📌 PROJECT TITLE

CRM AutoPilot – Fully Automated AI Lead Generation & Client Closing Engine

---

🧠 1. What is this project? (In simple words)

This project is an **AI-powered sales automation system**.

👉 It behaves like a **24×7 virtual sales executive** that:

- Finds potential customers
- Talks to them
- Understands their replies
- Sends proposals
- Collects payment
- Starts work automatically

All **without human involvement**.

🎯 2. Why does this project exist? (Problem & Solution)

❌ Problem (Manual Sales)

- Finding clients is slow
- Writing emails takes time
- Following up is boring
- People forget replies
- Payments are delayed
- Hiring salespeople is costly

✅ Solution (Your Project)

Your system **automates the entire sales funnel** using:

- APIs
- AI (GPT)
- Automation logic
- Payment webhooks

🧩 3. What EXACTLY does your system do?

Your system performs **7 major functions**:

1️⃣ Lead Generation

Finds potential customers (leads) automatically.

Example

```
Dentist in Jaipur

Gym in Bangalore

Restaurant in Delhi
```

Stores:

- Business name
- Email
- Industry
- City

2️⃣ Lead Management

Every lead has a **status**:

```
NEW → CONTACTED → INTERESTED → PROPOSAL_SENT → PAID
```

This status decides **what the system does next**.

3️⃣ AI Personalization

Uses AI to write **human-like, personalized messages**.

Example

```
Hi Dr. Sharma,

I noticed your clinic in Jaipur...
```

Not generic spam.

4️⃣ Outreach Automation

Automatically sends:

- Emails (MVP)

- Later: LinkedIn, SMS, WhatsApp

5️⃣ Reply Understanding (AI Decision Making)

AI reads replies and understands intent:

- Interested
- Not interested
- Asking price

6️⃣ Proposal Generation

AI generates:

- Service plan
- Pricing
- Expected results

7️⃣ Payment Automation

Uses Stripe to:

- Send payment link
- Detect successful payment
- Convert lead → customer

🔄 4. Complete System Flow (Very Important)

```
Lead Found

   ↓

AI Writes Message

   ↓

Message Sent

   ↓

Reply Read by AI

   ↓

AI Generates Proposal
```

```
    ↓

Stripe Payment

    ↓

Auto Onboarding
```

🏗️ 5. Technical Architecture (How it is built)

Frontend

- Next.js 14
- TypeScript
- Tailwind CSS
- Dashboard UI

Backend

- FastAPI (Python)
- REST APIs

- Business logic

AI

- GPT-4o (or Gemini)
- Prompt engineering
- Decision logic

Database

- PostgreSQL / SQLite
- Leads
- Messages
- Payments

Payments

- Stripe Payment Links
- Webhooks

Deployment

- Frontend → Vercel
- Backend → Render / Railway

🧠 6. Core Concepts You Must Know

Concept	Meaning
Lead	Potential customer
Status	Current stage of lead
Outreach	Sending messages
Proposal	Pricing + plan
Webhook	Payment notification
Automation	No manual work

📦 7. What your MVP WILL include

✔ Lead CRUD
✔ AI email writing
✔ Email sending (or mock)
✔ Reply classification
✔ Proposal generation
✔ Stripe webhook
✔ Dashboard

❌ 8. What your MVP will NOT include (for now)

❌ Phone calls
❌ WhatsApp approval
❌ LinkedIn automation
❌ Large-scale scraping

(You'll mention these as **future improvements**.)

📁 9. Folder Structure (High Level)

```
crm-autopilot/

 ├── backend/

 │   ├── app/
 │   ├── routes/
 │   ├── models/
 ├── frontend/
 │   ├── pages/
 │   ├── components/
 ├── README.md
```

🧪 10. How this project will be evaluated (Interviews)

Interviewers will check:

- Can you explain the flow?
- Can you explain lead status?
- Can you explain AI decisions?
- Can you explain Stripe webhook?
- Can you explain trade-offs?

This project **checks all boxes**.

💼 11. How you explain this project in 2 lines (Very Important)

> "I built an AI-powered CRM system that automates lead generation, personalized outreach, reply classification, proposal generation, and Stripe-based payment workflows using FastAPI, Next.js, and GPT-4."

🚀 12. Why this project is perfect for YOU

Because:

- You already know Python, Flask/FastAPI concepts
- You're learning AI & automation
- You want high-value roles (8–15 LPA+)
- You want **real-world system design**

✅ Final One-Line Summary

> Your project is a **full-stack, AI-driven sales automation platform** that converts potential customers (leads) into paying clients without human involvement.

✅ STEP 1: LEAD STATUS FLOW (COMPLETE & FINAL)

🔹 Purpose of Status Flow

The **status flow** defines **where a lead is in the sales lifecycle**.

> Rule: **At any time, a lead must be in exactly ONE state.**

This prevents confusion, bugs, and inconsistent automation.

🔹 FINAL STATUS LIST (MVP)

We will use **exactly 5 states**:

```
NEW
CONTACTED
INTERESTED
PROPOSAL_SENT
PAID
```

No more, no less (for MVP).

🔹 STATE-BY-STATE DEFINITION (VERY IMPORTANT)

1️⃣ NEW

📌 Meaning

A lead has been **created in the system** but **no message has been sent**.

📥 How a lead enters NEW

- CSV upload
- API lead creation
- Scraping result

✅ Allowed actions

- Generate AI email
- Edit lead info
- Delete lead

❌ Not allowed

- Send proposal
- Collect payment

🔄 **Exit condition**

When **first outreach is sent**

➡ **Transition**
`NEW → CONTACTED`

2️⃣ CONTACTED

📌 **Meaning**

The lead has **received at least one outreach message**.

📥 **How a lead enters CONTACTED**

- Email sent
- LinkedIn message sent
- SMS sent

✅ **Allowed actions**

- Send follow-ups
- Monitor replies
- Classify replies using AI

❌ **Not allowed**

- Send payment link
- Mark as paid

🔄 **Exit conditions**

- Lead replies positively
- AI detects interest

➡️ **Transition**

`CONTACTED → INTERESTED`

3️⃣ INTERESTED

📌 **Meaning**

The lead has **shown buying intent**.

Examples of buying intent:

- “How much does it cost?”
- “Can you share details?”
- “Let’s talk”

📥 **How a lead enters INTERESTED**

- AI classifies reply as **Interested**
- Manual override by admin

✅ **Allowed actions**

- Generate proposal
- Send proposal
- Schedule calls (future)

❌ **Not allowed**

- Direct payment (without proposal)

🔄 **Exit condition**

Proposal is sent

➡ **Transition**

`INTERESTED → PROPOSAL_SENT`

4️⃣ PROPOSAL_SENT

📌 **Meaning**

A **formal proposal** has been sent to the lead.

Proposal can be:

- Text

- PDF
- AI-generated plan

📥 How a lead enters PROPOSAL_SENT

- Proposal message is successfully sent

✅ Allowed actions

- Send payment link
- Follow up on proposal
- Modify proposal

❌ Not allowed

- Regenerate lead
- Re-send cold email

🔄 Exit condition

Payment received

➡ Transition

`PROPOSAL_SENT → PAID`

5️⃣ PAID

📌 Meaning

The lead has **successfully paid** and is now a **client**.

📥 How a lead enters PAID

- Stripe webhook confirms payment

✅ Allowed actions

- Auto onboarding
- Task creation
- Client management

❌ Not allowed

- Outreach
- Proposal sending

🔄 Exit condition

None (terminal state)

🔷 COMPLETE STATUS FLOW DIAGRAM

```
  +-------+
   | NEW   |
   +-------+
       |
       | Outreach Sent
       v
+----------------+
| CONTACTED      |
+----------------+
       |
       | Positive Reply
       v
+----------------+
| INTERESTED     |
+----------------+
       |
       | Proposal Sent
       v
+---------------------+
| PROPOSAL_SENT       |
+---------------------+
       |
       | Payment Success
       v
    +---------+
    | PAID    |
    +---------+
```

🔹 STATUS TRANSITION RULES (CRITICAL)

These rules **must be enforced in code**.

From	To	Trigger
NEW	CONTACTED	Outreach sent
CONTACTED	INTERESTED	AI detects interest
INTERESTED	PROPOSAL_SENT	Proposal sent
PROPOSAL_SENT	PAID	Stripe webhook

❌ No skipping states
❌ No backward movement

🔹 STATUS FLOW AS CODE (ENUM)

You will implement this later like:

```
from enum import Enum

class LeadStatus(str, Enum):
    NEW = "NEW"
    CONTACTED = "CONTACTED"
    INTERESTED = "INTERESTED"
    PROPOSAL_SENT = "PROPOSAL_SENT"
    PAID = "PAID"
```

🔹 WHY THIS STEP IS EXTREMELY IMPORTANT

Because now:

- AI knows **what to do next**
- API knows **which actions are valid**
- Frontend knows **what buttons to show**
- Stripe knows **when to trigger onboarding**

This is the **backbone** of your entire system.

Day1

🟢 STEP 1: Create Project Structure (DO THIS FIRST)

Create a folder:

```
crm-autopilot/
 └── backend/
     ├── app/
     │   ├── main.py
     │   ├── database.py
     │   ├── models.py
     │   ├── schemas.py
     │   └── routes/
     │       └── leads.py
     ├── requirements.txt
```

🟢 STEP 2: Install Dependencies

Inside `backend/`:

```
pip install fastapi uvicorn sqlalchemy pydantic
python-dotenv
```

Save them:

```
# requirements.txt
fastapi
uvicorn
sqlalchemy
pydantic
Python-dotenv
email-validator==2.3.0
```

🟢 STEP 3: Database Setup (`database.py`)

We’ll start with **SQLite** (simple, perfect for MVP).

```
# app/database.py
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker,
declarative_base

DATABASE_URL = "sqlite:///./crm.db"

engine = create_engine(
    DATABASE_URL,
    connect_args={"check_same_thread": False}
)

SessionLocal = sessionmaker(
    autocommit=False,
    autoflush=False,
    bind=engine
)

Base = declarative_base()
```

🟢 STEP 4: Define Lead Status (VERY IMPORTANT)

```
# app/models.py
from sqlalchemy import Column, Integer, String,
DateTime
from sqlalchemy.sql import func
from enum import Enum
from .database import Base

class LeadStatus(str, Enum):
    NEW = "NEW"
    CONTACTED = "CONTACTED"
    INTERESTED = "INTERESTED"
    PROPOSAL_SENT = "PROPOSAL_SENT"
    PAID = "PAID"
```

🟢 STEP 5: Lead Model (Database Table)

```
# app/models.py (continue)
class Lead(Base):
    __tablename__ = "leads"
    id = Column(Integer, primary_key=True, index=True)
    business_name = Column(String, nullable=False)
    email = Column(String, nullable=False, unique=True)
    industry = Column(String, nullable=False)
    city = Column(String, nullable=False)
    status = Column(String,
default=LeadStatus.NEW.value)
    created_at = Column(DateTime(timezone=True),
server_default=func.now())
```

🟢 STEP 6: Create Schemas (Request / Response)

```
# app/schemas.py
from pydantic import BaseModel, EmailStr

class LeadCreate(BaseModel):
    business_name: str
    email: EmailStr
    industry: str
    city: str

class LeadResponse(BaseModel):
    id: int
    business_name: str
    email: str
    industry: str
    city: str
    status: str

    class Config:
        orm_mode = True
```

🟢 STEP 7: Create Lead Routes (`routes/leads.py`)

```
# app/routes/leads.py
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from ..database import SessionLocal
from ..models import Lead, LeadStatus
from ..schemas import LeadCreate, LeadResponse

router = APIRouter(prefix="/leads", tags=["Leads"])

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

@router.post("/", response_model=LeadResponse)
def create_lead(lead: LeadCreate, db: Session =
Depends(get_db)):
    existing = db.query(Lead).filter(Lead.email ==
lead.email).first()
    if existing:
        raise HTTPException(status_code=400,
detail="Lead already exists")

    new_lead = Lead(
        business_name=lead.business_name,
        email=lead.email,
        industry=lead.industry,
        city=lead.city,
        status=LeadStatus.NEW.value
    )

    db.add(new_lead)
    db.commit()
    db.refresh(new_lead)
    return new_lead

@router.get("/", response_model=list[LeadResponse])
def list_leads(db: Session = Depends(get_db)):
    return db.query(Lead).all()
```

🟢 STEP 8: Main App (`main.py`)

```
# app/main.py
from fastapi import FastAPI
from .database import Base, engine
from .routes import leads

Base.metadata.create_all(bind=engine)

app = FastAPI(title="CRM AutoPilot")

app.include_router(leads.router)

@app.get("/")
def root():
    return {"message": "CRM AutoPilot Backend Running"}
```

🟢 STEP 9: Run the Server

From `backend/` folder:

```
uvicorn app.main:app --reload
```

Open browser:

```
http://127.0.0.1:8000/docs
```

You should see **Swagger UI** 🎉

🟢 STEP 10: Test with Example (VERY IMPORTANT)

Create a Lead

```
POST /leads
```

Body:

```
{
  "business_name": "Royal Spice Restaurant",
  "email": "royalspice@gmail.com",
  "industry": "Restaurant",
  "city": "Delhi"
}
```

Response:

```
{
  "id": 1,
  "business_name": "Royal Spice Restaurant",
  "email": "royalspice@gmail.com",
  "industry": "Restaurant",
  "city": "Delhi",
  "status": "NEW"
}
```

List Leads

```
GET /leads
```