**📌 Объединённое техническое задание: Bot Oracle**

**1) Общая концепция**

Bot Oracle — это Telegram-бот с **двумя ролями**:

1. **Администратор (Manager)**
   * живой персонаж (пол/манера выбирается в анкете при старте),
   * ведёт повседневное взаимодействие:
     + выдаёт «сообщение дня» (по запросу или в проактиве),
     + отвечает на первые 5 бесплатных персональных вопросов,
     + ведёт CRM-подобное «поддержание контакта» (PING, NUDGE, DAILY PROMPT, RECOVERY и пр.),
     + эмоционально оборачивает все статусы (лимиты, подписки, ошибки).
   * основная задача — **удержание и продажи** подписки.
2. **Оракул (Oracle)**
   * включается только при подписке,
   * отвечает на персональные вопросы пользователя,
   * стиль спокойный, мудрый, серьёзный,
   * лимит **10 обращений в день**,
   * всегда по запросу пользователя (никаких проактивных сообщений).

**2) Первый вход**

* /start → анкета: возраст + пол.
* На основе ответа выбирается стиль общения.
* Администратор приветствует: бесплатное «сообщение дня» + 5 бесплатных персональных ответов.

**3) Бесплатный функционал**

* **Сообщение дня** (1 раз/день).
* **5 бесплатных персональных ответов** (счётчик, списывается).
* Все ответы/уведомления идут через Администратора в эмоциональном стиле.

**4) Подписка**

* Тарифы: день / неделя / месяц.
* Оплата — Робокасса (ResultURL → статус платежа → активация подписки).
* После активации доступен **Оракул**.
* Ограничение: 10 обращений/день.
* Админ регулярно пушит на подписку (но с лимитами частоты).

**5) Модуль Humanized Admin Engagement (CRM-поведение администратора)**

Администратор работает как **встроенный CRM-менеджер**:

**Типы задач**

* PING — «как дела?»
* NUDGE\_SUB — мягкое подталкивание к подписке
* DAILY\_MSG\_PROMPT — предложение получить сообщение дня
* DAILY\_MSG\_PUSH — редкий проактив «сегодняшнее сообщение»
* LIMIT\_INFO — уведомления об остатках бесплатных ответов или лимита Оракула
* RECOVERY — возврат после молчания
* POST\_SUB\_ONBOARD — приветствие после покупки подписки

**Логика**

* Все задачи хранятся в таблице admin\_tasks.
* Планировщик раз в день создаёт задачи (с учётом частотных лимитов и «тихих часов»).
* Воркер каждые 1–2 минуты проверяет due-задачи и отправляет.
* Время доставки выбирается рандомно внутри «человекообразных» окон (morning/day/evening).
* Реакция на входящие: ближайшие PING/NUDGE сдвигаются, добавляется «THANKS/REACT».

**6) Поведение персонажей**

* **Администратор**: эмоциональный, игривый, капризный, заботливый, ревнивый.
  + Оборачивает все системные статусы («у тебя кончились бесплатные», «лимит исчерпан», «подписка активирована»).
  + Может сам инициировать контакт (по планировщику).
* **Оракул**: мудрый, серьёзный, отвечает только на вопросы по подписке.
  + Никогда не пишет сам.
  + Всегда учитывает лимит 10/день.

**7) Архитектура данных**

* users — анкета, свободные ответы, подписка.
* daily\_messages — пул сообщений.
* daily\_sent — факт выдачи сообщения дня.
* oracle\_questions — вопросы/ответы.
* subscriptions — активные подписки.
* payments — платежи (Робокасса).
* admin\_tasks — CRM-задачи администратора.
* admin\_templates — варианты текстов (по типу, тону).
* events — общий событийный лог.

**8) Интеграции**

* **Telegram (aiogram)** — FSM для анкеты, хендлеры сообщений, persona-factory для эмоционального текста.
* **Робокасса (FastAPI)** — генерация ссылок, приём ResultURL (Password2-подпись), активация/продление подписки.
* **APScheduler** — генерация CRM-задач, рассылка due-контактов.

**9) Ограничения и антиспам**

* max\_contacts\_per\_day (по умолчанию 3) — не считая ответов на входящие.
* «Тихие часы» (22:00–08:00 по TZ пользователя).
* NUDGE\_SUB не чаще 1 раза в 48 ч и не более 2 раз в неделю.
* DAILY\_MSG\_PUSH не чаще 1/день и только если пользователь сам не запросил.

**10) Этапы разработки**

1. БД + миграции.
2. Анкета и persona-factory.
3. Ежедневные сообщения (по запросу + база daily\_sent).
4. Бесплатные ответы (счётчик 5).
5. Подписка (Робокасса + 3 тарифа).
6. Оракул (вопросы, лимит 10/день).
7. CRM-модуль админа (admin\_tasks + scheduler + dispatcher).
8. Шаблоны текстов для разных типов задач.
9. Тестирование и деплой (Docker).

**1. Общая концепция**

**Bot Oracle** — это Telegram-бот с двумя ролями:

1. **Администратор (Manager)**
   * Персонаж (пол и манера общения выбираются пользователем при старте).
   * Поддерживает контакт с пользователем в течение дня (как CRM-менеджер).
   * Отвечает на первые бесплатные вопросы.
   * Выдаёт ежедневное сообщение.
   * Эмоционально объясняет статусы (лимиты, подписки).
   * Мягко подталкивает к подписке.
2. **Оракул (Oracle)**
   * Активируется только при подписке.
   * Отвечает на персональные вопросы в спокойном и мудром стиле.
   * Ограничение — **10 вопросов в день**.
   * Никогда не инициирует общение сам.

**2. Первый вход**

* Пользователь жмёт **/start**.
* Бот задаёт вопросы анкеты:
  1. «Сколько тебе лет?»
  2. «Ты мальчик или девочка (мужчина/женщина/другое)?»
* Эти данные определяют стиль общения.
* Администратор приветствует: сообщает про ежедневное сообщение и дарит 5 бесплатных персональных ответов.

**3. Бесплатный функционал**

1. **Ежедневное сообщение**
   * Один раз в день, случайное из базы.
   * Если запрошено повторно — Администратор отвечает в игривом стиле («Эй, хитрец, второе предсказание за день? 🙄 Завтра приходи 😉»).
2. **Бесплатные персональные ответы**
   * При регистрации даётся 5 штук.
   * Каждый раз счётчик уменьшается.
   * Когда они заканчиваются, Администратор эмоционально сообщает об этом и предлагает подписку.

**4. Подписка**

* Тарифы:
  + **1 день**
  + **7 дней (неделя)**
  + **30 дней (месяц)**
* Оплата через Робокассу.
* После оплаты активируется **Оракул**.
* Лимит: 10 вопросов в день.
* Администратор периодически напоминает о подписке, но с ограничениями по частоте.

**5. Роли и поведение**

**5.1 Администратор**

* Эмоциональный, человекообразный, может быть игривым, обидчивым, заботливым, ревнивым.
* Все системные сообщения подаются через эмоции.
* Может сам инициировать контакт (по плану CRM).

**5.2 Оракул**

* Серьёзный, мудрый, отвечает только по подписке.
* Всегда по запросу, без проактивных сообщений.
* Соблюдает лимит 10 вопросов в день.

**6. Humanized Admin Engagement (CRM-модуль)**

Администратор работает как CRM-менеджер:

**6.1 Типы контактов**

* **PING** — короткое «как дела?»
* **NUDGE\_SUB** — мягкое подталкивание к подписке
* **DAILY\_MSG\_PROMPT** — предложение получить сообщение дня
* **DAILY\_MSG\_PUSH** — редкий проактив с сообщением дня
* **LIMIT\_INFO** — уведомление об остатках лимита
* **RECOVERY** — «возврат» для неактивных пользователей
* **POST\_SUB\_ONBOARD** — приветствие после покупки подписки

**6.2 Логика задач**

* Все задачи хранятся в таблице admin\_tasks.
* Планировщик раз в день создаёт задачи, учитывая частотные лимиты и «тихие часы».
* Воркер каждые 1–2 минуты проверяет due-задачи и отправляет их.
* Время выбирается случайно в пределах «человекообразных» окон (утро, день, вечер).
* Реакция на входящие сообщения: ближайшие PING/NUDGE сдвигаются, добавляется короткая эмоциональная реакция.

**6.3 Частотные ограничения**

* Не более 3 проактивных контактов в день (настраивается).
* NUDGE\_SUB — не чаще 1 раза в 48 часов и максимум 2 раза в неделю.
* Сообщения не отправляются в «тихие часы» (22:00–08:00 по локальному времени пользователя).

**7. Архитектура данных**

Основные таблицы:

* users — анкета, статус подписки, остаток бесплатных ответов.
* daily\_messages — база сообщений.
* daily\_sent — фиксация отправки сообщения дня.
* oracle\_questions — вопросы и ответы.
* subscriptions — данные о подписках.
* payments — платежи (Робокасса).
* admin\_tasks — задачи администратора.
* admin\_templates — текстовые шаблоны сообщений администратора.
* events — общий событийный лог.

**8. Интеграции**

* **Telegram (aiogram)** — хендлеры команд и сообщений, управление персонажем.
* **Робокасса (FastAPI)** — генерация ссылок, приём уведомлений ResultURL (с подписью Password2), активация подписки.
* **APScheduler** — ежедневное планирование задач и рассылка сообщений.

**9. Ограничения и антиспам**

* Максимум 3 проактивных контакта в день.
* Тихие часы: 22:00–08:00.
* Пуши подписки ограничены (не чаще 1/48 ч).
* Сообщения дня не сгружаются все сразу утром, а распределяются по дню.

**10. Этапы разработки**

1. Настройка окружения и БД.
2. Реализация анкеты и персонализации общения.
3. Ежедневные сообщения (по запросу и база daily\_sent).
4. Бесплатные ответы (счётчик, эмоциональные статусы).
5. Подписка (тарифы + Робокасса).
6. Оракул (вопросы, лимит 10/день).
7. CRM-модуль администратора (планировщик, задачи, отправка).
8. Тексты-шаблоны для всех типов сообщений.
9. Тестирование и деплой (Docker).

**11. Тональность и примеры сообщений**

**Сообщение дня (повторно):**  
«Эй, хитрец, второе предсказание за день? 🙄 Завтра приходи 😉»

**Закончились бесплатные ответы:**  
«Ну всё, я выговорилась 😅 Бесплатных больше нет. Хочешь дальше — зови Оракула 💎»

**Лимит 10 вопросов исчерпан:**  
«Сегодня ты пытливее, чем Google 😅 10 вопросов уже сжёг. Завтра продолжим!»

**Подписка активирована:**  
«Готово ✅ Теперь ты VIP. Оракул ждёт твоих вопросов. Помни: максимум 10 в день.»

**0) Технологический стек**

* **Python 3.11**
* **aiogram 3.x** (бот)
* **FastAPI** (веб-хуки Робокассы + служебные API)
* **APScheduler** (планирование crm-контактов и «сообщения дня»)
* **PostgreSQL 14+** (asyncpg/SQLAlchemy)
* **Docker + docker-compose**
* Логи: стандартный logging + таблица events

**1) Продуктовая логика (коротко)**

* **Администратор (Manager)**: человекообразный персонаж (пол и стиль выбраны в анкете). Ведёт игривую/эмоциональную коммуникацию, выдаёт «сообщение дня», отвечает на 5 бесплатных вопросов, мягко продаёт подписку, инициирует «гуманизированные» контакты в течение дня по CRM-логике.
* **Оракул (Oracle)**: доступен только по подписке; даёт персональные ответы; **лимит 10 обращений в день**; сам никогда не пишет первым.

**2) Первый вход / Анкета**

* /start → вопросы:
  1. Возраст (числом)
  2. Пол: «мужчина / женщина / другое»
* Эти параметры сохраняются и определяют:
  1. форму обращений (нейтральные при «другое»);
  2. тональность (≤25 — игривее; 26–45 — нейтрально; 46+ — уважительно).
* Приветствие администратора: упоминание ежедневного сообщения + **5 бесплатных персональных ответов**.

**3) Бесплатный функционал**

* **Сообщение дня**: 1/сутки (по запросу пользователя; иногда — проактивно по CRM-модулю в «человеческое» время).
* **5 бесплатных персональных ответов**: списываются по одному за вопрос. По исчерпании — эмоциональный нудж на подписку.

**4) Подписка (Робокасса)**

* Тарифы: **день / неделя / месяц**.
* После успешной оплаты (ResultURL) активируется/продлевается подписка.
* Открывается доступ к **Оракулу** с лимитом **10/день**.

**5) Humanized Admin Engagement (CRM-модуль)**

* Типы админ-задач:
  + PING — тёплый «как дела?»
  + DAILY\_MSG\_PROMPT — предложить получить «сегодняшнее»
  + DAILY\_MSG\_PUSH — редкий проактив с «сегодняшним» (если сам не спрашивал)
  + NUDGE\_SUB — мягко на подписку (жёсткие частотные лимиты)
  + LIMIT\_INFO — остатки бесплатных/лимита Оракула
  + RECOVERY — вернуть «пропавшего» пользователя
  + POST\_SUB\_ONBOARD — мини-онбординг после покупки
  + THANKS/REACT — короткая эмоциональная реакция на их входящее
* Планирование: по локальному TZ пользователя (из user\_prefs), в окна **утро/день/вечер** с рандомом + джиттер; соблюдение тихих часов (22:00–08:00 по умолчанию).
* Частотные лимиты: максимум **3 проактивных касания/день** (не считая ответов), NUDGE\_SUB — не чаще 1/48ч и ≤2/неделю.

**БАЗА ДАННЫХ (DDL)**

-- Пользователи

CREATE TABLE users (

id SERIAL PRIMARY KEY,

tg\_user\_id BIGINT UNIQUE NOT NULL,

username TEXT,

age INT,

gender TEXT, -- 'male','female','other'

first\_seen\_at TIMESTAMP DEFAULT now(),

last\_seen\_at TIMESTAMP,

tz TEXT DEFAULT 'UTC',

free\_answers\_left INT DEFAULT 0,

is\_blocked BOOLEAN DEFAULT FALSE,

blocked\_at TIMESTAMP

);

-- База «сообщений дня»

CREATE TABLE daily\_messages (

id SERIAL PRIMARY KEY,

text TEXT NOT NULL,

is\_active BOOLEAN DEFAULT TRUE,

weight INT DEFAULT 1

);

-- Факт выдачи «сообщения дня»

CREATE TABLE daily\_sent (

id SERIAL PRIMARY KEY,

user\_id INT REFERENCES users(id),

sent\_date DATE NOT NULL,

message\_id INT REFERENCES daily\_messages(id),

UNIQUE (user\_id, sent\_date)

);

-- Подписки

CREATE TABLE subscriptions (

id SERIAL PRIMARY KEY,

user\_id INT REFERENCES users(id),

plan\_code TEXT NOT NULL, -- 'DAY','WEEK','MONTH'

started\_at TIMESTAMP NOT NULL,

ends\_at TIMESTAMP NOT NULL,

status TEXT NOT NULL, -- 'active','expired','canceled'

robokassa\_inv\_id TEXT,

amount NUMERIC(10,2),

currency TEXT DEFAULT 'RUB'

);

-- Платежи (Робокасса)

CREATE TABLE payments (

id SERIAL PRIMARY KEY,

user\_id INT REFERENCES users(id),

inv\_id TEXT UNIQUE NOT NULL,

tariff TEXT, -- 'DAY','WEEK','MONTH'

status TEXT NOT NULL, -- 'success','fail','pending'

amount NUMERIC(10,2),

currency TEXT DEFAULT 'RUB',

paid\_at TIMESTAMP,

raw\_payload JSONB

);

-- Вопросы к Оракулу (включая бесплатные)

CREATE TABLE oracle\_questions (

id SERIAL PRIMARY KEY,

user\_id INT REFERENCES users(id),

asked\_at TIMESTAMP DEFAULT now(),

asked\_date DATE GENERATED ALWAYS AS (asked\_at::date) STORED,

source TEXT NOT NULL, -- 'FREE','SUB'

question TEXT NOT NULL,

answer TEXT,

model TEXT,

tokens\_used INT

);

-- CRM: предпочтения пользователя

CREATE TABLE user\_prefs (

user\_id INT PRIMARY KEY REFERENCES users(id),

tz TEXT DEFAULT 'UTC',

quiet\_start TIME DEFAULT '22:00',

quiet\_end TIME DEFAULT '08:00',

max\_contacts\_per\_day INT DEFAULT 3,

allow\_proactive BOOLEAN DEFAULT TRUE

);

-- CRM: частоты/окна

CREATE TABLE contact\_cadence (

user\_id INT PRIMARY KEY REFERENCES users(id),

days\_between\_pings INT DEFAULT 2,

days\_between\_nudges INT DEFAULT 2,

prefers\_windows JSONB DEFAULT '{

"morning":[9,12],

"day":[12,17],

"evening":[17,21]

}'

);

-- CRM: задачи администратора

CREATE TABLE admin\_tasks (

id BIGSERIAL PRIMARY KEY,

user\_id INT REFERENCES users(id),

type TEXT NOT NULL, -- 'PING','NUDGE\_SUB','DAILY\_MSG\_PROMPT','DAILY\_MSG\_PUSH','RECOVERY','THANKS','LIMIT\_INFO','POST\_SUB\_ONBOARD'

status TEXT NOT NULL DEFAULT 'scheduled', -- 'scheduled','due','sent','replied','snoozed','failed','canceled'

payload JSONB DEFAULT '{}'::jsonb,

scheduled\_at TIMESTAMP,

due\_at TIMESTAMP,

sent\_at TIMESTAMP,

result\_code TEXT,

created\_at TIMESTAMP DEFAULT now(),

updated\_at TIMESTAMP DEFAULT now()

);

CREATE INDEX idx\_admin\_tasks\_user\_due ON admin\_tasks(user\_id, due\_at) WHERE status IN ('scheduled','due');

CREATE INDEX idx\_admin\_tasks\_status ON admin\_tasks(status);

-- CRM: история по задачам

CREATE TABLE admin\_task\_events (

id BIGSERIAL PRIMARY KEY,

task\_id BIGINT REFERENCES admin\_tasks(id),

user\_id INT REFERENCES users(id),

event TEXT NOT NULL, -- 'created','scheduled','sent','reply\_detected','snoozed','canceled','failed'

meta JSONB DEFAULT '{}'::jsonb,

occurred\_at TIMESTAMP DEFAULT now()

);

-- Событийный лог

CREATE TABLE events (

id BIGSERIAL PRIMARY KEY,

user\_id INT REFERENCES users(id),

type TEXT NOT NULL, -- 'start','daily\_sent','free\_answer','payment\_success','sub\_started','oracle\_answer','limit\_hit','blocked'

meta JSONB DEFAULT '{}'::jsonb,

occurred\_at TIMESTAMP DEFAULT now()

);

-- Шаблоны эмоциональных сообщений администратора

CREATE TABLE admin\_templates (

id SERIAL PRIMARY KEY,

type TEXT NOT NULL, -- как admin\_tasks.type

tone TEXT NOT NULL, -- 'playful','care','pout','intrigue','tease'

text TEXT NOT NULL,

enabled BOOLEAN DEFAULT TRUE,

weight INT DEFAULT 1

);

**КОНФИГУРАЦИЯ (.env пример)**

BOT\_TOKEN=xxxxxxxx:yyyyyyyy

BASE\_URL=https://your.domain

OPENAI\_API\_KEY=sk-... # пока не используем промпты (есть заглушки)

DATABASE\_URL=postgresql+asyncpg://app:app@db:5432/app

ROBO\_LOGIN=merchant\_login

ROBO\_PASS1=merchant\_password\_1

ROBO\_PASS2=merchant\_password\_2

FREE\_QUESTIONS=5

USE\_ADMIN\_PERSONA=true

ADMIN\_IDS=123456789,987654321

HUMANIZED\_MAX\_CONTACTS\_PER\_DAY=3

NUDGE\_MIN\_HOURS=48

NUDGE\_MAX\_PER\_WEEK=2

DISPATCH\_INTERVAL\_SECONDS=60

**КОД-СКЕЛЕТЫ**

**A) Persona-обёртка (обращения/тональность администратора)**

# services/persona.py

import os

USE\_ADMIN\_PERSONA = os.getenv("USE\_ADMIN\_PERSONA","true").lower() == "true"

def build\_address(age: int|None, gender: str|None) -> str:

if gender == "female":

return "солнышко" if (age and age <= 25) else "дорогая"

if gender == "male":

return "дружище" if (age and age <= 25) else "уважаемый"

return "друг"

def persona\_factory(user\_row: dict):

addr = build\_address(user\_row.get("age"), user\_row.get("gender"))

def wrap(text: str, ctx: str|None=None) -> str:

if not USE\_ADMIN\_PERSONA:

return text

return f"{addr}, {text}"

return wrap

**B) Пользователь и бесплатные ответы**

# services/users.py

import os

FREE\_QUESTIONS = int(os.getenv("FREE\_QUESTIONS", 5))

async def get\_or\_create\_user(db, tg\_user):

user = await db.fetchrow("SELECT \* FROM users WHERE tg\_user\_id=:id", id=tg\_user.id)

if user:

await db.execute("UPDATE users SET last\_seen\_at=now() WHERE id=:uid", uid=user["id"])

return user

await db.execute("""

INSERT INTO users (tg\_user\_id, username, first\_seen\_at, free\_answers\_left)

VALUES (:id, :username, now(), :fq)

""", id=tg\_user.id, username=tg\_user.username, fq=FREE\_QUESTIONS)

return await db.fetchrow("SELECT \* FROM users WHERE tg\_user\_id=:id", id=tg\_user.id)

**C) Сообщение дня**

# services/daily.py

import random

from datetime import date

async def has\_daily\_today(db, user\_id: int) -> bool:

return bool(await db.fetchrow(

"SELECT 1 FROM daily\_sent WHERE user\_id=:u AND sent\_date=:d",

u=user\_id, d=date.today()

))

async def pick\_daily\_message(db) -> dict:

rows = await db.fetch("SELECT id, text, weight FROM daily\_messages WHERE is\_active")

population = [r for r in rows for \_ in range(r["weight"])]

return random.choice(population) if population else None

**D) Проверка подписки и лимита Оракула**

# services/subscriptions.py

from datetime import date

async def is\_sub\_active(db, user\_id: int) -> bool:

return bool(await db.fetchrow("""

SELECT 1 FROM subscriptions

WHERE user\_id=:u AND status='active' AND ends\_at > now()

LIMIT 1

""", u=user\_id))

async def oracle\_used\_today(db, user\_id: int) -> int:

row = await db.fetchrow("""

SELECT COUNT(\*) c FROM oracle\_questions

WHERE user\_id=:u AND asked\_date=:d AND source='SUB'

""", u=user\_id, d=date.today())

return row["c"] if row else 0

**E) Хендлеры бота (aiogram 3.x) — основные**

# bot/handlers.py

from aiogram import Router, types, F

from services.users import get\_or\_create\_user

from services.persona import persona\_factory

from services.daily import has\_daily\_today, pick\_daily\_message

from services.subscriptions import is\_sub\_active, oracle\_used\_today

from datetime import date

router = Router()

@router.message(F.text == "/start")

async def start(m: types.Message, db):

u = await get\_or\_create\_user(db, m.from\_user)

# тут FSM: спросить возраст → сохранить; спросить пол → сохранить

persona = persona\_factory(u)

await m.answer(persona(

"я твой админ 🤗 у нас есть бесплатное сообщение каждый день и 5 персональных ответов в подарок."

))

@router.message(F.text == "Сообщение дня")

async def daily\_msg(m: types.Message, db):

u = await db.fetchrow("SELECT \* FROM users WHERE tg\_user\_id=:id", id=m.from\_user.id)

persona = persona\_factory(u)

if await has\_daily\_today(db, u["id"]):

await m.answer(persona("хитрец, второе предсказание за день? 🙄 не-а. завтра приходи 😉"))

return

msg = await pick\_daily\_message(db)

if not msg:

await m.answer(persona("сегодня без новостей, но я слежу 😌"))

return

await db.execute("""

INSERT INTO daily\_sent (user\_id, sent\_date, message\_id) VALUES (:u, :d, :m)

""", u=u["id"], d=date.today(), m=msg["id"])

await m.answer(persona(f"сегодняшнее сообщение: {msg['text']}"))

@router.message() # любые тексты — как вопросы

async def any\_text(m: types.Message, db):

u = await db.fetchrow("SELECT \* FROM users WHERE tg\_user\_id=:id", id=m.from\_user.id)

persona = persona\_factory(u)

if await is\_sub\_active(db, u["id"]):

used = await oracle\_used\_today(db, u["id"])

if used >= 10:

await m.answer(persona("сегодня ты пытливее, чем Google 😅 лимит 10 исчерпан, завтра продолжим 🔮"))

return

answer = await call\_oracle\_stub(m.text) # заглушка

await db.execute("""

INSERT INTO oracle\_questions (user\_id, source, question, answer)

VALUES (:u, 'SUB', :q, :a)

""", u=u["id"], q=m.text, a=answer)

await m.answer(answer)

return

# нет подписки — используем бесплатные

if u["free\_answers\_left"] > 0:

await db.execute("UPDATE users SET free\_answers\_left = free\_answers\_left - 1 WHERE id=:u", u=u["id"])

answer = await call\_admin\_stub(m.text) # заглушка

left = u["free\_answers\_left"] - 1

await db.execute("""

INSERT INTO oracle\_questions (user\_id, source, question, answer)

VALUES (:u, 'FREE', :q, :a)

""", u=u["id"], q=m.text, a=answer)

await m.answer(persona(f"{answer}\n\nосталось бесплатных ответов: {left}", ctx="free\_answer"))

else:

await m.answer(persona("бесплатные закончились 😅 хочешь дальше — зови Оракула 💎", ctx="free\_empty"))

*(функции call\_admin\_stub/call\_oracle\_stub ниже, раздел «Заглушки»)*

**F) Робокасса: формирование ссылки и ResultURL**

# services/payments.py

import hashlib, os

ROBO\_LOGIN = os.getenv("ROBO\_LOGIN")

ROBO\_PASS1 = os.getenv("ROBO\_PASS1")

PLAN\_PRICES = {"DAY": 99.00, "WEEK": 299.00, "MONTH": 899.00}

def build\_robo\_link(inv\_id: str, plan\_code: str, base\_url: str) -> str:

out\_sum = f"{PLAN\_PRICES[plan\_code]:.2f}"

desc = f"Bot Oracle {plan\_code}"

sig = hashlib.md5(f"{ROBO\_LOGIN}:{out\_sum}:{inv\_id}:{ROBO\_PASS1}".encode()).hexdigest()

return (

"https://auth.robokassa.ru/Merchant/Index.aspx"

f"?MerchantLogin={ROBO\_LOGIN}&OutSum={out\_sum}&InvId={inv\_id}"

f"&Description={desc}&SignatureValue={sig}"

f"&SuccessURL={base\_url}/robokassa/success"

f"&Culture=ru"

)

# api/robokassa.py

from fastapi import APIRouter, Request, HTTPException

import hashlib, os

from datetime import timedelta

router = APIRouter()

ROBO\_PASS2 = os.getenv("ROBO\_PASS2")

PLAN\_DUR = {"DAY": 1, "WEEK": 7, "MONTH": 30}

def ok\_signature(out\_sum, inv\_id, sig):

must = hashlib.md5(f"{out\_sum}:{inv\_id}:{ROBO\_PASS2}".encode()).hexdigest().lower()

return (sig or "").lower() == must

@router.post("/robokassa/result")

async def robokassa\_result(request: Request, db):

form = dict(await request.form())

out\_sum = form.get("OutSum")

inv\_id = form.get("InvId")

sig = form.get("SignatureValue")

if not (out\_sum and inv\_id and sig) or not ok\_signature(out\_sum, inv\_id, sig):

raise HTTPException(400, "bad signature")

pay = await db.fetchrow("SELECT \* FROM payments WHERE inv\_id=:inv", inv=inv\_id)

if not pay:

raise HTTPException(404, "payment not found")

if pay["status"] == "success":

return "OK"

await db.execute("UPDATE payments SET status='success', paid\_at=now() WHERE inv\_id=:inv", inv=inv\_id)

# активация/продление подписки

plan = pay["tariff"]

now = (await db.fetchrow("SELECT now() n"))["n"]

sub = await db.fetchrow("""

SELECT \* FROM subscriptions

WHERE user\_id=:u AND status='active' AND ends\_at > now()

ORDER BY ends\_at DESC LIMIT 1

""", u=pay["user\_id"])

if sub:

new\_ends = sub["ends\_at"] + timedelta(days=PLAN\_DUR[plan])

await db.execute("UPDATE subscriptions SET ends\_at=:e WHERE id=:sid", e=new\_ends, sid=sub["id"])

else:

await db.execute("""

INSERT INTO subscriptions (user\_id, plan\_code, started\_at, ends\_at, status, robokassa\_inv\_id, amount, currency)

VALUES (:u, :p, :s, :e, 'active', :inv, :amt, 'RUB')

""", u=pay["user\_id"], p=plan, s=now, e=now+timedelta(days=PLAN\_DUR[plan]),

inv=inv\_id, amt=pay["amount"])

# CRM: добавить POST\_SUB\_ONBOARD

await db.execute("""

INSERT INTO admin\_tasks (user\_id, type, status, scheduled\_at, due\_at)

VALUES (:u, 'POST\_SUB\_ONBOARD', 'scheduled', now(), now() + interval '10 minutes')

""", u=pay["user\_id"])

return "OK"

**G) Покупка (клик по тарифу)**

# bot/handlers\_subscribe.py

from aiogram import Router, types

import uuid, os

from services.payments import PLAN\_PRICES, build\_robo\_link

router = Router()

BASE\_URL = os.getenv("BASE\_URL")

@router.callback\_query(lambda c: c.data in ("BUY\_DAY","BUY\_WEEK","BUY\_MONTH"))

async def buy(c: types.CallbackQuery, db):

plan = c.data.replace("BUY\_","") # DAY/WEEK/MONTH

user = await db.fetchrow("SELECT id FROM users WHERE tg\_user\_id=:id", id=c.from\_user.id)

inv\_id = str(uuid.uuid4())

price = PLAN\_PRICES[plan]

await db.execute("""

INSERT INTO payments (user\_id, inv\_id, tariff, status, amount, currency)

VALUES (:u, :inv, :tar, 'pending', :amt, 'RUB')

""", u=user["id"], inv=inv\_id, tar=plan, amt=price)

url = build\_robo\_link(inv\_id, plan, BASE\_URL)

await c.message.answer(f"Готово. Перейди к оплате:\n{url}")

**H) CRM-планировщик (ежедневная генерация задач)**

# crm/daily\_planner.py

from datetime import datetime, time

import random

async def plan\_for\_user(db, user):

prefs = await db.fetchrow("SELECT \* FROM user\_prefs WHERE user\_id=:u", u=user["id"])

cadence = await db.fetchrow("SELECT \* FROM contact\_cadence WHERE user\_id=:u", u=user["id"])

if not prefs:

await db.execute("INSERT INTO user\_prefs (user\_id) VALUES (:u)", u=user["id"])

prefs = await db.fetchrow("SELECT \* FROM user\_prefs WHERE user\_id=:u", u=user["id"])

if not cadence:

await db.execute("INSERT INTO contact\_cadence (user\_id) VALUES (:u)", u=user["id"])

cadence = await db.fetchrow("SELECT \* FROM contact\_cadence WHERE user\_id=:u", u=user["id"])

# Сколько уже было «sent/replied» сегодня (по TZ) — опускаем детали TZ для краткости

sent\_today = await db.fetchval("""

SELECT count(\*) FROM admin\_tasks

WHERE user\_id=:u AND status IN ('sent','replied')

AND sent\_at::date = current\_date

""", u=user["id"])

remaining = max(0, (prefs["max\_contacts\_per\_day"] or 3) - sent\_today)

if remaining == 0 or not prefs["allow\_proactive"]:

return

need\_prompt = not await db.fetchrow("""

SELECT 1 FROM daily\_sent WHERE user\_id=:u AND sent\_date=current\_date

""", u=user["id"])

candidates = []

if need\_prompt:

candidates.append("DAILY\_MSG\_PROMPT")

candidates.append("PING")

last\_nudge = await db.fetchrow("""

SELECT max(sent\_at) ts FROM admin\_tasks

WHERE user\_id=:u AND type='NUDGE\_SUB' AND status IN ('sent','replied')

""", u=user["id"])

ok\_for\_nudge = not last\_nudge or (datetime.utcnow() - (last\_nudge["ts"] or datetime(1970,1,1))).days >= (cadence["days\_between\_nudges"] or 2)

if ok\_for\_nudge:

candidates.append("NUDGE\_SUB")

windows = (cadence["prefers\_windows"]

or {"morning":[9,12],"day":[12,17],"evening":[17,21]})

for t in random.sample(candidates, k=min(remaining, len(candidates))):

win = random.choices(list(windows.keys()), weights=[0.3,0.3,0.4])[0]

h1, h2 = windows[win]

hour = random.randint(h1, h2-1)

minute = random.randint(0,59)

due = datetime.utcnow().replace(hour=hour, minute=minute, second=0, microsecond=0)

quiet\_start = prefs["quiet\_start"] or time(22,0)

quiet\_end = prefs["quiet\_end"] or time(8,0)

# если попали в «тихие часы», сдвинем на 09:15 как пример

if hour >= quiet\_start.hour or hour < quiet\_end.hour:

due = due.replace(hour=9, minute=15)

await db.execute("""

INSERT INTO admin\_tasks (user\_id, type, status, scheduled\_at, due\_at)

VALUES (:u, :t, 'scheduled', now(), :due)

""", u=user["id"], t=t, due=due)

**I) CRM-диспетчер (рассылка due-задач)**

# crm/dispatcher.py

async def pick\_template\_text(db, task\_type: str) -> str:

rows = await db.fetch("SELECT text, weight FROM admin\_templates WHERE type=:t AND enabled", t=task\_type)

if not rows:

return "[эмоциональный текст по умолчанию]"

bag = [r["text"] for r in rows for \_ in range(r["weight"] or 1)]

import random

return random.choice(bag)

async def dispatch\_due\_tasks(db, bot):

now = await db.fetchval("SELECT now()")

tasks = await db.fetch("""

SELECT t.\*, u.tg\_user\_id, u.age, u.gender

FROM admin\_tasks t

JOIN users u ON u.id=t.user\_id

WHERE t.status IN ('scheduled','due') AND t.due\_at <= :n

ORDER BY t.due\_at

LIMIT 200

""", n=now)

for t in tasks:

persona = persona\_factory(dict(age=t["age"], gender=t["gender"]))

text = await pick\_template\_text(db, t["type"])

try:

await bot.send\_message(t["tg\_user\_id"], persona(text))

await db.execute("UPDATE admin\_tasks SET status='sent', sent\_at=now(), updated\_at=now() WHERE id=:id", id=t["id"])

await db.execute("INSERT INTO admin\_task\_events (task\_id, user\_id, event) VALUES (:tid,:uid,'sent')",

tid=t["id"], uid=t["user\_id"])

except Exception:

await db.execute("UPDATE users SET is\_blocked=TRUE, blocked\_at=now() WHERE id=:u", u=t["user\_id"])

await db.execute("UPDATE admin\_tasks SET status='failed', result\_code='error', updated\_at=now() WHERE id=:id", id=t["id"])

**J) Планировщик (APScheduler) — запуск**

# scheduler.py

from apscheduler.schedulers.asyncio import AsyncIOScheduler

from crm.daily\_planner import plan\_for\_user

from crm.dispatcher import dispatch\_due\_tasks

def start\_scheduler(loop, db, bot):

sch = AsyncIOScheduler(timezone="UTC")

sch.add\_job(lambda: plan\_all\_users(db), trigger="cron", hour=6, minute=5) # генерация утренняя

sch.add\_job(lambda: dispatch\_due\_tasks(db, bot), trigger="interval", seconds=60)

sch.start()

async def plan\_all\_users(db):

users = await db.fetch("SELECT id FROM users WHERE is\_blocked=FALSE")

for u in users:

try:

await plan\_for\_user(db, u)

except Exception:

pass

**K) Заглушки ответов (ИИ пока не подключаем)**

# services/stubs.py

async def call\_admin\_stub(text: str) -> str:

return f"я услышал тебя и вот мой короткий ответ: {text[:80]}…"

async def call\_oracle\_stub(text: str) -> str:

return f"мой персональный ответ для тебя: {text[:120]}… (подробности появятся позже)"

**UI-СООБЩЕНИЯ (ключевые состояния; всегда через Администратора)**

* **Сообщение дня (повторно в тот же день)**  
  «Эй, хитрец, второе предсказание за день? 🙄 Не-а. Завтра приходи 😉»
* **После бесплатного ответа**  
  «Лови мой ответ 🔮 Осталось {N} — думай, на что их потратить 😉»
* **Бесплатные закончились**  
  «Ну всё, я выговорилась 😅 Бесплатных больше нет. Хочешь дальше — зови Оракула 💎»
* **Меню подписки**  
  «Хочешь по-взрослому? 1️⃣ День 2️⃣ Неделя 3️⃣ Месяц — выбирай, я подключу тебе Оракула 🔮»
* **Подписка активирована**  
  «Готово ✅ Теперь ты VIP. Оракул ждёт твоих вопросов. Помни: максимум 10 в день.»
* **Лимит Оракула исчерпан**  
  «Сегодня ты пытливее, чем Google 😅 10 вопросов уже сжёг. Завтра продолжим!»

**Ограничения и этика**

* «Тихие часы» (по умолчанию 22:00–08:00 локально) — не пишем проактивно.
* max\_contacts\_per\_day=3 — не считаем ответы на входящие.
* NUDGE\_SUB — не чаще 1/48ч и ≤2/неделю.
* Оракул никогда не инициирует диалог сам.
* Все системные статусы сообщаем «по-людски» (через персону).

**Этапы внедрения**

1. Миграции БД (Alembic)
2. Анкета и persona-factory
3. «Сообщение дня» (по кнопке + запись daily\_sent)
4. 5 бесплатных ответов (списание, тексты)
5. Меню тарифов + создание платежа (Робокасса)
6. ResultURL → активация/продление подписки
7. Оракул: учёт лимита 10/день и запись oracle\_questions
8. CRM-модуль: генерация задач, рассылка due-сообщений, шаблоны текстов
9. Защита: тихие часы, антиспам, обработка блокировок
10. Тестирование и деплой (Docker)

**1) Пакет текстовых шаблонов для admin\_templates**

Вставьте как начальный сид (можно одним SQL-файлом при деплое).  
tone: playful, care, pout, intrigue, tease (можно добавлять свои).

-- PING (тёплые касания)

INSERT INTO admin\_templates(type, tone, text, weight) VALUES

('PING','care','как ты там? я тут немного переживаю 🙃',2),

('PING','playful','эй, я рядом. что новенького у тебя сегодня? ✨',2),

('PING','pout','пропадаешь опять? а я жду... 🙄',1),

('PING','intrigue','у меня ощущение, что сегодня будет поворот. расскажешь потом? 😉',1);

-- DAILY\_MSG\_PROMPT (предложение получить «сегодняшнее»)

INSERT INTO admin\_templates(type, tone, text, weight) VALUES

('DAILY\_MSG\_PROMPT','playful','хочешь, открою твоё сегодняшнее сообщение? только не подглядывай заранее 😉',2),

('DAILY\_MSG\_PROMPT','intrigue','кажется, вселенная сегодня шепчет кое-что интересное. спросишь?',2),

('DAILY\_MSG\_PROMPT','tease','могу сказать «сегодняшнее», но… может, чуть позже? 😏',1);

-- DAILY\_MSG\_PUSH (редкий проактив с текстом дня; {TEXT} подставляется программно)

INSERT INTO admin\_templates(type, tone, text, weight) VALUES

('DAILY\_MSG\_PUSH','care','сегодняшнее сообщение как раз вовремя: {TEXT}',2),

('DAILY\_MSG\_PUSH','intrigue','подоспело «сегодня»: {TEXT}',1);

-- NUDGE\_SUB (мягкая продажа подписки)

INSERT INTO admin\_templates(type, tone, text, weight) VALUES

('NUDGE\_SUB','playful','правда, до сих пор без подписки? 🙄 давай уже по-взрослому — я подключу тебе всё 💎',2),

('NUDGE\_SUB','intrigue','не всем я такое предлагаю… но тебе стоит взять подписку. дальше интереснее 😉',1),

('NUDGE\_SUB','tease','ладно-ладно, не уговариваю. просто с подпиской я буду щедрее 😌',1);

-- LIMIT\_INFO (остатки бесплатных или лимита оракула; {N} / {LEFT} подставляются)

INSERT INTO admin\_templates(type, tone, text, weight) VALUES

('LIMIT\_INFO','playful','остался всего {N} выстрел 🎯 думай, на что его потратить!',2),

('LIMIT\_INFO','care','у тебя ещё {LEFT} на сегодня. используй с умом 💖',1);

-- POST\_SUB\_ONBOARD (после покупки подписки)

INSERT INTO admin\_templates(type, tone, text, weight) VALUES

('POST\_SUB\_ONBOARD','care','готово ✅ теперь ты VIP. оракул ждёт. помни: максимум 10 в день.',2),

('POST\_SUB\_ONBOARD','playful','ну наконец-то 😌 теперь будет интересно. спрашивай!',1);

-- RECOVERY (возвращаем неактивного)

INSERT INTO admin\_templates(type, tone, text, weight) VALUES

('RECOVERY','care','я скучала 🙂 давай продолжим?',2),

('RECOVERY','playful','ой, пропадал(а)! не теряй меня так надолго 😉',1);

-- THANKS/REACT (реакция на входящее)

INSERT INTO admin\_templates(type, tone, text, weight) VALUES

('THANKS','care','вижу тебя 🌟 спасибо, что написал(а)!',2),

('THANKS','playful','слышу-слышу! продолжай, мне это нравится 😉',1),

('REACT','pout','вот теперь мне лучше 😌 продолжай не пропадать.',1);

**2) Сиды для «сообщений дня» (daily\_messages)**

INSERT INTO daily\_messages(text, is\_active, weight) VALUES

('будь внимательнее к мелочам — они сегодня важнее громких жестов', true, 2),

('не спеши: правильный шаг сам попросится вперёд', true, 1),

('сегодняшняя удача любит аккуратность и спокойствие', true, 1),

('прислушайся к человеку, который редко говорит — его слова ценны', true, 1),

('не забывай про себя: баланс важнее побед', true, 1);

**3) docker-compose.yml (пример)**

version: "3.8"

services:

db:

image: postgres:14

environment:

POSTGRES\_DB: app

POSTGRES\_USER: app

POSTGRES\_PASSWORD: app

volumes:

- dbdata:/var/lib/postgresql/data

ports:

- "5432:5432"

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U app -d app"]

interval: 5s

timeout: 3s

retries: 10

app:

build: .

depends\_on:

db:

condition: service\_healthy

env\_file: .env

command: ["python","-m","app.main"]

restart: unless-stopped

volumes:

dbdata:

app.main — ваш модуль запуска бота + FastAPI (обычно через uvicorn или внутри asyncio-лупа с aiogram).

**4) Alembic (базовая миграция)**

**4.1 Инициализация Alembic**

alembic init alembic

В alembic.ini укажите строку подключения (или используйте env vars). В alembic/env.py подтяните DATABASE\_URL из окружения.

**4.2 Первая ревизия (пример alembic/versions/0001\_init.py)**

from alembic import op

import sqlalchemy as sa

revision = "0001\_init"

down\_revision = None

branch\_labels = None

depends\_on = None

def upgrade():

op.execute("""

-- сюда вставьте весь DDL из раздела БД ТЗ (users, daily\_messages, daily\_sent, subscriptions, payments,

-- oracle\_questions, user\_prefs, contact\_cadence, admin\_tasks, admin\_task\_events, events, admin\_templates)

""")

def downgrade():

op.execute("""

DROP TABLE IF EXISTS admin\_templates;

DROP TABLE IF EXISTS events;

DROP TABLE IF EXISTS admin\_task\_events;

DROP TABLE IF EXISTS admin\_tasks;

DROP TABLE IF EXISTS contact\_cadence;

DROP TABLE IF EXISTS user\_prefs;

DROP TABLE IF EXISTS oracle\_questions;

DROP TABLE IF EXISTS payments;

DROP TABLE IF EXISTS subscriptions;

DROP TABLE IF EXISTS daily\_sent;

DROP TABLE IF EXISTS daily\_messages;

DROP TABLE IF EXISTS users;

""")

Практически: лучше разбить на отдельные op.create\_table(...), но если быстрее — можно выполнить одним op.execute("""...DDL...""").

**5) Утилитные SQL-скрипты**

**5.1 Создать базовые user\_prefs и contact\_cadence для всех существующих пользователей**

INSERT INTO user\_prefs (user\_id)

SELECT id FROM users u

WHERE NOT EXISTS (SELECT 1 FROM user\_prefs p WHERE p.user\_id=u.id);

INSERT INTO contact\_cadence (user\_id)

SELECT id FROM users u

WHERE NOT EXISTS (SELECT 1 FROM contact\_cadence c WHERE c.user\_id=u.id);

**5.2 Ограничение на «NUDGE\_SUB» (контроль частоты) — пример SQL-предиката в выборке задач**

-- отбираем пользователей, кому можно поставить NUDGE\_SUB

SELECT u.id

FROM users u

LEFT JOIN (

SELECT user\_id, MAX(sent\_at) AS last\_nudge

FROM admin\_tasks

WHERE type='NUDGE\_SUB' AND status IN ('sent','replied')

GROUP BY user\_id

) n ON n.user\_id = u.id

WHERE (n.last\_nudge IS NULL OR n.last\_nudge <= now() - interval '48 hours');

**6) Пример «склеенного» запуска (бот + веб в одном приложении)**

# app/main.py

import asyncio

import os

from aiogram import Bot, Dispatcher

from aiogram.fsm.storage.memory import MemoryStorage

from fastapi import FastAPI

import uvicorn

from scheduler import start\_scheduler

from api.robokassa import router as robo\_router

# вам нужен общий объект db (например, asyncpg pool)

from app.db import get\_db\_pool # реализуйте

BOT\_TOKEN = os.getenv("BOT\_TOKEN")

app = FastAPI(title="Bot Oracle API")

app.include\_router(robo\_router)

async def run\_bot():

dp = Dispatcher(storage=MemoryStorage())

bot = Bot(BOT\_TOKEN, parse\_mode="HTML")

# зарегистрируйте роутеры/хендлеры

from bot.handlers import router as core\_router

from bot.handlers\_subscribe import router as sub\_router

dp.include\_router(core\_router)

dp.include\_router(sub\_router)

db = await get\_db\_pool()

# старт планировщика (crm + due-диспетчер)

start\_scheduler(asyncio.get\_event\_loop(), db, bot)

await dp.start\_polling(bot, db=db)

async def run\_api():

config = uvicorn.Config(app, host="0.0.0.0", port=8000, log\_level="info")

server = uvicorn.Server(config)

await server.serve()

async def main():

await asyncio.gather(run\_bot(), run\_api())

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.run(main())

**7) Чек-лист перед продом**

* Все миграции применены (alembic upgrade head).
* Сиды daily\_messages и admin\_templates загружены.
* .env заполнен (Bot Token, Robo, DB, BASE\_URL).
* ResultURL у Робокассы настроен на https://<BASE\_URL>/robokassa/result.
* SuccessURL настроен (можно статическую заглушку).
* Проверены: бесплатные ответы (5 → 0), сообщение дня (повторный запрос), лимит Оракула 10/день, покупка/продление подписки.
* CRM: создаются задачи, соблюдаются «тихие часы», не более 3 касаний/день, NUDGE\_SUB ≤ 2/неделю.
* Блокировки (403) помечают users.is\_blocked=true.
* Логи и events пишутся.