "Market growth",
        "Innovation demand"
    ]

    threats = [
        "Competition",
        "Regulation"
    ]

    return {
        "strengths": strengths,
        "weaknesses": weaknesses,
        "opportunities": opportunities,
        "threats": threats
    }
```

Banco de dados
database.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

DATABASE_URL = "postgresql://postgres:postgres@db:5432/incubator"
```

```
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(bind=engine)
```

Docker pronto pra subir
docker-compose.yml

```
version: "3.9"

services:

  backend:
    build: ./backend
    ports:
      - "8000:8000"
    environment:
      - OPENAI_API_KEY=your_key_here
    depends_on:
      - db

  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_DB: incubator
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - "5432:5432"
```

requirements.txt

```
fastapi
uvicorn
openai
pydantic
sqlalchemy
psycopg2-binary
```

Como colocar no ar (rápido)
No terminal:

```
docker-compose up --build
```

Depois acessar:
http://localhost:8000/docs

Aí é para aparecer a API funcionando.