Scenario For Web Development

https://brtneura.com

Copyright Information

Document Control

Term	Details
Document Title	Candidates Exam Scenario
Author	BRTNeura
Version Number	
Date Created	
Date **Submitted**	

Index

1. Executive Summary

What you'll do:

You'll build a small **full-stack web app** based on **one scenario** (assigned to you). It must include:

- **Backend:** Python **FastAPI** (Pydantic, Uvicorn)
- **Frontend: React + TypeScript** (Vite)
- **Auth: Firebase Auth** (server verifies tokens)
- **DB:** As specified in your scenario (**Firestore / Postgres / SQLite**)
- **AI/Analytics:** A small, **simple** component (e.g., heuristic or basic model) with a quick evaluation
- **Tests:** ≥ **3 backend** (pytest) + ≥ **1 frontend** (React Testing Library)
- **Docs & DevEx:** README with setup, **OpenAPI** at /docs (or Postman), architecture diagram, .env.example, and a **≤5-min Loom** walkthrough

Timebox: 6–8 hours.
Pass bar: Score ≥ **70/100** on the common rubric. **Top tier:** 85+

What to submit (deliverables checklist)

- Repo with two folders: /api (FastAPI) and /web (React)
- README.md with:
  - Setup steps & run commands (dev, test, seed)
  - **Architecture diagram** (Mermaid or PNG)
  - Endpoints + sample requests/responses
  - **ADR** (why you chose your datastore/indices)
  - **"How I used Cursor"** (3–5 lines)
- **OpenAPI** at /docs (or a Postman/Thunder collection)
- **Seed script** to load sample data (≥1k rows where relevant)
- **Tests** (all pass with one command)
- **Deploy links** (Netlify/Vercel for FE + Render/Fly/Cloud Run for API) **or** Docker Compose
- **Loom video** (≤5 min) demoing the core flow

2. Scenario

Task Runner & Webhook Orchestrator (Postgres)

You are a developer who needs to develop this website whose function is to let users define HTTP webhooks to run on schedules and view run logs with retries and a dead-letter queue.

Must-use tech:
FastAPI, React+TS, Firebase Auth, Postgres, PyTest, background worker.

Schema:
tasks(id, user_id, url, method, headers_encrypted, body, schedule_cron, enabled)
runs(id, task_id, status, latency_ms, response_code, error, created_at)
dlq(id, task_id, error, created_at)

API:

- POST /tasks GET /tasks GET /tasks/{id}/runs
- Worker executes tasks, retries with exponential backoff up to 3, then DLQ.

Frontend:
Task composer, run history table with filters, latency histogram.

AI:
Failure classifier (network/auth/payload) by simple rules + explanation string.

Security:
Encrypt headers server-side; mask in UI.

Tests:
(1) Retry/backoff sequence, (2) DLQ path, (3) Secret masking not leaked.

3. Evaluation Criteria

TOTAL Score: 100

1) Functional completeness (15)

- **Core user flows work** end-to-end as specified (create/read/update, scenario-specific flows). (0–8)
- **Edge cases handled** (empty state, invalid input, not found). (0–4)
- **Polish** (pagination, meaningful errors/toasts). (0–3)

2) Architecture & data modeling (10)

- **Sound schema/collections** with correct keys, indexes, relations. (0–5)
- **Separation of concerns** (routers/services/models/utils). (0–3)
- **Trade-offs explained** in ADR (why SQL vs Firestore, why chosen indexes). (0–2)

3) Backend API quality (15)

- **Contract** matches scenario (routes, verbs, status codes, validation). (0–6)
- **Performance awareness** (pagination server-side, N+1 avoided, basic indexing). (0–5)
- **Resilience** (timeouts, retries/background jobs where required). (0–4)

4) Frontend implementation (10)

- **State management** is reasonable (React Query/contexts/hooks). (0–4)
- **Forms & UX** (validation, loading/error states, accessible components). (0–4)
- **Integration** with backend/auth done correctly. (0–2)

5) Auth & security (10)

- **Firebase Auth implemented**; tokens verified **server-side**. (0–4)
- **Authorization** (role checks, multi-tenant scoping where applicable). (0–4)
- **Secrets & inputs** (no secrets in client, input sanitization, rate limits if required). (0–2)

6) AI/analytics piece (10)

- **Exists and is relevant** to the scenario (not a stub). (0–4)
- **Basic evaluation** (e.g., confusion matrix, precision@k, z-score thresholds, or documented assumptions). (0–4)
- **Cost/complexity discipline** (simple, explainable approach; no paid APIs). (0–2)

7) Performance & reliability (10)

- **Seeded data size** (≥1k rows/items where relevant). (0–3)
- **Latency** (P50 ≤ 300 ms on core list endpoints locally—documented). (0–3)

- **Background work** (schedulers/workers behave as specified; retries/backoff). (0–4)

8) Testing (10)

- **Backend tests (≥3)** cover a happy path and at least one failure path. (0–6)
- **Frontend test (≥1)** covers a key interaction (filter, submit, or route). (0–3)
- **Tests pass** via a single command. (0–1)

9) Code quality & DevEx (5)

- **Readable code** (names, small functions, lint/format). (0–3)
- **DX scripts** (dev, test, seed, or Makefile). (0–2)

10) Documentation & communication (5)

- **README**: setup, run, env, architecture diagram, endpoints, **"How I used Cursor"** note. (0–4)
- **Postman/OpenAPI** or collection linked; short Loom ≤5 min. (0–1)

Bonus & penalties (±10)

Bonus (+1–10):

- Stretch feature shipped well (+2–4)
- Observability (health, structured logs, basic tracing) (+1–2)
- Clean migrations/data seeds for SQL; Firestore rules written (+1–2)
- Thoughtful UX touches (keyboard nav, a11y) (+1–2)

Penalties (−1–10):

- Hard-coded secrets or client-side secrets (−5)
- Auth missing on server (−5)
- Can't run from README (−3)
- Heavy copy-paste/boilerplate without integration (−2)
- Fails scenario contract (wrong routes/DB) (−3)

Auto-reject conditions

- No server-side auth/authorization when required.
- App cannot be started following README.
- Plagiarism or inability to explain code during follow-up.

- Data exposure (e.g., cross-tenant leaks).

4. Tips to Pass (and score 85+)

1) Land the core user flow early

- Implement **one happy path** E2E in the first 2–3 hours (auth → create → list → detail).
- Defer fancy UI until after the flow works.

2) Keep AI simple but *real*

- Use **rules + a tiny model** (e.g., logistic regression, TF-IDF + cosine).
- Show **one tiny metric** (confusion matrix, precision@k, or threshold rationale).
- No paid APIs needed.

3) Prove server-side auth

- Validate Firebase ID token **in FastAPI** (every protected route).
- If multi-tenant: enforce org_id on the server for every query.

4) Nail pagination & indexing

- **Server-side pagination** (limit + cursor/offset) is required on "list" endpoints.
- Add 1–2 **indexes** that actually speed up your list queries; note them in the README.

5) Ship tests that matter

- Backend: (a) happy path, (b) **one failure path** (e.g., validation/rate-limit), (c) a behavior tied to your scenario (e.g., dedupe, scheduler).
- Frontend: test a **key interaction** (filtering, submit, or rendering a list).

6) Be reliable under load (seed data)

- Seed **≥1k rows** (transactions, leads, products… as relevant).
- Paste **measured p50 latency** (curl or simple timer) into README.

7) Keep secrets secret

- Provide **.env.example**; never hardcode keys; never put secrets in the client bundle.

8) Background jobs: make them observable

- Add a **health endpoint** and log each run with status/latency (for schedulers/workers).
- Document retry/backoff behavior if your scenario calls for it.

9) Communicate like a pro

- Short, accurate **README**; clear limitations; “what I’d do next.”
- Loom shows: auth → core flow → the AI/analytics bit → tests passing (10–15 sec each).

10) Use Cursor wisely (and say how)

- Show *where* you used it (e.g., stub generation, test scaffolding) and *how* you verified output.
- Don’t accept code blindly—commit in **small diffs** with messages.

5. Suggested 6–8h game plan

1. **Setup (45 min):** Repos, auth stub, DB schema/collections, routes/components.
2. **Core flow (2–3h):** E2E path working + server-side pagination.
3. **AI/analytics (60–90 min):** Implement simplest viable logic + one metric.
4. **Polish + perf (60 min):** Seed ≥1k rows; add indexes; measure p50; fix logs.
5. **Tests (45–60 min):** 3 backend + 1 frontend, all green.
6. **Docs & demo (30–45 min):** README, OpenAPI, Loom.

6. Common pitfalls (avoid!)

- Client-only auth (no server verification)
- No pagination; “load all” queries
- Hardcoded secrets; missing .env.example
- Can’t run from README; missing scripts
- AI part is a placeholder (no metric, no effect on UI)
- No seed data → you can’t show performance