Interview Practice Lab — Interview Intelligence Enhancements + Seed Data Plan

Developer-facing documentation to upgrade the existing Avatar Interview Practice Lab into a realistic, data-backed Interview Prep product with stickiness (job tracking + memory + personalized coaching).

Generated: 2026-01-04

1) What we have today (baseline)

The current codebase already includes dedicated interview endpoints and “exercise mode” for case studies + coding challenges. The key tables in the existing interview track are: role_kits, interview_rubrics, user_documents, user_profile_extracted, interview_configs, interview_plans, interview_sessions, interview_analysis (plus exercise tables for coding/case).

This plan focuses on:

- Upgrading Interview Intelligence (plans, question logic, probing, evaluation).
- Making data real (company-backed patterns for tech via curated/scraped sources; archetype-based for non-tech).
- Adding sticky workflows (saved jobs → practice → track → AI learns).
- Updating dashboard/results to match interview outcomes.

2) Product positioning recommendation (for v1)

To stay competitive and ship faster, keep role coverage broad, but focus positioning on ONE promise: “job-linked practice” — users practice for specific saved JDs (not generic questions). This differentiates vs generic interview simulators.

- Primary users: college grads + early-career (0–5 yrs), across tech + business roles.
- Primary loop: Save job → practice interview + exercise → get actionable feedback + plan → re-practice → track readiness.
- Secondary wedge: tech companies + company-specific patterns for top 50 (via curated/scraped experiences + coding sets).

3) Interview Intelligence: the engine (how realism is achieved)

3.1 Inputs (what the user uploads/selects)

- Resume/CV: extract claim triggers (projects, skills, tenure, impact metrics, gaps, transitions).
- Job Description: extract responsibilities, must-have skills, nice-to-have, seniority signals.

- Company selection: choose company (optional) OR choose company archetype (startup/mid/enterprise; consumer/fintech/saas; regulated).
- Interview profile: interview type (HR / HM / Technical / Panel), style (friendly/neutral/stress), seniority (entry/mid/senior).

3.2 Heuristic rules (universal interviewer behavior)

- Validate claims → ask for proof, numbers, scope, trade-offs.
- Probe depth → if answer is vague, drill down into specifics and decision rationale.
- Consistency checks → cross-check answers vs resume timeline and stated skills.
- Risk stress-test → gaps, short tenures, over-claiming, mismatch vs JD.
- Fit assessment → how candidate will operate in THIS environment (archetype + role expectations).

3.3 Question patterns (store patterns, not question banks)

We store templated question patterns plus probe trees. Runtime fills them using: (a) extracted resume triggers, (b) JD requirements, (c) company/archetype, (d) candidate answers.

```
Pattern: Resume Claim → Depth
Q0: "You mentioned {{PROJECT/CLAIM}}. Walk me through the context and your specific
role."
Probe1: "What was the hardest part and what trade-off did you choose?"
Probe2: "How did you measure impact? What metric moved?"
Probe3: "If you redo it today, what would you change and why?"
```

3.4 Evaluation rubrics (signals + level calibration)

Use an 8–10 dimension rubric with explicit behavioral anchors by level (entry/mid/senior). The uploaded behavioral evaluation reference already uses level-calibrated examples (junior/senior/staff) — replicate that style into rubric anchors.

Recommended base dimensions (interview types add/remove dimensions):

- Communication clarity
- Structured thinking
- Depth/ownership (how much they actually drove)
- Problem solving / reasoning
- Execution details (tools, steps, verification)
- Collaboration / conflict handling
- Ownership / initiative
- Learning / growth mindset
- Role fit (JD match + gaps)
- Confidence / presence (delivery signals)

4) New features to add (beyond conversation)

4.1 Case Study Mode (Interview Exercise Mode) — upgrade

You already have case_templates + exercise_sessions + analysis tables. Enhancements needed:

- Case templates must support staged reveals (candidate asks clarifying questions → system reveals datasets).
- Add "interviewer scoring checkpoints" per phase (framing → hypotheses → analysis → recommendation → risks).
- Add "case artifacts": notes scratchpad, structured answer outline, final recommendation summary.

Your existing case_templates.csv already includes revealable_data and probing_map fields — keep, but standardize JSON format (see section 8 seed data).

4.2 Coding Debug/Modify/Explain Mode (live coding without full IDE)

Goal: simulate a technical round where the interviewer gives a buggy snippet / failing test case / modification request.

You already have a coding_exercises table with fields like code_snippet, bug_description, failing_test_case, modification_requirement, expected_behavior, expected_signals, suggested_fix, probing_questions, etc. (from the uploaded coding_exercises.csv).

- UX: show code snippet + failing test + prompt; user answers verbally and/or types patch (MVP: type patch in textarea).
- LLM judging: compare user patch vs expected_behavior + edge_cases; generate feedback on correctness + reasoning.
- Add "explain your reasoning" scoring: can they justify trade-offs, complexity, and edge cases.
- Optional: add "code review mode" (interviewer asks: 'what would you change in this PR?').

4.3 Sticky loop: Job Tracker + Practice Memory

This is the moat: the app becomes the place where their job search + readiness lives.

- Save jobs: user stores job URL/JD, company, role, status (saved/applied/interviewing/offer/rejected), deadlines.
- One-click practice: from a saved job, generate interview plan + exercises tailored to that JD + role kit.
- Progress memory: track recurring weaknesses, repeated missed signals, improvement trend by dimension.
- AI Coach plan: weekly practice plan + 'next best session' recommendation based on gaps.

5) Data strategy (real world signal without copying question banks)

5.1 Tech interviews: curated + scraped “experience posts” → extracted patterns

For top tech companies, the best realism comes from aggregating interview experiences and mapping them into: skill_focus, question_patterns, probes, and rubrics.

- Use your Apify LeetCode interview experience scraper (company + maxToScrape + startDate). Store raw posts, then extract normalized patterns.
- Use your existing ‘Top Leetcode questions’ Excel as a seed set for coding prompts by topic + level + companies mentioned.
- Optionally add curated public sources (blogs, interview handbooks) — but store as patterns and signals, not copied lists.

5.2 Non-tech roles: role + company archetype + JD → generated question patterns

For business/marketing/sales/ops roles, company-specific data is noisy; use archetypes + JD to generate realistic scenarios and probe trees.

- Company archetypes: startup execution-heavy; enterprise process-heavy; regulated (banks/pharma) risk/compliance; high-growth sales org; etc.
- Role archetypes: IC vs manager, stakeholder-heavy vs execution-heavy.
- Case templates: market sizing, pricing, growth experiment, sales objection handling, ops process improvement.

6) Concrete DB schema diff (additions)

Below is a proposed schema diff. Keep existing interview tables; add these to support job tracking + data ingestion + intelligence patterns.

```
-- NEW: companies (optional; can start as a seed list)
companies (
  id uuid pk,
  name text unique,
  country text,
  tier text, -- top50/top200/other
  archetype text, -- startup/enterprise/regulated/consumer/saas/fintech...
  tags jsonb,
  created_at timestamptz
);

-- NEW: saved_jobs (stickiness loop)
saved_jobs (
  id uuid pk,
  user_id uuid fk auth_users,
  company_id uuid fk companies null,
  company_name text, -- fallback if company_id null
  role_title text,
  role_category text, -- 'swe','data','pm','sales','marketing','ops',...
  job_url text,
  jd_text text, -- if scraped / pasted
  jd_document_id uuid fk user_documents null,
  status text, -- saved/applied/interviewing/offer/rejected
  next_interview_at timestamptz null,
  notes text null,
  created_at timestamptz,
  updated_at timestamptz
);

-- NEW: job_practice_links (many sessions per job)
job_practice_links (
  id uuid pk,
  saved_job_id uuid fk saved_jobs,
  interview_config_id uuid fk interview_configs,
  interview_session_id uuid fk interview_sessions,
  exercise_session_id uuid fk exercise_sessions null,
  created_at timestamptz
);

-- NEW: question_pattern_library (core IP)
question_patterns (
  id uuid pk,
  pattern_type text, -- resume_claim/jd_requirement/behavioral/scenario/probe
  role_category text,
  interview_type text,
  template text, -- mustache style variables
  probe_tree jsonb, -- follow-up logic
  tags jsonb,
  source_type text, -- curated/generated/scraped
  source_ref text null, -- url or dataset id
  created_at timestamptz
);

-- NEW: company_role_blueprints (company-ish layer)
company_role_blueprints (
  id uuid pk,
  company_id uuid fk companies,
  role_category text,
  seniority text,
  skill_focus jsonb,
  question_pattern_ids jsonb,
  rubric_overrides jsonb null,
  updated_at timestamptz
);

-- NEW: scraped_sources + raw posts (for Apify runs)
data_source_runs (
  id uuid pk,
  source text, -- apify_leetcode/glassdoor/reddit/etc
  run_params jsonb,
  status text,
  started_at timestamptz,
  finished_at timestamptz,
  item_count int,
  storage_ref text -- dataset id/url
);

scraped_interview_posts (
  id uuid pk,
  source_run_id uuid fk data_source_runs,
  company_name text,
  tags text,
  created_at_source timestamptz,
  upvotes int,
  raw_content text, -- store raw post (for internal processing)
  extracted jsonb, -- normalized extracted items
  created_at timestamptz
);

-- NEW: user_skill_memory (personalization)
user_skill_memory (
  id uuid pk,
  user_id uuid fk auth_users,
  role_category text,
  dimension text,
  baseline_score numeric,
  latest_score numeric,
  trend jsonb, -- last N sessions
  common_issues jsonb,
  updated_at timestamptz
);
```

7) Code logic upgrades (backend + prompts)

7.1 Document ingestion + extraction

- On /api/interview/documents/upload: after storing resume/JD, run an extraction job → user_profile_extracted.
- Extraction outputs should be structured JSON: work_history[], projects[], skills[], metrics[], gaps[], claim_triggers[].
- For JD: responsibilities[], must_have[], nice_to_have[], keywords[], seniority_signal[].

7.2 Plan generation (interview_plans)

Plan generation should combine: role_kit defaults + user-specific inputs (resume/JD) + company blueprint (if any) + heuristics.

```
interview_plan JSON (recommended shape)
{
  "metadata": { "roleCategory":"swe", "seniority":"entry", "interviewType":"technical",
"company":"Google" },
  "phases": [
    { "id":"intro", "mins":3, "objective":["rapport","role-context"],
"patterns":["intro_basic"] },
    { "id":"resume_deep_dive", "mins":10, "objective":["validate-claims"],
"triggers":["project","internship"], "patterns":["resume_claim_depth_v1"] },
    { "id":"coding", "mins":25, "objective":["problem-solving","coding"],
"exerciseMode":"coding_debug", "topics":["arrays","hashmap"],
"patterns":["coding_prompt_v1"] },
    { "id":"behavioral", "mins":10, "objective":["values-fit"],
"patterns":["conflict","ownership","growth"] },
    { "id":"close", "mins":2, "objective":["questions-from-candidate"],
"patterns":["candidate_questions"] }
  ],
  "globalRules": {
    "probeDepth": 3,
    "avoidRepeats": true,
    "beStrictOnVagueness": true
  }
}
```

7.3 Runtime interviewer (conversation agent)

Runtime interviewer should behave like a real interviewer: short prompts, time-boxed, and always probing when answers are vague.

Implement a small “controller” that decides next question based on phase + candidate answer signals.

- Phase controller: picks next pattern, fills variables, asks question.
- Answer classifier: tags answer as strong/weak/vague, extracts claims/metrics, detects missing pieces.
- Probe selector: chooses follow-ups from probe_tree based on what’s missing.
- Time management: if time running out, skip to close phase.

7.4 Post-session analysis (interview_analysis)

Replace generic communication-only analysis with interview-specific outputs: hire signals, gaps vs JD, and next practice actions.

```
analysis JSON (recommended outputs)
{
  "overall": { "score": 72, "readiness":"borderline", "hireSignal":"weak_hire" },
  "dimensionScores": [
    { "key":"communication", "score":75, "evidence":[...], "fixes":[...] },
    { "key":"problem_solving", "score":68, "evidence":[...], "fixes":[...] },
    { "key":"role_fit", "score":62, "gaps":[...], "fixes":[...] }
  ],
  "jdMatch": { "matchedSkills":[...], "missingSkills":[...], "riskFlags":[...] },
  "answerRewrites": [ { "question":"...", "yourAnswer":"...", "betterAnswer":"...",
"whyBetter":"..." } ],
  "practicePlan7d": [ { "day":1, "focus":"SQL joins", "session":"coding_debug",
"exerciseId":"..." }, ... ],
  "memoryUpdate": { "commonIssues":[...], "newStrengths":[...], "nextBestAction":"..."
}
}
```

8) Initial seed data plan (what to ship with)

8.1 What is already seeded (from your uploaded CSVs)

role_kits.csv: 15 rows with columns: id, name, level, domain, description, default_interview_types, default_rubric_id, skills_focus, estimated_duration, track_tags, is_active, created_at, updated_at, role_context, core_competencies, typical_questions.

coding_exercises.csv: 8 rows with columns: id, role_kit_id, name, activity_type, language, difficulty, code_snippet, bug_description, failing_test_case, modification_requirement, expected_behavior, expected_signals, common_failure_modes, suggested_fix, edge_cases, complexity_expected, probing_questions, tags, is_active, created_at, updated_at.

case_templates.csv: 14 rows with columns: id, role_kit_id, name, case_type, difficulty, prompt_statement, context, clarifying_questions_allowed, revealable_data, evaluation_focus, probing_map, expected_duration_minutes, tags, is_active, created_at, updated_at.

interview_rubrics.csv: 4 rows with columns: id, name, description, dimensions, scoring_guide, is_default, created_at, updated_at.

Top LeetCode Excel (cleaned): 322 problems; sample topics include Array (and companies embedded in c1..c5).

8.2 Seed data we should add (minimum viable)

- Companies: 50 (25 India + 25 global) as a seed list with archetypes and tags.
- Company-role blueprints: for top ~10 global tech companies initially (Google, Amazon, Microsoft, Meta, Apple, Netflix, Uber, Stripe, Airbnb, LinkedIn), cover SWE + Data roles.
- Question patterns library: ~120 patterns total (resume/JD/behavioral/scenario/probes).
- Case templates: 15–20 strong cases across business roles (growth, marketing, ops, sales), each with staged reveals.
- Coding exercises: map 60–100 coding problems to “activities” (explain → debug → modify) using your LeetCode sheet and/or scraped company trends.

8.3 Mapping the “roles list” UI (your screenshot) to role_kits

Your role selection UI contains many role labels (PM, SWE, TPM, Data Engineer, Data Scientist, etc.). For v1, keep role_kits as the canonical catalog; add tags to map multiple UI labels to one role category.

- Example mapping: Backend Engineer + SWE + Frontend Engineer + Mobile Engineer → role_category='swe' with specializations in track_tags.
- Security roles (Security Engineer, IAM Analyst, GRC Analyst, Penetration Tester, CTI) → role_category='security'.
- Quant roles (Quant Trader, Quant Researcher) → role_category='quant' (later).

9) Ingestion pipeline for Apify + external sources (developer plan)

9.1 Safe approach (recommended)

- Start with Apify actor outputs you already have access to (LeetCode interview experience scraper). Store raw posts; do NOT directly show verbatim posts in product UI.
- Run an internal extractor to convert posts into patterns and skill_focus (normalize).
- Keep a “source run” record for traceability and debugging.

9.2 Proposed extractor (LLM + rules)

```
Extractor output per post
{
  "company":"Google",
  "rolesMentioned":["SWE","L3"],
  "rounds":[ {"type":"coding","topics":["arrays","hashmap"],"notes":"..."},
{"type":"behavioral","topics":["conflict","ownership"]} ],
  "questionSignals":[ {"pattern":"resume_claim_depth_v1","evidence":"Asked about
internship project depth"} ],
  "rubricSignals":[ {"dimension":"problem_solving","signal":"needs to articulate
tradeoffs"} ]
}
```

10) Dashboard + Results redesign (must-do changes)

Current dashboard/results are scenario-centric. Interview app needs job-centric progress and readiness tracking.

- Dashboard: show Saved Jobs (next interview date), Readiness score per job, streaks, and “next best practice”.
- Results list: filter by job, role, company, interview type, exercise type (case/coding/interview).
- Result detail: include JD-match, hire signals, and a practice plan; link to recommended next session/exercise.
- Progress: dimension trends over time; heatmap by dimension vs interview type.

- Memory: recurring weaknesses and ‘fixed’ weaknesses (so users feel progress).

11) API changes (new endpoints)

- POST /api/jobs/save (url + paste JD + company + role)
- GET /api/jobs (list saved jobs + status)
- PUT /api/jobs/:id (update status, add notes, set interview date)
- POST /api/jobs/:id/generate-plan (creates interview_config + interview_plan bound to saved job)
- GET /api/patterns (admin) / POST /api/patterns (admin)
- POST /api/data-sources/apify/run (admin) (optional: triggers run via apify; or ingest uploaded dataset)
- POST /api/data-sources/apify/ingest (admin) (writes scraped_interview_posts)

12) System prompts (exact templates for developer)

Keep prompts short, deterministic, and role-calibrated. Use a 3-agent setup: Interviewer, Evaluator, Coach.

12.1 Interviewer system prompt (template)

```
SYSTEM (Interviewer)
You are a professional interviewer conducting a {{interviewType}} interview for the
role {{roleTitle}} ({{seniority}}).
You must:
- Ask one question at a time.
- Keep questions concise (1–2 sentences).
- Probe for specifics when answers are vague: context, actions, impact, tradeoffs,
metrics.
- Cross-check against resume facts and JD requirements; call out inconsistencies
politely.
- Manage time: follow the interview_plan phases and move on when time is low.
- Maintain style: {{style}} (friendly / neutral / stress) without being rude.
You have access to:
- resume_extract JSON
- jd_extract JSON
- company_archetype JSON
- interview_plan JSON
Do NOT reveal the plan; do NOT give feedback during the interview unless asked.
```

12.2 Evaluator system prompt (post-session)

```
SYSTEM (Evaluator)
You are evaluating an interview transcript for {{roleTitle}} at {{companyOrArchetype}}.
Return STRICT JSON only matching this schema:
{
  "overall": { "score":0-100, "readiness":"not_ready|borderline|ready",
"hireSignal":"strong_hire|hire|lean_hire|lean_no_hire|no_hire" },
  "dimensionScores": [ { "key":"...", "score":0-100,
"evidence":[{"quote":"<short>","why":"..."}], "fixes":[{"title":"...","how":"..."}] }
],
  "jdMatch": { "matchedSkills":[...], "missingSkills":[...], "riskFlags":[...],
"priorityGaps":[...] },
  "answerRewrites": [ { "question":"...", "yourAnswer":"...", "betterAnswer":"...",
"whyBetter":"..." } ],
  "nextActions": [ { "type":"practice|exercise|resume_fix", "title":"...",
"details":"...", "estimatedMinutes":15 } ],
  "memoryUpdate": { "newRecurringIssues":[...], "resolvedIssues":[...],
"recommendedFocusNext":"..." }
}
Rules:
- Use only transcript + extracts; no hallucinated facts.
- Evidence quotes must be <= 25 words.
```

12.3 Coach system prompt (weekly plan + stickiness)

```
SYSTEM (Coach)
You are an interview coach. Use the evaluator JSON + user's last 5 sessions + saved
jobs list.
Goal: create a 7-day plan that mixes:
- 2 interview simulations
- 2 coding/debug activities (if tech)
- 1 case study (if relevant)
- 2 micro drills (short answers, storytelling, metrics)
Return JSON:
{ "plan":[{ "day":1, "goal":"...", "items":[...]}], "streakGoal":5, "whyThisPlan":"..."
}
```

Appendix A) Sample seed snippets (ready for dev seeding)

A1) Example company archetypes (insert into companies)

```
India seeds (examples): TCS (enterprise-services), Infosys, Wipro, HCL, Tech Mahindra,
Flipkart (consumer-tech), Zomato, Paytm (fintech), Razorpay (fintech), Swiggy, Ola,
BYJU'S (edtech), PhonePe, Freshworks (saas), MakeMyTrip, Tata Steel (industrial), ICICI
Bank (regulated), HDFC Bank (regulated), Reliance Jio (telecom), L&T (industrial), etc.

Global seeds (examples): Google, Amazon, Microsoft, Meta, Apple, Netflix, Uber, Airbnb,
Stripe, LinkedIn, Salesforce, Adobe, IBM, Oracle, JPMorgan Chase (regulated), Goldman
Sachs (regulated), Walmart, Shopify, Atlassian, Bloomberg, etc.
```

A2) Example question patterns (insert into question_patterns)

```
{
  "pattern_type":"resume_claim",
  "role_category":"swe",
  "interview_type":"technical",
  "template":"You mentioned {{project_name}}. What was the hardest bug or failure you
hit, and how did you debug it?",
  "probe_tree": {
    "if_vague":[
      "What was the exact symptom?",
      "What hypotheses did you form first?",
      "What did you try that failed?"
    ],
    "always":[
      "What was your specific contribution vs the team?",
      "How did you verify the fix and prevent regression?"
    ]
  },
  "tags":["debugging","ownership","verification"],
  "source_type":"curated"
}
```