Interview Helper — Architecture & Implementation Plan

This document maps the full system architecture, data flow, API endpoints, prompts, deployment, and an actionable build checklist for the Interview Helper MVP.

---

1. High-level architecture (text + ASCII diagram)

Frontend (Next.js + Tailwind), Backend (FastAPI) — AI Service Layer — Data Fetchers (Clearbit, GitHub, NewsAPI, Scrapers) Database (Postgres) Cache (Redis) Storage (S3-compatible for PDFs) Auth (NextAuth / Supabase Auth)

```
[User Browser] --> [Next.js Frontend]
       |                  |
       |                  v
       |            API Routes (optional)
       v                  |
[Auth Service] <----> [Backend API (FastAPI)] <----> [AI Service / LLM API]
                               |   ^   \
                               |   |    -> [Cache Redis]
                               v   |
                    [Data Fetcher Worker / Scraper / 3rd-party APIs]
                               |
        -------------------------------------------------------------
        |                     |                 |                  |
   [Clearbit API]       [GitHub API]     [NewsAPI / RSS]    [Website Scraper]

                           |
                           v
                     [Postgres DB]
                           |
                   [S3 bucket for PDFs]
```

---

2. Components & Responsibilities

- **Next.js Frontend**
- Pages: Landing, Login, Dashboard, New Prep, Results, Resources, Settings
- Uses Tailwind & Headless UI for layout & components

- Calls Backend REST endpoints; uses optimistic UI for fast feedback
- **Auth Service**
- NextAuth.js (hosted in Next.js) or Supabase Auth
- Issue JWTs or session cookies for backend calls
- **Backend API (FastAPI)**
- Endpoints: `/api/prep` (POST), `/api/sessions` (GET/POST), `/api/session/{id}` (GET), `/api/user/settings` (GET/PUT), `/api/pdf/{id}` (GET)
- Responsibilities: validate input, orchestrate fetchers, call AI Service, store session
- **Data Fetchers (Worker)**
- Worker process (Celery/RQ or FastAPI background tasks)
- Plugins: Clearbit client, GitHub client, News client, HTML scraper using `httpx` + `BeautifulSoup` or `trafilatura`
- Respect `robots.txt` and rate limits; add exponential backoff
- **AI Service Layer**
- Prompt templating, input sanitization, and post-processing
- Interfaces to OpenAI/Anthropic/HF or local LLMs
- Contains heuristics to limit hallucinations (source quoting, confidence scores)
- **Database & Cache**
- Postgres for user data, saved sessions, and fetched summaries
- Redis for short-term caching of fetched company/interviewer metadata and rate-limiting
- **Storage**
- S3-compatible storage for generated PDFs and large assets
- **Monitoring & Logging**
- Sentry for errors, Prometheus/Grafana for infra metrics, structured logs in Logtail

---

3. API Contract (backend endpoints)

`POST /api/prep`

Request body (JSON):

```
{
  "user_id": "uuid",
  "company_url": "https://example.com",
  "interviewer_name": "John Doe",
  "socials": { "github": "https://github.com/johndoe", "linkedin": "..." },
  "role_focus": "DSA+Backend",
  "timeframe": "7 days",
  "skill_level": "intermediate",
  "notes": "focus on system design"
}
```

Response: 202 Accepted (queued) with `session_id`. Frontend polls `/api/session/{id}` or uses websocket for progress.

`GET /api/session/{id}`

- Returns session status: `queued`, `fetching`, `processing`, `done`, `failed` and result payload when `done`.

`GET /api/sessions` (user) — paginated list of saved sessions

`POST /api/session/{id}/export-pdf` — trigger pdf build

---

4. Data Fetching Strategy (detailed)

1. **Initial cache check** (Redis): `company_domain:hash` and `interviewer_handle:hash`
2. If cached & fresh (e.g., <24h), return cached summary
3. Else trigger worker to:
4. Call Clearbit (if API key) for canonical company metadata
5. Fetch company's `/about`, `/careers`, `/blog` pages; extract sections with `trafilatura` or `readability` to avoid nav/footer
6. Call NewsAPI for recent mentions (limit 5)
7. If GitHub provided, call GitHub API to fetch public repos, top languages, recent activity
8. (Optional) If LinkedIn uploaded or provided as file, parse that
9. Clean results: remove boilerplate, extract bullet points (mission, products, tech tags)
10. Store cleaned data in Postgres & Redis

Respect rate limits: add per-domain and per-API backoff + queue

5. Prompt Templates & Safety

Top-level prompt (example)

```
You are an interview coach. Using the following structured input, generate a
personalized interview prep plan.

Company summary:
{{company_summary}}

Interviewer summary:
{{interviewer_summary}}

Role focus: {{role_focus}}
Timeframe: {{timeframe}}
Skill level: {{skill_level}}

Produce:
1) Two-paragraph company overview
2) Top 8 technical topics likely to be asked
3) A day-by-day action plan with resources (links allowed)
4) Quick behavioral bullets relevant to company culture
5) "Confidence" note: list sources used

Be concise, do not hallucinate, and when uncertain, say so.
```

Post-processing rules

- If AI asserts a fact about the interviewer/company, append `— Source: [URL]` when the fact originates from a fetched page.
- If confidence low, show a banner: "Verify these facts before relying on them."

6. Data Models (Postgres simplified)

- `users` (id, email, name, provider, created_at)
- `sessions` (id, user_id, input_json, status, result_json, created_at, updated_at)
- `companies` (id, domain, canonical_name, summary, meta_json, last_fetched)
- `interviewers` (id, name_or_handle, github_json, linkedin_blob, summary, last_fetched)
- `resources` (id, session_id, url, title, type)

7. Sequence (detailed) — what happens when user clicks "Generate"

1. Frontend POST `/api/prep` with validated input
2. Backend creates `sessions` row with status `queued` and returns `session_id`
3. Worker picks up job from queue
4. Worker: check caches -> fetch company -> fetch interviewer -> clean & store
5. Worker: assemble `ai_input` and call AI service (with retry/backoff)
6. Worker: receive AI output -> post-process (linkify, add source list), store in `sessions.result_json`, set status `done`
7. Frontend polls `/api/session/{id}` and renders output when `done`

---

8. Caching, Rate Limits & Cost Control

- Cache company/interviewer data for 24 hours (configurable)
- Rate-limit fetch per domain and per API key (use Redis counters)
- Limit AI token usage by summarizing long docs locally before sending
- Use a tiered AI strategy: short plan -> cheap model, deep plan -> higher-cost model

---

9. Error Handling & UX

- Long-running task UX: show progress states and approximate steps (Fetching data → Processing → Finalizing)
- On partial failures (e.g., cannot fetch LinkedIn), show partial results with clear warnings
- Retries: 3 attempts with exponential backoff for fetches and AI calls

---

10. Security & Privacy

- Always serve over HTTPS
- Encrypt sensitive fields at rest where required
- Provide a "Delete my data" endpoint & UI
- Rate-limit per-user to prevent abuse
- Store minimal PII: encourage users to upload only public info or provide links

---

11. Deployment & Infra Recommendations

- **Frontend**: Vercel (Next.js)
- **Backend**: Render / Railway / Fly.io (FastAPI + worker) or AWS ECS
- **DB**: Neon / Supabase / Railway Postgres
- **Cache**: Redis (Upstash / Redis Cloud)

- **Storage**: S3 (AWS) or DigitalOcean Spaces
- **Monitoring**: Sentry + Logtail + Prometheus

---

12. Roadmap & Milestones (MVP -> v1)

- **MVP (Weeks 1-4)**

- Auth, Next.js skeleton, FastAPI skeleton

- `POST /api/prep` -> queue -> worker -> call a simple AI prompt using dummy data

- Render a basic results page

- **v0.5 (Weeks 5-8)**

- Add real data fetchers: Clearbit, GitHub, NewsAPI

- Add caching and PDF export

- **v1 (Weeks 9-12)**

- Improve prompts, source quoting, partial result UI, rate limiting

- Add billing & premium features

---

13. Implementation Checklist (concrete tasks)

-

If you want, I can now:

- produce a **sequence diagram** image or Mermeid.js diagram for the flow,
- scaffold the initial **FastAPI** server with the `/api/prep` endpoint and a dummy worker,
- or lay out the **Next.js pages** in a single-file prototype.

Tell me which one to generate next and I'll do it.