

**Activity Finder App Assessment**

V1.0  
*10.14.2025*

**“Local Activity Finder”**

### **Objective**

This role requires someone who can perform at a high level first, as true leadership, teaching, and evaluation of others depend on proven ability and mastery in execution.

This assessment is designed to evaluate a candidate who can balance technical execution and strategic direction. Someone who can architect, prototype, and document a system in a way that allows an external or downstream team to confidently pick it up and continue.

The focus is not on how much code is written, but on the quality of decisions, clarity of reasoning, and readiness for handoff. The prototype should demonstrate technical competence and architectural foresight, while the documentation should reflect the candidate’s ability to communicate how and why the system was built without filler or fluff.

The expectation is that, if the author stepped away after submission, another competent team could build from their foundation with minimal friction.

Build a small product prototype that helps a user:

1. choose or detect a location,
2. see current conditions and a short forecast, and
3. get nearby activities/points of interest that make sense given the weather.

The prototype must be production-minded: clear architecture, tight docs, and ready to hand off to a downstream team for final implementation.

## **Functional Scope (must implement these features)**

* **Location input**: user can search a place name or use current location.
* **Weather**: show **current** conditions and **6–24h forecast** for the chosen location.
* **Activities/POIs**: list **nearby** items (e.g., parks, museums, viewpoints) with distance; allow simple filters (e.g., “outdoors only”).
* **Recommendations**: at least one rule that adapts suggestions to weather (e.g., “if precipitation > 0, rank indoor options higher”).
* **Save list**: user can “save” activities for later (persisted across page refreshes by any reasonable means).

Use freely available public APIs. Good options include:

* **Weather**: Open-Meteo (no key), OpenWeatherMap (free tier)
* **Activities/POIs**: OpenTripMap (free tier), Wikidata/Wikipedia search, or any open geodata
* **Geocoding**: Nominatim (OpenStreetMap) or other open geocoder  
   (You may substitute equivalent, free public sources.)

## **The 5 Hard Requirements (non-negotiable)**

1. **Clear boundary via an internal API layer + OpenAPI**
   * Expose at least these endpoints from **your app’s own API** (not directly from the third-party APIs):  
     + GET /api/weather?lat={lat}&lon={lon} → normalized JSON for current + short forecast
     + GET /api/activities?lat={lat}&lon={lon}&radius={m}&type={indoor|outdoor|all} → normalized POIs
     + GET /api/recommendations?lat={lat}&lon={lon} → weather-aware, ranked list
   * Publish an **OpenAPI (Swagger) spec** for your endpoints at /openapi.json (or similar).
2. **Resilience: rate-limit handling, caching, and graceful degradation**
   * Implement **caching** with a short TTL for weather and POIs.
   * Implement **retry with backoff** for 429/5xx from upstream APIs.
   * Provide a **“mock mode”** (env flag) that serves deterministic responses from fixtures if upstream APIs are unavailable.
3. **Security & Privacy hygiene**
   * **No secrets** committed; all keys/config via **environment variables**.
   * **Input validation** for request params (lat/lon bounds, radius caps, type whitelist).
   * **Minimal data retention**: do not store precise user location beyond what’s needed for the request; document what you store and why.
4. **Complete, handoff-ready documentation**
   * **README** with one-command local setup, configuration, and run instructions.
   * **Architecture doc** with:
     + component & data-flow diagram,
     + error/timeout/rate-limit behaviors,
     + caching strategy & TTLs,
     + ADRs (at least 2) capturing key tradeoffs.
   * **Handoff guide** for a downstream team: backlog of next steps, milestones, risk register, and an implementation plan. The downstream team will be in charge of final testing, maintenance, and integration with a mobile app.
5. **Quality gates: tests + CI task + lint**
   * At least **5 unit tests** covering: request validation, caching behavior, retry/backoff, recommendation rule, and one serialization/normalization path.
   * A single **test command** that runs all tests headlessly.
   * A single **lint command** that enforces style/format rules.
   * A **CI script** (any runner) that executes install → lint → test on each push.

## **Non-Goals - Skip These Items**

* Payments, accounts, or complex auth flows (you may add a simple session if helpful).
* Full accessibility polish—do the basics well; call out gaps in your handoff guide.
* Multi-language support (note in backlog if relevant).

## **Suggested UX**

* **Search bar** for location + “use my location” option.
* **Weather panel**: condition, temp, precipitation, wind, next-6h mini forecast.
* **Filters**: distance radius and “indoor/outdoor/all”.
* **Activities list**: card with title, category, distance, short description, and “Save”.
* **Saved items**: simple list the user can revisit.

## **Recommendation Logic (example)**

* Score each activity:
  + Base score by relevance (proximity)  
    **Weather boost**:
    - If precipitation probability or amount > threshold → +score for indoor, −score for outdoor  
      If wind speed > threshold → +score for indoor
    - If temperature within a “pleasant” band → +score for outdoor
* Return items sorted by score, include the factors used.
* Document any logic used in the recommendations. Grading will be based on quality of documentation as well as the quality of the recommendations.

## **Deliverables Checklist**

* Working prototype (local run)
* Internal API with /api/weather, /api/activities, /api/recommendations
* OpenAPI spec exposed and committed
* Caching + retry/backoff + mock mode
* Tests, lint, and CI task
* README, Architecture doc (with diagrams), Handoff guide, ADRs
* Short **Research Summary** (1–2 pages): API choices, rate-limit policies, data licensing notes, and risks

## **Evaluation Rubric**

* **Architecture & Resilience** – boundaries, diagrams, caching, retries, error handling
* **Prototype Quality** – internal API shape, normalization, recommendation logic, UX clarity
  + You can choose any technology stack that you want, but justify your choice against other options in the documentation. Laravel is the recommended choice.
* **Documentation & Handoff** – README, handoff guide, ADRs, clarity and completeness
  + You should expect that the downstream team is capable in the technology of your choice. You should expect the downstream team to be competent programmers, but not specialists in a specific API or business analysts.
* **Testing & CI** – coverage of critical paths, ease of running, linting discipline
* **Research & Tradeoffs** – API/licensing awareness, rate-limit strategy, explicit risks/mitigations

AI-assisted writing is allowed, but overly verbose, generic, or mechanically generated documentation that lacks substance or specificity will lose points.

## **Submission**

* Link to repository.
* Instructions to run locally with a single command (plus .env.example).
* A short demo video or GIF (≤3 min) showing location search, weather view, filters, recommendations, and saved items.