RUDN university

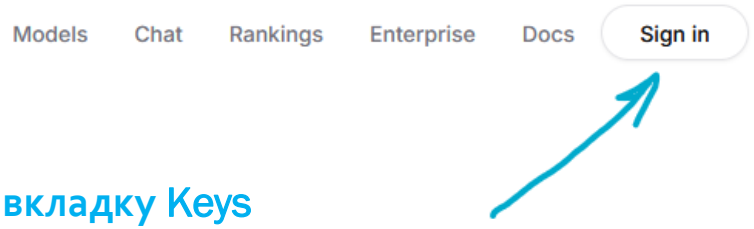
ОСНОВЫ **HTTP**

OBI MODEL, TCP/IP, CLIENT-SERVER ARCHITECTURE, HTTP PROTOCOL, HTTP STREAM, HTTP METHODS, HTTP CODES

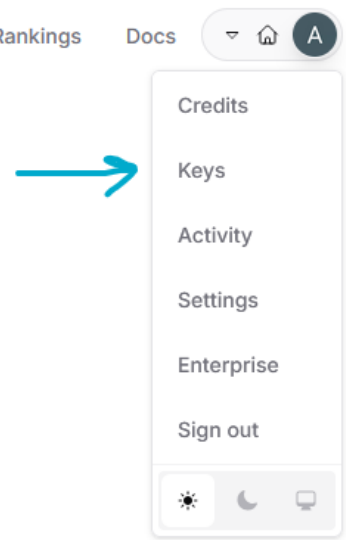
OPENROUTER АВТОРИЗАЦИЯ И ПОЛУЧЕНИЕ ТОКЕНА

https://openrouter.ai

1 Зарегистрироваться

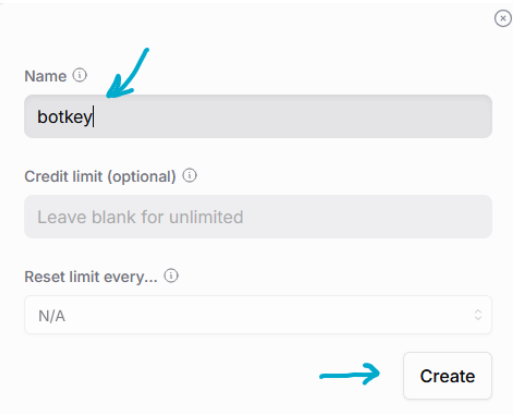


2 Перейти на вкладку Keys



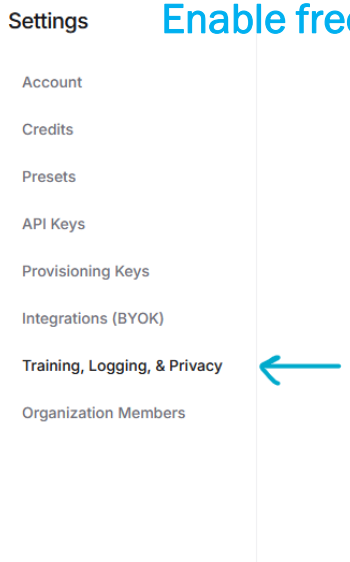
3 Создать ключ API

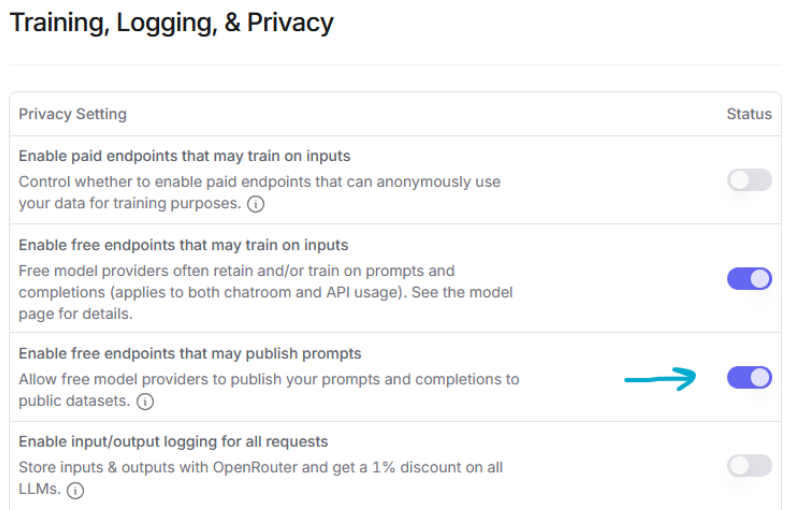
4 Ввести имя ключа и нажать Create



! После создания сохранить себе значение ключа

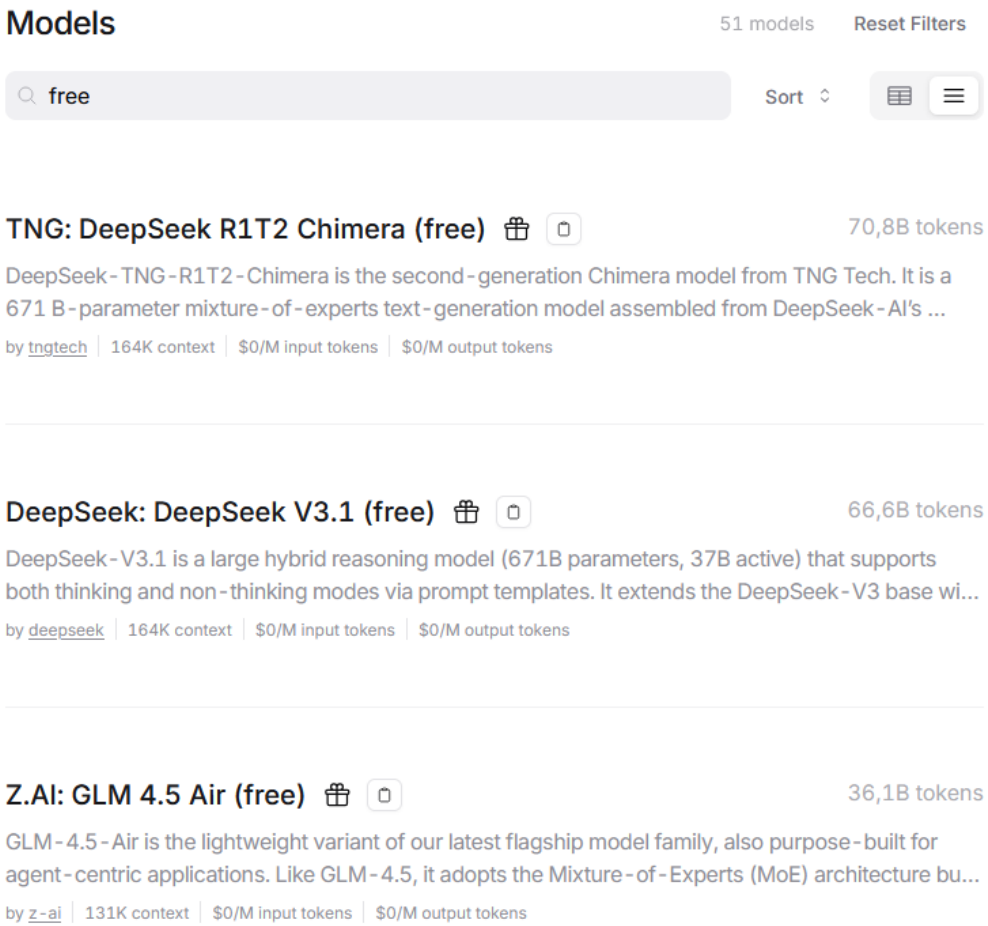
5 Перейти в Settings -> Training, Logging, & Privacy и нажать флажок Enable free endpoints that may publish promts

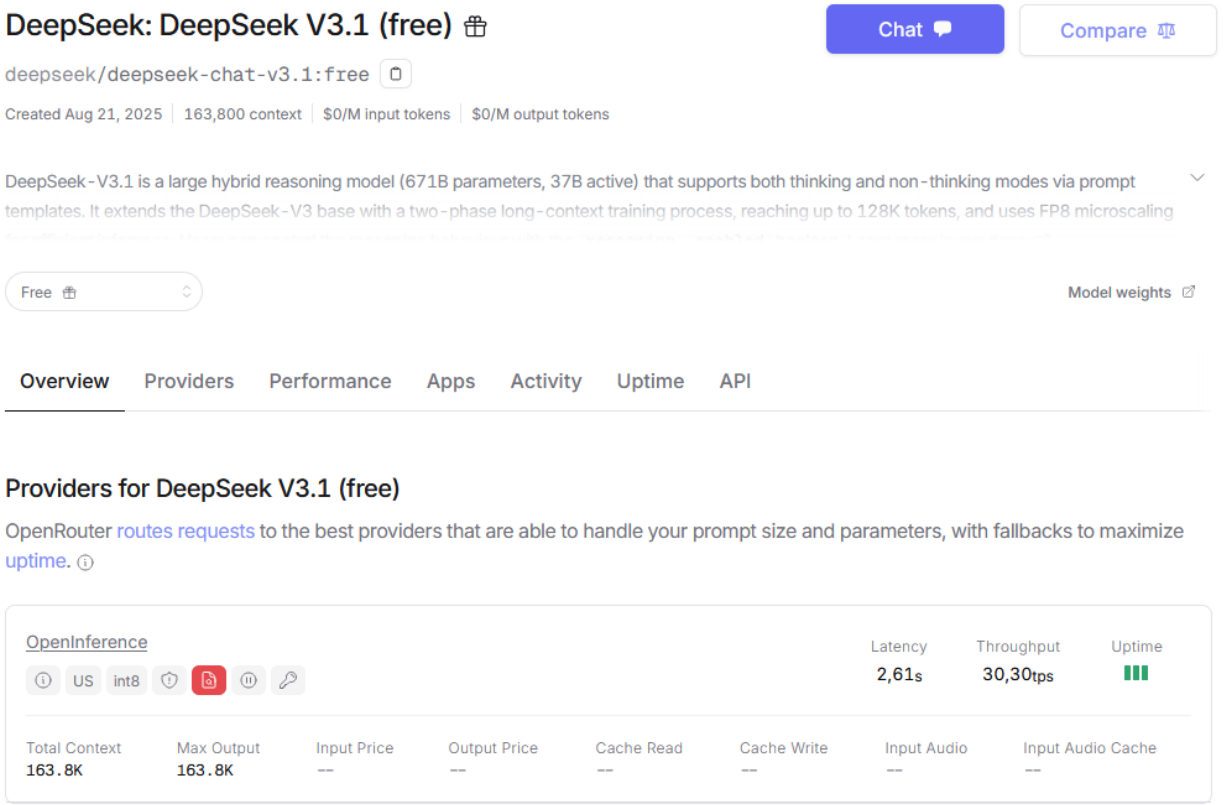




OPENROUTER БЕСПЛАТНЫЕ МОДЕЛИ

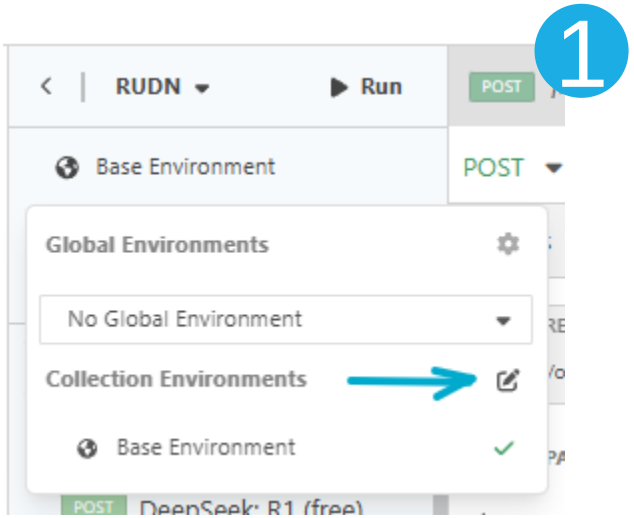
https://openrouter.ai/models?max_price=0&q=free

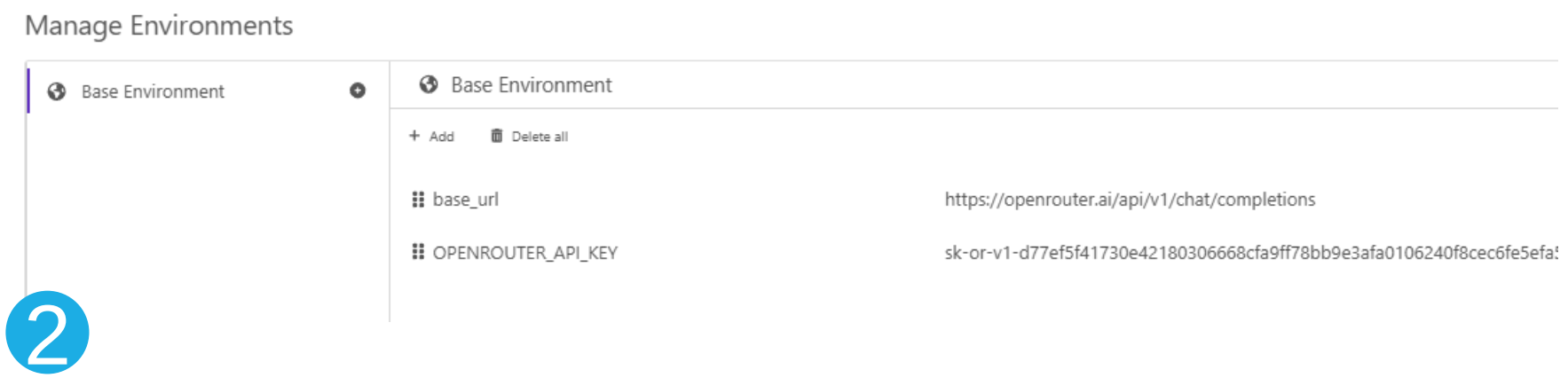




INSOMNIA. НАСТРОЙКА ПЕРЕМЕННЫХ ОКРУЖЕНИЯ

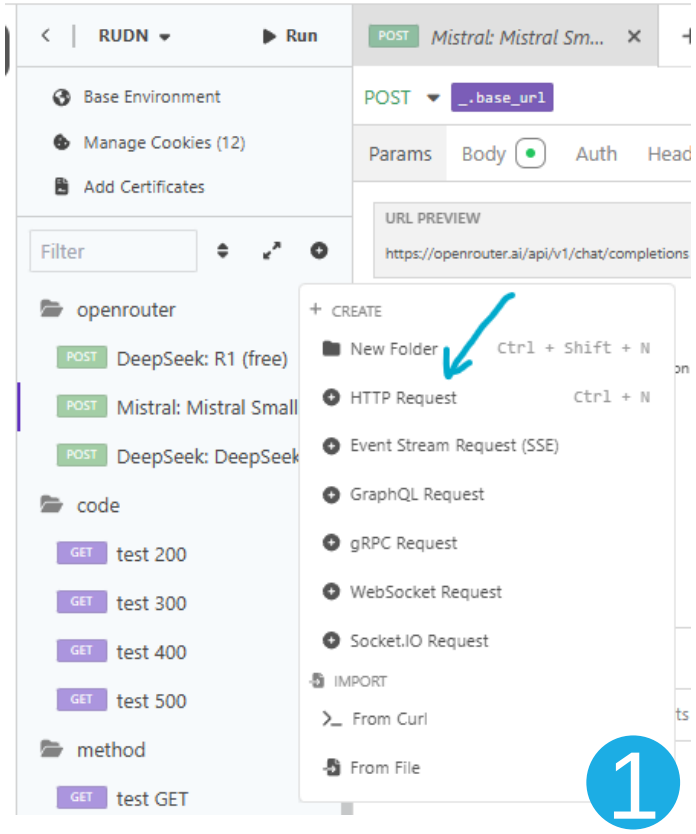
https://insomnia.rest

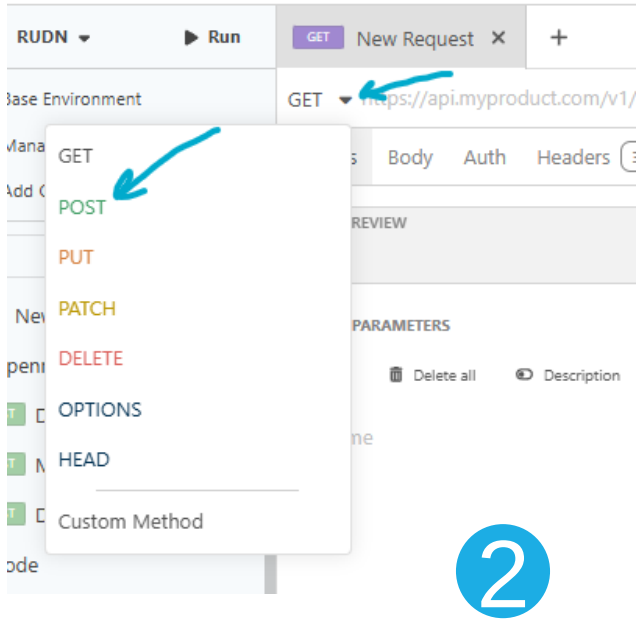


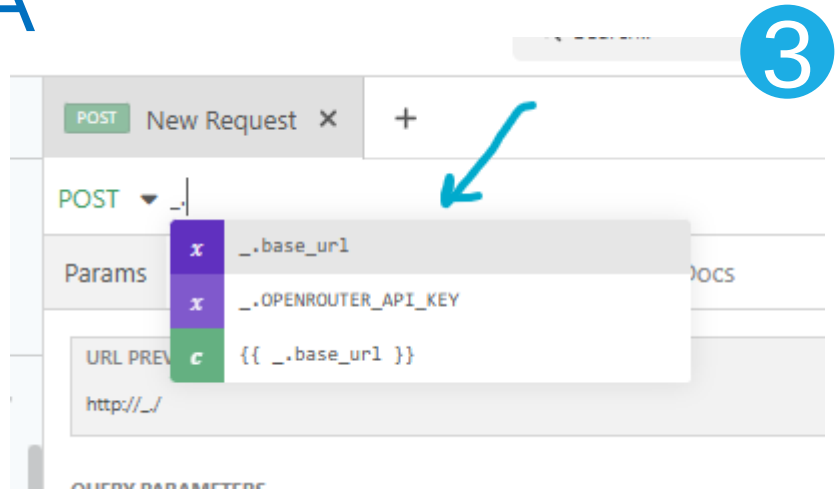


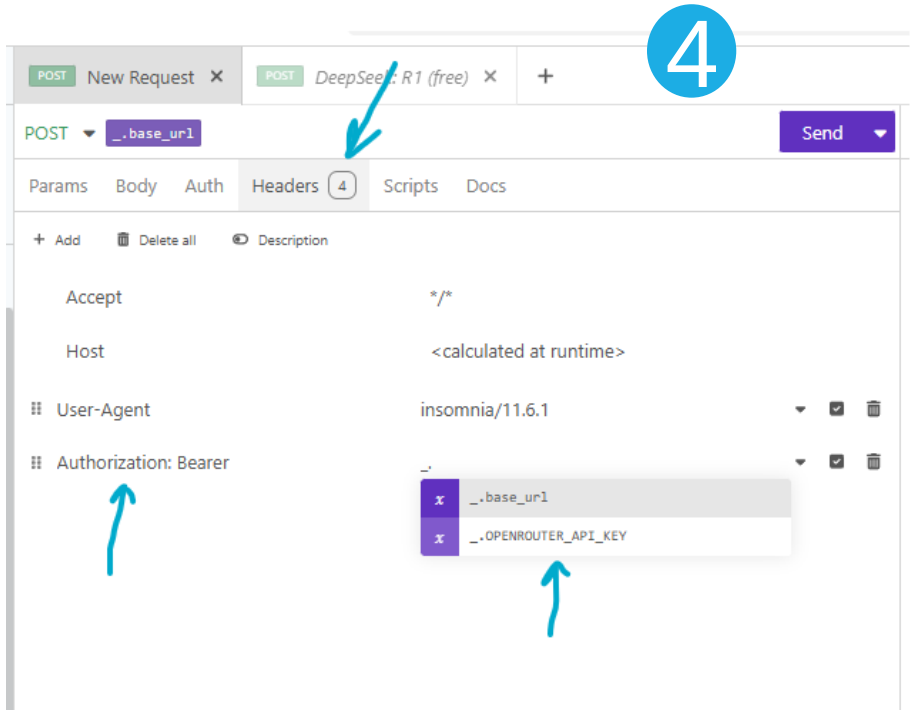
INSOMNIA. СОЗДАНИЕ **POST** ЗАПРОСА

https://insomnia.rest



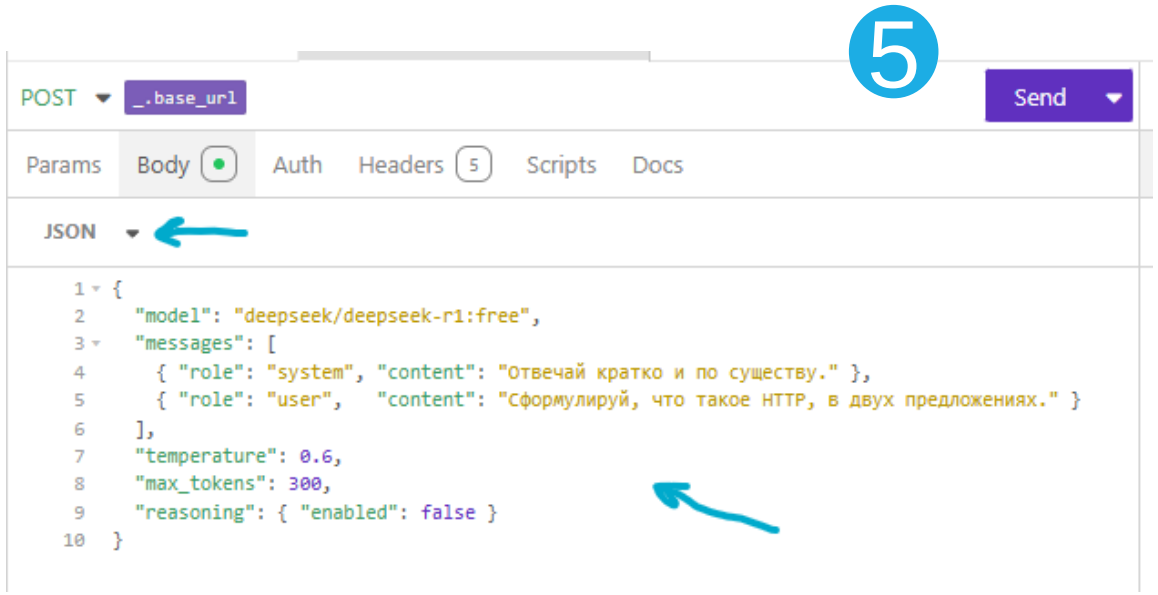


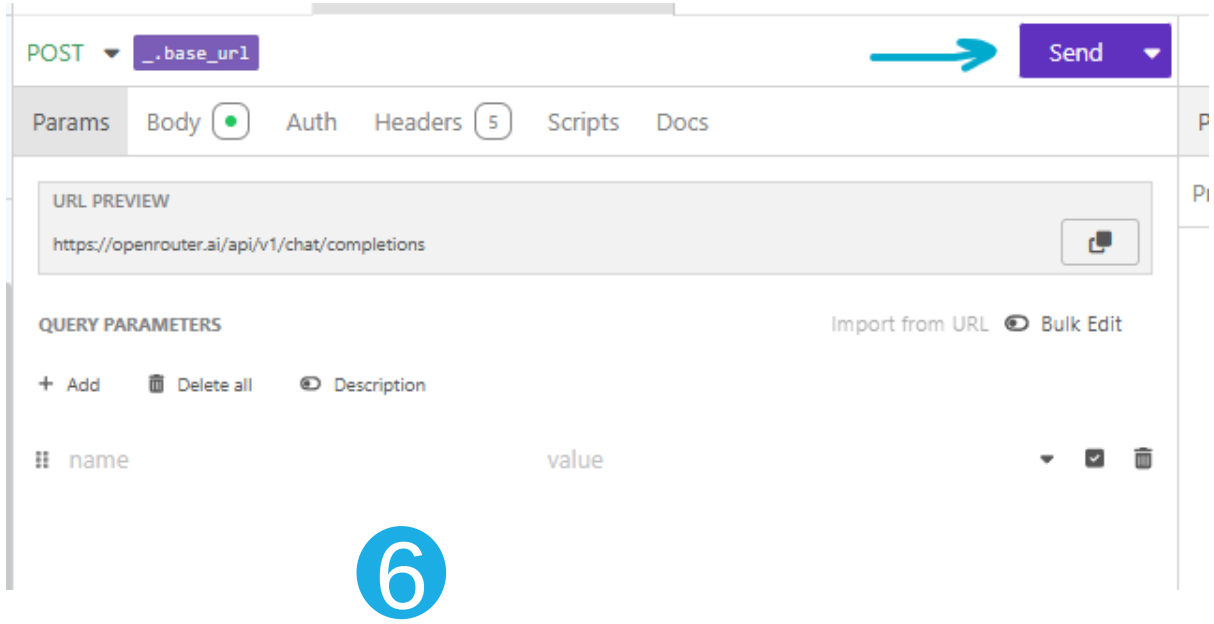




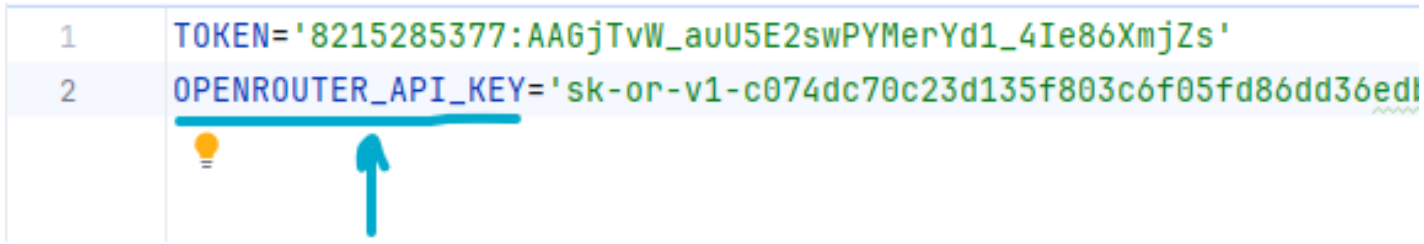
INSOMNIA. СОЗДАНИЕ **POST** ЗАПРОСА

https://insomnia.rest





ПОДГОТОВКА ОКРУЖЕНИЯ



В файл **.env** добавляем ключ **OpenRouter** с названием **OPENROUTER_API_KEY**

СОЗДАНИЕ ТАБЛИЦЫ ДЛЯ ХРАНЕНИЯ МОДЕЛЕЙ

```sql
-- Создаем таблицу с моделями и признаком активной модели
CREATE TABLE IF NOT EXISTS models (
  id     INTEGER PRIMARY KEY,
  key    TEXT NOT NULL UNIQUE,
  label  TEXT NOT NULL,
  active INTEGER NOT NULL DEFAULT 0 CHECK (active IN (0,1))
);
```

Создаем таблицу **models** для хранения информации о различных, если она ещё не существует.

IF NOT EXISTS - проверяет, существует ли таблица. Если существует - запрос пропускается (избегаем ошибок при повторном запуске)

INTEGER/TEXT – типы данных

PRIMARY KEY – первичный ключ, автоматически создает индекс для быстрого поиска

DEFAULT 0 - если значение не указано, по умолчанию 0

CHECK (active IN (0,1)) - ограничение: разрешены только значения 0 или 1

СОЗДАНИЕ ТАБЛИЦЫ ДЛЯ ХРАНЕНИЯ МОДЕЛЕЙ

```
-- Ставим ограничение на поле active - только одна активная модель
CREATE UNIQUE INDEX IF NOT EXISTS ux_models_single_active ON models(active) WHERE active=1;
```

Создаем уникальный индекс (если он ещё не существует), реализующий ограничение – в любой момент времени может быть только 1 запись со значением ACTIVE = 1.

СОЗДАНИЕ ТАБЛИЦЫ ДЛЯ ХРАНЕНИЯ МОДЕЛЕЙ

```sql
-- Добавляем список моделей в таблицу
INSERT OR IGNORE INTO models(id, key, label, active) VALUES
    (1, 'deepseek/deepseek-chat-v3.1:free', 'DeepSeek V3.1 (free)', 1),
    (2, 'deepseek/deepseek-r1:free', 'DeepSeek R1 (free)', 0),
    (3, 'mistralai/mistral-small-24b-instruct-2501:free', 'Mistral Small 24b (free)', 0),
    (4, 'meta-llama/llama-3.1-8b-instruct:free', 'Llama 3.1 8B (free)', 0);
```

Добавляем в таблицу **models** информацию о используемых нами моделях. В качестве активной выбираем первую модель.

СОЗДАНИЕ ФУНКЦИЙ ДЛЯ РАБОТЫ С ДАННЫМИ

Создаем функцию получения списка моделей.

```
265    def list_models() -> list[dict]:  2 usages  new *
266        with _connect() as conn:
267            rows = conn.execute("SELECT id,key,label,active FROM models ORDER BY id").fetchall()
268            return [{"id":r["id"], "key":r["key"], "label":r["label"], "active":bool(r["active"])} for r in rows]
```

СОЗДАНИЕ ФУНКЦИЙ ДЛЯ РАБОТЫ С ДАННЫМИ

Создаем функцию получения активной модели.

```
270    def get_active_model() -> dict:  5 usages  new *
271        with _connect() as conn:
272            row = conn.execute("SELECT id,key,label FROM models WHERE active=1").fetchone()
273            if row:
274                return {"id":row["id"], "key":row["key"], "label":row["label"], "active":True}
275            row = conn.execute("SELECT id,key,label FROM models ORDER BY id LIMIT 1").fetchone()
276            if not row:
277                raise RuntimeError("В реестре моделей нет записей")
278            conn.execute( sql: "UPDATE models SET active=CASE WHEN id=? THEN 1 ELSE 0 END",  parameters: (row["id"],))
279            return {"id":row["id"], "key":row["key"], "label":row["label"], "active":True}
```

СОЗДАНИЕ ФУНКЦИЙ ДЛЯ РАБОТЫ С ДАННЫМИ

Создаем функцию установки признака активности модели.

```
281    def set_active_model(model_id: int) -> dict:  2 usages  new *
282        with _connect() as conn:
283            conn.execute("BEGIN IMMEDIATE")
284            exists = conn.execute( sql: "SELECT 1 FROM models WHERE id=?",  parameters: (model_id,)).fetchone()
285            if not exists:
286                conn.rollback()
287                raise ValueError("Неизвестный ID модели")
288            conn.execute( sql: "UPDATE models SET active=CASE WHEN id=? THEN 1 ELSE 0 END",  parameters: (model_id,))
289            conn.commit()
290        return get_active_model()
```

BEGIN IMMEDIATE – ставим блокировку на запись: другие записи не смогут изменить активность модели в момент обновления записи.

CASE WHEN – устанавливаем active=1 выбранной записи и active=0 всем остальным за один проход

СОЗДАНИЕ КОМАНД БОТА

Создаем команду для получения списка моделей

```
295  @bot.message_handler(commands=["models"])  new *
296  def cmd_models(message: types.Message) -> None:
297      items = list_models()
298      if not items:
299          bot.reply_to(message,  text: "Список моделей пуст.")
300          return
301      lines = ["Доступные модели:"]
302      for m in items:
303          star = "★" if m["active"] else " "
304          lines.append(f"{star} {m['id']}. {m['label']}  [{m['key']}]")
305      lines.append("\nАктивировать: /model <ID>")
306      bot.reply_to(message, "\n".join(lines))
```

СОЗДАНИЕ КОМАНД БОТА

Создаем команду для выбора активной модели

```
308    @bot.message_handler(commands=["model"])  new *
309    def cmd_model(message: types.Message) -> None:
310        arg = message.text.replace( _old: "/model",  _new: "",  _count: 1).strip()
311        if not arg:
312            active = get_active_model()
313            bot.reply_to(message,  text: f"Текущая активная модель: {active['label']} [{active['key']}]\n(сменить: /model <ID> или /models)")
314            return
315        if not arg.isdigit():
316            bot.reply_to(message,  text: "Использование: /model <ID из /models>")
317            return
318        try:
319            active = set_active_model(int(arg))
320            bot.reply_to(message,  text: f"Активная модель переключена: {active['label']} [{active['key']}]")
321        except ValueError:
322            bot.reply_to(message,  text: "Неизвестный ID модели. Сначала /models.")
```

СОЗДАНИЕ КОМАНД БОТА

Добавляем команды в список команд

```python
@bot.message_handler(commands=["start", "help"])  user *
def cmd_start(message: types.Message) -> None:
    """
    Поприветствовать пользователя и кратко описать команды.
    """
    text = (
        "Привет! Это заметочник на SQLite.\n\n"
        "Команды:\n"
        "  /note_add <текст>\n"
        "  /note_list [N]\n"
        "  /note_find <подстрока>\n"
        "  /note_edit <id> <текст>\n"
        "  /note_del <id>\n"
        "  /note_count\n"
        "  /note_export\n"
        "  /note_stats [days]\n"
        "  /models \n"
        "  /model <id>\n"
```

```python
def _setup_bot_commands() -> None:  1 usage  user *
    """
    Регистрирует команды в меню клиента Telegram (удобно для новичков).
    """
    cmds = [
        types.BotCommand( command: "start", description: "Приветствие и помощь"),
        types.BotCommand( command: "note_add", description: "Добавить заметку"),
        types.BotCommand( command: "note_list", description: "Список заметок"),
        types.BotCommand( command: "note_find", description: "Поиск заметок"),
        types.BotCommand( command: "note_edit", description: "Изменить заметку"),
        types.BotCommand( command: "note_del", description: "Удалить заметку"),
        types.BotCommand( command: "note_count", description: "Сколько заметок"),
        types.BotCommand( command: "note_export", description: "Экспорт заметок в .txt"),
        types.BotCommand( command: "note_stats", description: "Статистика по датам"),
        types.BotCommand( command: "model", description: "Установить активную модель"),
        types.BotCommand( command: "models", description: "Получить список моделей"),
```

ИНТЕГРАЦИЯ С **OPENROUTER**

Создаем файл **openrouter_client.py**

```python
from __future__ import annotations
import os, time, requests
from dataclasses import dataclass
from typing import Dict, List, Tuple
from dotenv import load_dotenv

load_dotenv()

OPENROUTER_API = "https://openrouter.ai/api/v1/chat/completions"
OPENROUTER_API_KEY = os.getenv("OPENROUTER_API_KEY")

@dataclass  5 usages  new *
class OpenRouterError(Exception):
    status: int
    msg: str
    def __str__(self) -> str:  new *
        return f"[{self.status}] {self.msg}"

def _friendly(status: int) -> str:  1 usage  new *
    return {
        400: "Неверный формат запроса.",
        401: "Ключ OpenRouter отклонён. Проверьте OPENROUTER_API_KEY.",
        403: "Нет прав доступа к модели.",
        404: "Эндпоинт не найден. Проверьте URL /api/v1/chat/completions.",
        429: "Превышены лимиты бесплатной модели. Попробуйте позднее.",
    }.get(status, "Сервис недоступен. Повторите попытку позже.")

```

Почему отдельный класс для клиента?

1. Единая точка интеграции. Заголовки, базовый URL, таймауты, формат payload, разбор ответа — в одном месте.
2. Если что-то поменяется у OpenRouter, правим один файл, а не весь проект.
3. Валидируем и нормализуем. Проверяем наличие ключа, модели, формируем «дружественные» сообщения об ошибках (401/404/429/5xx), гарантируем одинаковый тип возврата (text: str, latency_ms: int).

ИНТЕГРАЦИЯ С **OPENROUTER**

Создаем файл **openrouter_client.py**

```python
def chat_once(messages: List[Dict], *,  2 usages  new *
              model: str,
              temperature: float = 0.2,
              max_tokens: int = 400,
              timeout_s: int = 30) -> Tuple[str, int]:
    if not OPENROUTER_API_KEY:
        raise OpenRouterError(401, "Отсутствует OPENROUTER_API_KEY (.env).")
    headers = {
        "Authorization": f"Bearer {OPENROUTER_API_KEY}",
        "Content-Type": "application/json",
    }
    payload = {
        "model": model,
        "messages": messages,
        "temperature": temperature,
        "max_tokens": max_tokens,
    }
    t0 = time.perf_counter()
    r = requests.post(OPENROUTER_API, json=payload, headers=headers, timeout=timeout_s)
    dt_ms = int((time.perf_counter() - t0) * 1000)
    if r.status_code // 100 != 2:
        raise OpenRouterError(r.status_code, _friendly(r.status_code))
    try:
        data = r.json()
        text = data["choices"][0]["message"]["content"]
    except Exception:
        raise OpenRouterError(500, "Неожиданная структура ответа OpenRouter.")
    return text, dt_ms
```

СОЗДАНИЕ КОМАНД БОТА

Создаем команду для опроса модели

```python
@bot.message_handler(commands=["ask"])  new *
def cmd_ask(message: types.Message) -> None:
    q = message.text.replace( _old: "/ask",  _new: "",  _count: 1).strip()
    if not q:
        bot.reply_to(message,  text: "Использование: /ask <вопрос>")
        return

    msgs = _build_messages(message.from_user.id, q[:600])
    model_key = get_active_model()["key"]

    try:
        text, ms = chat_once(msgs, model=model_key, temperature=0.2, max_tokens=400)
        out = (text or "").strip()[:4000]          # не переполняем сообщение Telegram
        bot.reply_to(message,  text: f"{out}\n\n({ms} мс; модель: {model_key})")
    except OpenRouterError as e:
        bot.reply_to(message,  text: f"Ошибка: {e}")
    except Exception:
        bot.reply_to(message,  text: "Непредвиденная ошибка.")
```

temperature=0.2 – модель лучше следует указаниям в system

max_tokens=400 – не выходим за ограничение Telegram (~4096 символов)

СОЗДАНИЕ КОМАНД БОТА

Создаем команду для опроса модели

```python
def _build_messages(user_id: int, user_text: str) -> list[dict]:  1 usage  new *
    system = (
        f"Ты отвечаешь кратко и по-существу.\n"
        "Правила:\n"
        "1) Технические ответы давай корректно и по пунктам.\n"
    )
    return [
        {"role": "system", "content": system},
        {"role": "user", "content": user_text},
    ]
```

СОЗДАНИЕ КОМАНД БОТА

Добавляем команду в список команд

```python
@bot.message_handler(commands=["start", "help"])  user *
def cmd_start(message: types.Message) -> None:
    """
    Поприветствовать пользователя и кратко описать команды.
    """
    text = (
        "Привет! Это заметочник на SQLite.\n\n"
        "Команды:\n"
        "  /note_add <текст>\n"
        "  /note_list [N]\n"
        "  /note_find <подстрока>\n"
        "  /note_edit <id> <текст>\n"
        "  /note_del <id>\n"
        "  /note_count\n"
        "  /note_export\n"
        "  /note_stats [days]\n"
        "  /models \n"
        "  /model <id>\n"
        "  /ask <вопрос>\n"
```

```python
def _setup_bot_commands() -> None:  1 usage  user *
    """
    Регистрирует команды в меню клиента Telegram (удобно для новичков).
    """
    cmds = [
        types.BotCommand( command: "start",  description: "Приветствие и помощь"),
        types.BotCommand( command: "note_add",  description: "Добавить заметку"),
        types.BotCommand( command: "note_list",  description: "Список заметок"),
        types.BotCommand( command: "note_find",  description: "Поиск заметок"),
        types.BotCommand( command: "note_edit",  description: "Изменить заметку"),
        types.BotCommand( command: "note_del",  description: "Удалить заметку"),
        types.BotCommand( command: "note_count",  description: "Сколько заметок"),
        types.BotCommand( command: "note_export",  description: "Экспорт заметок в .txt"),
        types.BotCommand( command: "note_stats",  description: "Статистика по датам"),
        types.BotCommand( command: "model",  description: "Установить активную модель"),
        types.BotCommand( command: "models",  description: "Получить список моделей"),
        types.BotCommand( command: "ask",  description: "Задать вопрос модели"),
```

ДОМАШНЯЯ РАБОТА

1. Значение переменной **OPENROUTER_API_KEY** в клиенте Openrouter должно вычитываться из переменных окружения
2. Добавить модели в список моделей до 10 штук. Проверить, что бот работает, список моделей корректен, установка активности модели работает без ошибок
3. Добавить в клиента Openrouter обработку кодов: 500, 502, 503, 504 с пояснением для пользователя