

GEO 平台 MVP 开发指导文档（Trae/Qoder 专用）

 **文档用途：** 交给 Trae / Qoder / Cursor 等 AI 编程工具的开发指导文档**目标：** 让 AI 编程工具能独立完成 GEO 平台 MVP 全栈开发**版本：** v1.0**日期：** 2026-04-22**MVP 范围：** Citation Tracker（AI 引用追踪）+ 用户系统 + 基础 Dashboard

一、项目概述

1.1 产品定义

GEO 平台（Generative Engine Optimization Platform）是一个 SaaS 产品，用于监控和追踪品牌/企业在国内主流 AI 搜索平台（文心一言、Kimi、通义千问等）中的引用可见性。

MVP 核心功能：

- 用户注册/登录/订阅管理
- 添加和管理监控关键词
- 定时在多个 AI 平台执行查询
- 解析结果，检测目标品牌是否被引用
- Dashboard 展示引用趋势
- 报告导出（CSV）

1.2 技术栈

层级	技术	版本要求	说明
前端	Next.js	14.x（App Router）	SSR + API Route
前端 UI	TailwindCSS + shadcn/ui	最新	组件库
前端图表	Recharts	2.x	图表可视化
前端认证	NextAuth.js	4.x	JWT 认证
后端	Python FastAPI	0.109+	异步 API

ORM	SQLAlchemy	2.0+	异步 ORM
数据库	PostgreSQL	15+	主数据库
缓存	Redis	7+	查询缓存 + 任务队列
浏览器自动化	Playwright	1.40+	AI 平台查询
任务调度	APScheduler	3.10+	定时任务
部署	Docker Compose	最新	本地开发

1.3 项目目录结构

代码块

```

1  geo-platform/
2  |— docker-compose.yml      # Docker 编排
3  |— .env.example           # 环境变量模板
4  |
5  |— frontend/              # Next.js 前端
6  |   |— package.json
7  |   |— next.config.js
8  |   |— tailwind.config.js
9  |   |— tsconfig.json
10 |   |— app/
11 |       |— layout.tsx      # 根布局
12 |       |— page.tsx        # 首页 (登录页)
13 |       |— (auth)/
14 |           |— login/page.tsx
15 |           |— register/page.tsx
16 |       |— (dashboard)/
17 |           |— layout.tsx  # Dashboard 布局 (含侧边栏)
18 |           |— page.tsx    # 数据总览
19 |           |— queries/
20 |               |— page.tsx # 查询词列表
21 |               |— [id]/page.tsx
22 |           |— citations/
23 |               |— page.tsx # 引用记录
24 |           |— reports/
25 |               |— page.tsx # 报告导出
26 |           |— settings/
27 |               |— page.tsx # 设置
28 |       |— api/
29 |           |— auth/[...nextauth]/route.ts
30 |           |— proxy/      # 代理请求到 FastAPI
31 |   |— components/
32 |       |— ui/              # shadcn 组件

```

```

33 | | | | └─ charts/
34 | | | | |   └─ CitationTrend.tsx
35 | | | | |   └─ PlatformCompare.tsx
36 | | | | └─ layout/
37 | | | | |   └─ Sidebar.tsx
38 | | | | |   └─ Header.tsx
39 | | └─ lib/
40 | | |   └─ api.ts           # API 客户端
41 | | |   └─ utils.ts
42 | |
43 | └─ backend/             # FastAPI 后端
44 | |   └─ requirements.txt
45 | |   └─ Dockerfile
46 | |   └─ app/
47 | | |   └─ main.py         # 入口
48 | | |   └─ config.py      # 配置
49 | | |   └─ database.py    # 数据库连接
50 | | |   └─ models/
51 | | | |   └─ user.py
52 | | | |   └─ query.py
53 | | | |   └─ citation.py
54 | | |   └─ schemas/
55 | | | |   └─ user.py
56 | | | |   └─ query.py
57 | | | |   └─ citation.py
58 | | |   └─ api/
59 | | | |   └─ deps.py       # 依赖注入 (认证)
60 | | | |   └─ auth.py
61 | | | |   └─ queries.py
62 | | | |   └─ citations.py
63 | | |   └─ services/
64 | | | |   └─ auth_service.py
65 | | | |   └─ query_service.py
66 | | | |   └─ citation_engine.py
67 | | |   └─ workers/
68 | | | |   └─ scheduler.py  # 定时任务
69 | | | |   └─ utils/
70 | | | | |   └─ playwright_manager.py
71 | | | | |   └─ parser.py
72 | | └─ alembic/         # 数据库迁移
73 | | |   └─ ...
74 | |
75 | └─ tests/             # 测试
76 | |   └─ test_citation_engine.py
77 | |   └─ test_api.py
78 |

```

二、数据库设计

2.1 完整建表 SQL

代码块

```
1  -- 用户表
2  CREATE TABLE users (
3      id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
4      email VARCHAR(255) UNIQUE NOT NULL,
5      password_hash VARCHAR(255) NOT NULL,
6      name VARCHAR(100),
7      plan VARCHAR(20) DEFAULT 'free', -- free/starter/pro/business
8      max_queries INTEGER DEFAULT 5,
9      is_active BOOLEAN DEFAULT TRUE,
10     created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
11     updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
12 );
13
14 -- 查询词表
15 CREATE TABLE queries (
16     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
17     user_id UUID REFERENCES users(id) ON DELETE CASCADE NOT NULL,
18     keyword VARCHAR(200) NOT NULL,
19     target_brand VARCHAR(100) NOT NULL,
20     brand_aliases JSONB DEFAULT '[]', -- 品牌别名列表
21     platforms JSONB NOT NULL DEFAULT '["wenxin", "kimi"]', -- 监控平台
22     frequency VARCHAR(20) DEFAULT 'weekly', -- daily/weekly
23     status VARCHAR(20) DEFAULT 'active', -- active/paused
24     last_queried_at TIMESTAMP WITH TIME ZONE,
25     next_query_at TIMESTAMP WITH TIME ZONE,
26     created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
27     updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
28 );
29
30 -- 引用记录表
31 CREATE TABLE citation_records (
32     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
33     query_id UUID REFERENCES queries(id) ON DELETE CASCADE NOT NULL,
34     platform VARCHAR(50) NOT NULL,
35     cited BOOLEAN NOT NULL DEFAULT FALSE,
36     citation_position INTEGER, -- 第几个提到 (1-10)
37     citation_text TEXT, -- 被引用的内容片段
38     competitor_brands JSONB DEFAULT '[]',
39     raw_response TEXT, -- 原始返回 (用于人工复核)
```

```

40     queried_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
41 );
42
43 -- 查询任务表
44 CREATE TABLE query_tasks (
45     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
46     query_id UUID REFERENCES queries(id) ON DELETE CASCADE NOT NULL,
47     platform VARCHAR(50) NOT NULL,
48     status VARCHAR(20) DEFAULT 'pending', -- pending/running/success/failed
49     error_message TEXT,
50     scheduled_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
51     started_at TIMESTAMP WITH TIME ZONE,
52     completed_at TIMESTAMP WITH TIME ZONE
53 );
54
55 -- 订阅/支付表
56 CREATE TABLE subscriptions (
57     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
58     user_id UUID REFERENCES users(id) ON DELETE CASCADE NOT NULL,
59     plan VARCHAR(20) NOT NULL,
60     status VARCHAR(20) DEFAULT 'active',
61     start_date DATE NOT NULL,
62     end_date DATE NOT NULL,
63     amount DECIMAL(10, 2),
64     payment_method VARCHAR(50), -- wechat/alipay
65     payment_id VARCHAR(255), -- 第三方支付流水号
66     created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
67 );
68
69 -- 索引
70 CREATE INDEX idx_queries_user_id ON queries(user_id);
71 CREATE INDEX idx_queries_status ON queries(status);
72 CREATE INDEX idx_queries_next_query_at ON queries(next_query_at);
73 CREATE INDEX idx_citation_records_query_id ON citation_records(query_id);
74 CREATE INDEX idx_citation_records_queried_at ON citation_records(queried_at);
75 CREATE INDEX idx_citation_records_platform ON citation_records(platform);
76 CREATE INDEX idx_query_tasks_status ON query_tasks(status);
77

```

2.2 平台枚举值

枚举值	中文名	URL	接入方式
wenxin	文心一言	https://yiyan.baidu.com	Playwright

kimi	Kimi	https://kimi.moonshot.cn	Playwright 或 API
tongyi	通义千问	https://tongyi.aliyun.com	Playwright 或 API
baidu_ai	百度 AI 搜索	https://www.baidu.com	Playwright
yuanbao	腾讯元宝	https://yuanbao.tencent.com	Playwright
qingyan	智谱清言	https://chatglm.cn	API (推荐)

三、API 设计

3.1 认证模块

代码块

```
1  POST /api/v1/auth/register
2  Request:
3  {
4    "email": "user@example.com",
5    "password": "secure_password_123",
6    "name": "张三"
7  }
8  Response 201:
9  {
10   "id": "uuid",
11   "email": "user@example.com",
12   "name": "张三",
13   "plan": "free",
14   "max_queries": 5
15 }
16
17 POST /api/v1/auth/login
18 Request:
19 {
20   "email": "user@example.com",
21   "password": "secure_password_123"
22 }
23 Response 200:
24 {
25   "access_token": "jwt_token_string",
26   "token_type": "bearer",
```

```
27     "user": { "id": "uuid", "email": "...", "name": "...", "plan": "free" }
28   }
29
30   GET /api/v1/auth/me
31   Headers: Authorization: Bearer <token>
32   Response 200: 同注册响应
33
```

3.2 查询词模块

代码块

```
1   GET /api/v1/queries
2   Headers: Authorization: Bearer <token>
3   Response 200:
4   {
5     "items": [
6       {
7         "id": "uuid",
8         "keyword": "个人养老金推荐",
9         "target_brand": "中国平安",
10        "platforms": ["wenxin", "kimi"],
11        "frequency": "weekly",
12        "status": "active",
13        "last_queried_at": "2026-04-20T10:00:00Z",
14        "created_at": "2026-04-15T08:00:00Z"
15      }
16    ],
17    "total": 15
18  }
19
20  POST /api/v1/queries
21  Headers: Authorization: Bearer <token>
22  Request:
23  {
24    "keyword": "个人养老金推荐",
25    "target_brand": "中国平安",
26    "brand_aliases": ["平安寿险", "平安保险"],
27    "platforms": ["wenxin", "kimi", "tongyi"],
28    "frequency": "weekly"
29  }
30  Response 201: 创建的查询对象
31
32  PUT /api/v1/queries/{query_id}
33  Headers: Authorization: Bearer <token>
34  Request:
```

```
35  {
36    "keyword": "30岁养老金推荐",
37    "status": "paused"
38  }
39  Response 200: 更新后的对象
40
41  DELETE /api/v1/queries/{query_id}
42  Headers: Authorization: Bearer <token>
43  Response 204: No Content
44
```

3.3 引用数据模块

代码块

```
1  GET /api/v1/citations?query_id={id}&platform=wenxin&start_date=2026-04-
   01&end_date=2026-04-22
2  Headers: Authorization: Bearer <token>
3  Response 200:
4  {
5    "items": [
6      {
7        "id": "uuid",
8        "query_id": "uuid",
9        "platform": "wenxin",
10       "cited": true,
11       "citation_position": 2,
12       "citation_text": "中国平安是国内领先的保险品牌...",
13       "competitor_brands": ["中国人寿", "太平洋保险"],
14       "queried_at": "2026-04-22T02:00:00Z"
15     }
16   ],
17   "total": 120
18 }
19
20 GET /api/v1/citations/stats?query_id={id}
21 Headers: Authorization: Bearer <token>
22 Response 200:
23 {
24   "total_queries": 24,
25   "total_citations": 12,
26   "citation_rate": 0.50,
27   "avg_position": 2.3,
28   "by_platform": {
29     "wenxin": { "queries": 6, "citations": 4, "rate": 0.67, "avg_position":
   2.0 },
```

```
30     "kimi": { "queries": 6, "citations": 3, "rate": 0.50, "avg_position": 3.2
    },
31     "tongyi": { "queries": 6, "citations": 2, "rate": 0.33, "avg_position":
    4.1 }
32 },
33 "trend": [
34     { "date": "2026-04-01", "citations": 2 },
35     { "date": "2026-04-08", "citations": 3 },
36     { "date": "2026-04-15", "citations": 4 },
37     { "date": "2026-04-22", "citations": 3 }
38 ]
39 }
40
41 POST /api/v1/queries/{query_id}/run-now
42 Headers: Authorization: Bearer <token>
43 Response 202:
44 {
45     "task_id": "uuid",
46     "status": "pending",
47     "message": "查询任务已加入队列"
48 }
49
```

3.4 报告导出

代码块

```
1 GET /api/v1/reports/export?query_id={id}&format=csv
2 Headers: Authorization: Bearer <token>
3 Response: CSV 文件下载
4 Content-Disposition: attachment; filename="geo-report-2026-04.csv"
5 Content-Type: text/csv
6
```

四、核心代码实现

4.1 Citation Engine（引用检测引擎）

代码块

```
1 # backend/app/services/citation_engine.py
2
3 import asyncio
```

```

4 import re
5 from datetime import datetime
6 from typing import Optional
7 from playwright.async_api import async_playwright, Page
8 from difflib import SequenceMatcher
9
10 # AI 平台配置
11 PLATFORMS = {
12     "wenxin": {
13         "url": "https://yiyao.baidu.com",
14         "input_selector": "textarea",
15         "send_selector": "button[type='submit']",
16         "answer_selector": ".answer-content",
17     },
18     "kimi": {
19         "url": "https://kimi.moonshot.cn",
20         "input_selector": "textarea",
21         "send_selector": "button.send-btn",
22         "answer_selector": ".response-content",
23     },
24     "tongyi": {
25         "url": "https://tongyi.aliyun.com/qianwen/",
26         "input_selector": "textarea",
27         "send_selector": "button.send-button",
28         "answer_selector": ".markdown-body",
29     },
30 }
31
32
33 class CitationEngine:
34     """AI 平台引用检测引擎"""
35
36     def __init__(self, db_session):
37         self.db = db_session
38
39     async def run_query(self, query_id: str, platform: str) -> dict:
40         """执行单个查询"""
41         config = PLATFORMS.get(platform)
42         if not config:
43             raise ValueError(f"不支持的平台: {platform}")
44
45         # 从数据库获取查询信息
46         query = await self._get_query(query_id)
47         if not query:
48             raise ValueError(f"查询词不存在: {query_id}")
49
50         # 使用 Playwright 查询

```

```

51     result = await self._query_platform(platform, query.keyword)
52
53     # 检测引用
54     citation = self._detect_citation(
55         response_text=result.get("answer", ""),
56         brand_name=query.target_brand,
57         aliases=query.brand_aliases or [],
58     )
59
60     # 保存结果
61     await self._save_result(
62         query_id=query_id,
63         platform=platform,
64         citation=citation,
65         raw_response=result.get("answer", ""),
66     )
67
68     return citation
69
70     async def _query_platform(self, platform: str, keyword: str) -> dict:
71         """在指定 AI 平台执行查询"""
72         config = PLATFORMS[platform]
73         result = {"answer": "", "sources": []}
74
75         async with async_playwright() as p:
76             browser = await p.chromium.launch(
77                 headless=True,
78                 args=["--no-sandbox", "--disable-setuid-sandbox"],
79             )
80             context = await browser.new_context(
81                 user_agent="Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36",
82                 viewport={"width": 1920, "height": 1080},
83             )
84             page = await context.new_page()
85
86             try:
87                 # 导航到平台
88                 await page.goto(config["url"], wait_until="domcontentloaded",
timeout=30000)
89
90                 # 等待输入框
91                 await page.wait_for_selector(config["input_selector"],
timeout=10000)
92
93                 # 输入查询词
94                 await page.fill(config["input_selector"], keyword)

```

```

95
96     # 点击发送
97     await page.click(config["send_selector"])
98
99     # 等待结果 (最多 60 秒)
100    try:
101        await page.wait_for_selector(
102            config["answer_selector"], timeout=60000
103        )
104    except Exception:
105        return result # 超时, 返回空结果
106
107    # 提取回答内容
108    answer_el = await
page.query_selector(config["answer_selector"])
109    if answer_el:
110        result["answer"] = await answer_el.inner_text()
111
112    # 提取引用来源 (如果有的话)
113    source_els = await page.query_selector_all(".source-item,
.reference")
114    result["sources"] = [
115        await el.inner_text() for el in source_els[:10]
116    ]
117
118    except Exception as e:
119        result["error"] = str(e)
120    finally:
121        await browser.close()
122
123    return result
124
125    def _detect_citation(
126        self, response_text: str, brand_name: str, aliases: list
127    ) -> dict:
128        """检测品牌是否被引用"""
129        text_lower = response_text.lower()
130        brand_lower = brand_name.lower()
131
132        # 1. 精确匹配
133        exact_match = brand_lower in text_lower
134
135        # 2. 别名匹配
136        alias_match = False
137        for alias in aliases:
138            if alias.lower() in text_lower:
139                alias_match = True

```

```

140         break
141
142     # 3. 模糊匹配 (品牌名可能被拆分或改写)
143     sentences = re.split(r"[。！？\n]+", response_text)
144     best_match = {"similarity": 0, "text": ""}
145     for sentence in sentences:
146         if len(sentence.strip()) < 5:
147             continue
148         sim = SequenceMatcher(
149             None, brand_lower, sentence.lower()
150         ).ratio()
151         if sim > best_match["similarity"]:
152             best_match = {"similarity": sim, "text": sentence.strip()}
153
154     cited = exact_match or alias_match or best_match["similarity"] > 0.4
155
156     return {
157         "cited": cited,
158         "match_type": (
159             "exact" if exact_match else "alias" if alias_match else "fuzzy"
160         ),
161         "citation_text": best_match["text"] if best_match["similarity"] >
0.3 else "",
162         "confidence": (
163             1.0 if exact_match
164             else 0.9 if alias_match
165             else max(best_match["similarity"], 0)
166         ),
167         "position": self._find_position(response_text, brand_name,
aliases) if cited else None,
168     }
169
170     def _find_position(self, text: str, brand: str, aliases: list) -> int:
171         """查找品牌在回答中的位置 (第几个段落)"""
172         paragraphs = re.split(r"\n{2,}", text)
173         for i, para in enumerate(paragraphs, 1):
174             if brand.lower() in para.lower():
175                 return i
176             for alias in aliases:
177                 if alias.lower() in para.lower():
178                     return i
179         return 1
180
181     # 数据库操作方法省略 (使用 SQLAlchemy async session)
182

```

4.2 定时调度器

代码块

```
1  # backend/app/workers/scheduler.py
2
3  from apscheduler.schedulers.asyncio import AsyncIOScheduler
4  from apscheduler.triggers.interval import IntervalTrigger
5  from sqlalchemy.ext.asyncio import AsyncSession
6  from app.services.citation_engine import CitationEngine
7  from app.models.query import Query
8
9
10 async def setup_scheduler(db_session_factory):
11     """初始化定时调度器"""
12     scheduler = AsyncIOScheduler()
13
14     scheduler.add_job(
15         execute_pending_queries,
16         IntervalTrigger(hours=1),
17         id="execute_queries",
18         args=[db_session_factory],
19         replace_existing=True,
20     )
21
22     scheduler.start()
23     return scheduler
24
25
26 async def execute_pending_queries(db_session_factory):
27     """执行待查询任务"""
28     async with db_session_factory() as session:
29         # 找出需要执行的查询
30         queries = await session.execute(
31             select(Query).where(
32                 Query.status == "active",
33                 Query.next_query_at <= datetime.utcnow(),
34             )
35         )
36
37         for query in queries.scalars().all():
38             for platform in query.platforms:
39                 # 创建任务记录
40                 task = QueryTask(
41                     query_id=query.id,
42                     platform=platform,
43                     status="pending",
```

```

44         )
45         session.add(task)
46         await session.commit()
47
48         # 异步执行
49         engine = CitationEngine(session)
50         try:
51             await engine.run_query(str(query.id), platform)
52             task.status = "success"
53         except Exception as e:
54             task.status = "failed"
55             task.error_message = str(e)
56         finally:
57             task.completed_at = datetime.utcnow()
58             await session.commit()
59
60         # 更新下次执行时间
61         if query.frequency == "daily":
62             query.next_query_at = datetime.utcnow() + timedelta(hours=24)
63         else:
64             query.next_query_at = datetime.utcnow() + timedelta(weeks=1)
65         await session.commit()
66

```

4.3 FastAPI 主入口

代码块

```

1  # backend/app/main.py
2
3  from fastapi import FastAPI
4  from fastapi.middleware.cors import CORSMiddleware
5  from app.api import auth, queries, citations
6  from app.database import engine, Base
7  from app.workers.scheduler import setup_scheduler
8  from app.database import async_session_factory
9
10 app = FastAPI(title="GEO Platform API", version="0.1.0")
11
12 # CORS
13 app.add_middleware(
14     CORSMiddleware,
15     allow_origins=["http://localhost:3000"],
16     allow_credentials=True,
17     allow_methods=["*"],
18     allow_headers=["*"],

```

```
19 )
20
21 # 路由
22 app.include_router(auth.router, prefix="/api/v1/auth", tags=["认证"])
23 app.include_router(queries.router, prefix="/api/v1", tags=["查询管理"])
24 app.include_router(citations.router, prefix="/api/v1", tags=["引用数据"])
25
26
27 @app.on_event("startup")
28 async def startup():
29     """应用启动时创建表和调度器"""
30     async with engine.begin() as conn:
31         await conn.run_sync(Base.metadata.create_all)
32     app.state.scheduler = await setup_scheduler(async_session_factory)
33
34
35 @app.get("/api/v1/health")
36 async def health_check():
37     return {"status": "ok", "version": "0.1.0"}
38
```

五、前端页面实现

5.1 Dashboard 首页（数据总览）

代码块

```
1 // frontend/app/(dashboard)/page.tsx
2
3 "use client";
4
5 import { useEffect, useState } from "react";
6 import { Card, CardContent, CardHeader, CardTitle } from
"@/components/ui/card";
7 import { CitationTrend } from "@/components/charts/CitationTrend";
8 import { PlatformCompare } from "@/components/charts/PlatformCompare";
9 import { api } from "@/lib/api";
10
11 export default function DashboardPage() {
12     const [stats, setStats] = useState<any>(null);
13
14     useEffect(() => {
15         api.get("/api/v1/citations/stats").then(setStats);
16     }, []);
```

```
17
18   if (!stats) return <div>加载中...</div>;
19
20   return (
21     <div className="space-y-6">
22       <h1 className="text-2xl font-bold">数据总览</h1>
23
24       { /* 统计卡片 */ }
25       <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
26         <Card>
27           <CardHeader>
28             <CardTitle className="text-sm">总查询次数</CardTitle>
29           </CardHeader>
30           <CardContent>
31             <div className="text-3xl font-bold">{stats.total_queries}</div>
32           </CardContent>
33         </Card>
34         <Card>
35           <CardHeader>
36             <CardTitle className="text-sm">总引用次数</CardTitle>
37           </CardHeader>
38           <CardContent>
39             <div className="text-3xl font-bold">{stats.total_citations}</div>
40           </CardContent>
41         </Card>
42         <Card>
43           <CardHeader>
44             <CardTitle className="text-sm">引用率</CardTitle>
45           </CardHeader>
46           <CardContent>
47             <div className="text-3xl font-bold">
48               {(stats.citation_rate * 100).toFixed(1)}%
49             </div>
50           </CardContent>
51         </Card>
52         <Card>
53           <CardHeader>
54             <CardTitle className="text-sm">平均引用位置</CardTitle>
55           </CardHeader>
56           <CardContent>
57             <div className="text-3xl font-bold">{stats.avg_position}</div>
58           </CardContent>
59         </Card>
60       </div>
61
62       { /* 图表 */ }
63       <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
```

```

64     <Card>
65         <CardHeader>
66             <CardTitle>引用趋势 (过去 30 天) </CardTitle>
67         </CardHeader>
68         <CardContent>
69             <CitationTrend data={stats.trend} />
70         </CardContent>
71     </Card>
72     <Card>
73         <CardHeader>
74             <CardTitle>平台对比</CardTitle>
75         </CardHeader>
76         <CardContent>
77             <PlatformCompare data={stats.by_platform} />
78         </CardContent>
79     </Card>
80 </div>
81 </div>
82 );
83 }
84

```

5.2 查询词列表页

代码块

```

1  // frontend/app/(dashboard)/queries/page.tsx
2
3  "use client";
4
5  import { useEffect, useState } from "react";
6  import { Button } from "@/components/ui/button";
7  import {
8      Table,
9      TableBody,
10     TableCell,
11     TableHead,
12     TableHeader,
13     TableRow,
14 } from "@/components/ui/table";
15 import { Badge } from "@/components/ui/badge";
16 import { api } from "@/lib/api";
17 import { useRouter } from "next/navigation";
18
19 export default function QueriesPage() {
20     const [queries, setQueries] = useState<any[]>([]);

```

```

21  const router = useRouter();
22
23  useEffect(() => {
24    api.get("/api/v1/queries").then((data) => setQueries(data.items));
25  }, []);
26
27  const platformNames: Record<string, string> = {
28    wenxin: "文心一言",
29    kimi: "Kimi",
30    tongyi: "通义千问",
31    baidu_ai: "百度 AI",
32    yuanbao: "腾讯元宝",
33    qingyan: "智谱清言",
34  };
35
36  return (
37    <div className="space-y-6">
38      <div className="flex justify-between items-center">
39        <h1 className="text-2xl font-bold">监控查询</h1>
40        <Button onClick={() => router.push("/queries/new")}>添加查询</Button>
41      </div>
42
43      <Table>
44        <TableHeader>
45          <TableRow>
46            <TableHead>关键词</TableHead>
47            <TableHead>目标品牌</TableHead>
48            <TableHead>平台</TableHead>
49            <TableHead>频率</TableHead>
50            <TableHead>状态</TableHead>
51            <TableHead>最后查询</TableHead>
52          </TableRow>
53        </TableHeader>
54        <TableBody>
55          {queries.map((q) => (
56            <TableRow
57              key={q.id}
58              className="cursor-pointer hover:bg-gray-50"
59              onClick={() => router.push(`/citations?query_id=${q.id}`)}
60            >
61            <TableCell className="font-medium">{q.keyword}</TableCell>
62            <TableCell>{q.target_brand}</TableCell>
63            <TableCell>
64              {q.platforms.map((p: string) => (
65                <Badge key={p} variant="outline" className="mr-1">
66                  {platformNames[p] || p}
67                </Badge>

```

```
68         }}
69     </TableCell>
70     <TableCell>{q.frequency === "daily" ? "每天" : "每周"}</TableCell>
71     <TableCell>
72         <Badge
73             variant={q.status === "active" ? "default" : "secondary"}
74         >
75             {q.status === "active" ? "活跃" : "暂停"}
76         </Badge>
77     </TableCell>
78     <TableCell>
79         {q.last_queried_at
80             ? new Date(q.last_queried_at).toLocaleString("zh-CN")
81             : "未查询"}
82     </TableCell>
83 </TableRow>
84     }}
85 </TableBody>
86 </Table>
87 </div>
88 );
89 }
90
```

六、环境配置

6.1 .env 文件

代码块

```
1 # 数据库
2 DATABASE_URL=postgresql+asyncpg://postgres:postgres123@db:5432/geo_platform
3
4 # Redis
5 REDIS_URL=redis://redis:6379/0
6
7 # JWT
8 JWT_SECRET=your-secret-key-change-in-production
9 JWT_EXPIRE_HOURS=24
10
11 # 前端
12 NEXT_PUBLIC_API_URL=http://localhost:8000
13
14 # Playwright
```

```
15 PLAYWRIGHT_BROWSERS_PATH=/ms-playwright
16
17 # 国内大模型 API (可选, 用于内容分析模块)
18 ZHIPU_API_KEY=
19 TONGYI_API_KEY=
20
```


6.2 docker-compose.yml

代码块

```
1  version: "3.8"
2
3  services:
4    db:
5      image: postgres:15-alpine
6      environment:
7        POSTGRES_USER: postgres
8        POSTGRES_PASSWORD: postgres123
9        POSTGRES_DB: geo_platform
10     ports:
11       - "5432:5432"
12     volumes:
13       - postgres_data:/var/lib/postgresql/data
14
15     redis:
16       image: redis:7-alpine
17       ports:
18         - "6379:6379"
19
20     backend:
21       build: ./backend
22       ports:
23         - "8000:8000"
24       environment:
25         DATABASE_URL:
26           postgresql+asyncpg://postgres:postgres123@db:5432/geo_platform
27         REDIS_URL: redis://redis:6379/0
28         JWT_SECRET: dev-secret-key
29       depends_on:
30         - db
31         - redis
32       volumes:
33         - ./backend:/app
34
35     frontend:
```

```
35     build: ./frontend
36     ports:
37       - "3000:3000"
38     environment:
39       NEXT_PUBLIC_API_URL: http://localhost:8000
40     depends_on:
41       - backend
42     volumes:
43       - ./frontend:/app
44       - /app/node_modules
45
46 volumes:
47   postgres_data:
48
```

七、开发步骤（按顺序执行）

 **重要：**请按以下步骤顺序开发，每完成一步再进入下一步。不要跳步。

Step 1: 项目初始化（第 1-2 天）

代码块

```
1  # 1. 创建项目目录
2  mkdir geo-platform && cd geo-platform
3
4  # 2. 初始化后端
5  mkdir backend && cd backend
6  python -m venv venv
7  source venv/bin/activate
8  pip install fastapi uvicorn sqlalchemy asyncpg alembic python-jose bcrypt
   httpx apscheduler playwright
9  playwright install chromium
10 cd ..
11
12 # 3. 初始化前端
13 npx create-next-app@latest frontend --typescript --tailwind --app
14 cd frontend
15 npm install recharts next-auth lucide-react axios
16 npx shadcn@latest init
17 npx shadcn@latest add button card table badge input dialog form
18 cd ..
```

```
19
20 # 4. 创建 docker-compose.yml
21 # 5. 创建 .env 文件
22 # 6. 启动 docker compose
23 docker compose up -d db redis
24
```

Step 2: 数据库 + ORM (第 3-4 天)

- 创建 `backend/app/database.py` (数据库连接)
- 创建 `backend/app/models/` (5 个模型文件)
- 运行 alembic 初始化并生成迁移
- 验证建表成功

Step 3: 认证模块 (第 5-6 天)

- 实现 `POST /api/v1/auth/register`
- 实现 `POST /api/v1/auth/login`
- 实现 JWT 中间件
- 实现 `GET /api/v1/auth/me`
- 前端实现登录/注册页面
- 实现 NextAuth 集成

Step 4: 查询词 CRUD (第 7-8 天)

- 实现查询词的增删改查 API
- 前端实现查询词列表页
- 前端实现添加/编辑表单
- 添加查询数量限制 (根据用户 plan)

Step 5: Citation Engine (第 9-14 天)

- 实现 Playwright 查询引擎 (先做文心一言 + Kimi)
- 实现引用检测算法
- 实现结果保存
- 手动测试验证

Step 6: 定时调度器 (第 15-16 天)

- 实现 APScheduler 调度

- 实现任务状态追踪
- 测试定时执行

Step 7: Dashboard + 报告 (第 17-20 天)

- 实现统计数据 API
- 实现前端图表
- 实现 CSV 导出
- 前端联调

Step 8: 测试 + 部署 (第 21-22 天)

- 端到端测试
 - 修复 bug
 - Docker 构建
 - 部署到阿里云
-

八、验收标准

8.1 MVP 必须满足

- 用户可注册、登录
- 可添加/编辑/删除查询词 (至少 5 个)
- 支持文心一言 + Kimi 两个平台的自动查询
- 引用检测准确率 $\geq 70\%$ (通过人工抽检验证)
- Dashboard 显示引用趋势图和平台对比图
- 支持 CSV 导出
- 定时任务稳定运行 (连续 7 天无崩溃)
- 查询失败自动记录错误信息

8.2 性能要求

- API 响应时间 $< 500\text{ms}$ (不含查询执行)
- 页面加载时间 $< 3\text{s}$
- 单次 AI 平台查询 $< 90\text{s}$ (含超时重试)
- 数据库查询 $< 100\text{ms}$

8.3 已知限制（MVP 阶段可接受）

- 仅支持文心一言 + Kimi（后续再加其他平台）
- 引用检测基于文本匹配，不做深度语义理解
- 查询频率最低每周 1 次
- 不支持团队协作（单用户）
- 仅支持微信支付占位（MVP 阶段可跳过实际支付）