Awesome—**Step 5: Favorites & Saved Searches (Alerts-ready)**.
We'll add DB tables, API routes (Python v2 Functions), and minimal UI hooks so it all works with what you've already built.

5.1 – SQL (run in SSMS, DB = `KidToKid`)

```
USE KidToKid;
GO

-- Favorites (many-to-many: user ↔ listing)
IF OBJECT_ID('dbo.favorite','U') IS NULL
BEGIN
  CREATE TABLE dbo.favorite (
    user_id    UNIQUEIDENTIFIER NOT NULL,
    listing_id UNIQUEIDENTIFIER NOT NULL REFERENCES dbo.listing(listing_id),
    created_at DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME(),
    CONSTRAINT PK_favorite PRIMARY KEY (user_id, listing_id)
  );
END
GO

-- Saved searches (store filters as JSON; we'll parse in API)
IF OBJECT_ID('dbo.saved_search','U') IS NULL
BEGIN
  CREATE TABLE dbo.saved_search (
    saved_search_id UNIQUEIDENTIFIER NOT NULL PRIMARY KEY DEFAULT NEWID(),
    user_id   UNIQUEIDENTIFIER NOT NULL,
    name      NVARCHAR(120) NOT NULL,
    query_json NVARCHAR(MAX) NOT NULL,   --
{category,size,condition,city,lat,lng,radiusKm}
    is_active BIT NOT NULL DEFAULT 1,
    created_at DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()
  );
END
GO

-- Convenience: ensure our dev buyer exists
IF NOT EXISTS (SELECT 1 FROM dbo.app_user WHERE email='dev@kidtokid.local')
  INSERT INTO dbo.app_user (email, display_name) VALUES
(N'dev@kidtokid.local', N'Dev Buyer');
GO
```

5.2 – Backend (Azure Functions Python v2)

Open `apps/api/function_app.py`. **Keep everything you already have** and add the parts below.
(They plug into the same `app = func.FunctionApp(...)` and reuse `_buyer_id()` and `_conn()`.)

5.2.1 – Favorites routes

```
# ---------- Favorites ----------
@app.route(route="favorites", methods=["GET"])
def get_favorites(req: func.HttpRequest) -> func.HttpResponse:
    try:
        with _conn() as c:
            cur = c.cursor()
            cur.execute("""
              SELECT f.listing_id, l.title, l.price_cents, l.city
              FROM dbo.favorite f
              JOIN dbo.listing l ON l.listing_id = f.listing_id
              WHERE f.user_id = ? AND l.is_active = 1
              ORDER BY f.created_at DESC
            """, _buyer_id())
            rows = cur.fetchall()
            data = [{"listing_id": str(r[0]), "title": r[1], "price_cents":
r[2], "city": r[3]} for r in rows]
            return func.HttpResponse(json.dumps(data),
mimetype="application/json")
    except Exception as e:
        logging.exception(e); return func.HttpResponse(f"Error: {e}",
status_code=500)

@app.route(route="favorites/{listingId}", methods=["POST"])
def add_favorite(req: func.HttpRequest) -> func.HttpResponse:
    lid = req.route_params["listingId"]
    try:
        with _conn() as c:
            cur = c.cursor()
            cur.execute("""
              MERGE dbo.favorite AS t
              USING (SELECT CAST(? AS UNIQUEIDENTIFIER) uid, CAST(? AS
UNIQUEIDENTIFIER) lid) AS s
              ON t.user_id=s.uid AND t.listing_id=s.lid
              WHEN NOT MATCHED THEN INSERT (user_id, listing_id) VALUES
(s.uid, s.lid);
            """, _buyer_id(), lid)
            c.commit()
            return func.HttpResponse(status_code=204)
    except Exception as e:
        logging.exception(e); return func.HttpResponse(f"Error: {e}",
status_code=500)

@app.route(route="favorites/{listingId}", methods=["DELETE"])
def remove_favorite(req: func.HttpRequest) -> func.HttpResponse:
    lid = req.route_params["listingId"]
    try:
        with _conn() as c:
            cur = c.cursor()
            cur.execute("DELETE FROM dbo.favorite WHERE user_id=? AND
listing_id=?", _buyer_id(), lid)
            c.commit()
            return func.HttpResponse(status_code=204)
    except Exception as e:
```

```
        logging.exception(e); return func.HttpResponse(f"Error: {e}",
status_code=500)
```

5.2.2 – Saved searches routes

We'll accept a tiny JSON body, store it, and provide a "run" route that returns current matches.
Allowed filters: `category`, `size`, `condition`, `city`, and optional `lat`,`lng`,`radiusKm` (we'll ignore radius for now unless you already store geo consistently; city + basic fields are enough).

```
import math

def _build_listing_filter(q: dict):
    # Build WHERE clause + params from whitelisted fields
    clauses, params = ["l.is_active = 1"], []
    if cat := q.get("category"):
        clauses.append("l.category = ?"); params.append(cat)
    if sz := q.get("size"):
        clauses.append("l.size = ?"); params.append(sz)
    if cond := q.get("condition"):
        clauses.append("l.[condition] = ?"); params.append(cond)
    if city := q.get("city"):
        clauses.append("l.city = ?"); params.append(city)
    # NOTE: For radius search you'd need Haversine with stored lat/lng;
skipping for MVP
    where_sql = " AND ".join(clauses)
    return where_sql, params

@app.route(route="saved-searches", methods=["GET"])
def list_saved_searches(req: func.HttpRequest) -> func.HttpResponse:
    try:
        with _conn() as c:
            cur = c.cursor()
            cur.execute("""
              SELECT saved_search_id, name, query_json, is_active, created_at
              FROM dbo.saved_search
              WHERE user_id = ?
              ORDER BY created_at DESC
            """, _buyer_id())
            rows = cur.fetchall()
            data = [{
                "saved_search_id": str(r[0]),
                "name": r[1],
                "query": json.loads(r[2]),
                "is_active": int(r[3]),
                "created_at": r[4].isoformat(),
            } for r in rows]
            return func.HttpResponse(json.dumps(data),
mimetype="application/json")
    except Exception as e:
        logging.exception(e); return func.HttpResponse(f"Error: {e}",
status_code=500)

@app.route(route="saved-searches", methods=["POST"])
```

```
def create_saved_search(req: func.HttpRequest) -> func.HttpResponse:
    try:
        body = req.get_json() if req.get_body() else {}
        name = body.get("name") or "My search"
        # Only whitelist expected keys
        q = {k: body.get(k) for k in
["category","size","condition","city","lat","lng","radiusKm"] if body.get(k)
is not None}
        q_json = json.dumps(q)
        with _conn() as c:
            cur = c.cursor()
            cur.execute("""
              INSERT INTO dbo.saved_search (user_id, name, query_json,
is_active)
              OUTPUT inserted.saved_search_id
              VALUES (?, ?, ?, 1)
            """, _buyer_id(), name, q_json)
            sid = str(cur.fetchone()[0])
            c.commit()
            return func.HttpResponse(json.dumps({"saved_search_id": sid}),
mimetype="application/json", status_code=201)
    except Exception as e:
        logging.exception(e); return func.HttpResponse(f"Error: {e}",
status_code=500)

@app.route(route="saved-searches/{sid}/run", methods=["POST"])
def run_saved_search(req: func.HttpRequest) -> func.HttpResponse:
    sid = req.route_params["sid"]
    try:
        with _conn() as c:
            cur = c.cursor()
            cur.execute("SELECT query_json FROM dbo.saved_search WHERE
saved_search_id = ? AND user_id = ?", sid, _buyer_id())
            row = cur.fetchone()
            if not row:
                return func.HttpResponse("Not found", status_code=404)
            q = json.loads(row[0])

            where_sql, params = _build_listing_filter(q)
            sql = f"""
              SELECT TOP 24 l.listing_id, l.title, l.price_cents, l.city,
                     CASE WHEN l.price_cents IS NULL OR l.price_cents=0 THEN 1
ELSE 0 END AS is_free
              FROM dbo.listing l
              WHERE {where_sql}
              ORDER BY l.created_at DESC
            """
            cur.execute(sql, *params)
            rows = cur.fetchall()
            data = [{"listing_id": str(r[0]), "title": r[1], "price_cents":
r[2], "city": r[3], "is_free": int(r[4])} for r in rows]
            return func.HttpResponse(json.dumps({"results": data, "query":
q}), mimetype="application/json")
    except Exception as e:
        logging.exception(e); return func.HttpResponse(f"Error: {e}",
status_code=500)
```

```
@app.route(route="saved-searches/{sid}/toggle", methods=["POST"])
def toggle_saved_search(req: func.HttpRequest) -> func.HttpResponse:
    sid = req.route_params["sid"]
    try:
        with _conn() as c:
            cur = c.cursor()
            # flip is_active (1→0 or 0→1)
            cur.execute("""
              UPDATE dbo.saved_search
                SET is_active = CASE WHEN is_active=1 THEN 0 ELSE 1 END
              WHERE saved_search_id=? AND user_id=?
            """, sid, _buyer_id())
            if cur.rowcount == 0:
                return func.HttpResponse("Not found", status_code=404)
            c.commit()
            return func.HttpResponse(status_code=204)
    except Exception as e:
        logging.exception(e); return func.HttpResponse(f"Error: {e}",
status_code=500)
```

Restart:

```
cd C:\Users\Selim Rawefi\My_Projects\KidToKid\apps\api
.venv\Scripts\activate
func start
```

You should now see these routes in the terminal:

- `favorites` (GET/POST/DELETE)
- `saved-searches` (GET/POST)
- `saved-searches/{sid}/run` (POST)
- `saved-searches/{sid}/toggle` (POST)

5.3 – Quick Postman tests

Favorites

1. `POST http://localhost:7071/api/favorites/<listingId>` → 204
2. `GET http://localhost:7071/api/favorites` → list items
3. `DELETE http://localhost:7071/api/favorites/<listingId>` → 204

Saved Searches

1. `POST http://localhost:7071/api/saved-searches`
   Body (JSON):

```
{ "name":"Free toys in Lisbon", "category":"toys", "city":"Lisbon",
"condition":"like-new" }
```

→ returns `{ "saved_search_id": "..." }`

2. `POST http://localhost:7071/api/saved-searches/<sid>/run`
   → returns `{ "results":[...], "query":{...} }`

3. `GET http://localhost:7071/api/saved-searches`
   → shows your saved searches

4. `POST http://localhost:7071/api/saved-searches/<sid>/toggle`
   → 204 (enable/disable)

5.4 – Frontend (very light wiring)

`src/lib/api.ts` – add:

```
export const favorites = {
  list: () => api<any[]>('/favorites'),
  add: (id: string) => api<void>(`/favorites/${id}`, { method: 'POST' }),
  remove: (id: string) => api<void>(`/favorites/${id}`, { method: 'DELETE' }),
};

export const savedSearches = {
  list: () => api<any[]>('/saved-searches'),
  create: (payload: any) => api<{saved_search_id:string}>('/saved-searches', {
    method: 'POST', body: JSON.stringify(payload)
  }),
  run: (sid: string) => api<{results:any[],query:any}>(`/saved-
searches/${sid}/run`, { method: 'POST' }),
  toggle: (sid: string) => api<void>(`/saved-searches/${sid}/toggle`, {
method: 'POST' }),
};
```

`src/pages/Home.tsx` – add a “♥ Favorite” button near “Add to Basket”:

```
import { favorites } from '../lib/api';

// inside each card:
<div className="flex gap-2 mt-3">
  <button onClick={() => basket.add(it.listing_id)} className="rounded-lg px-3
py-2 bg-sky-500 text-white text-sm">
    Add to Basket
  </button>
  <button onClick={() => favorites.add(it.listing_id)} className="rounded-lg
px-3 py-2 bg-pink-500 text-white text-sm">
```

```
    ♥ Favorite
  </button>
</div>
```

(Optional) New page `Favorites.tsx` (similar to Basket list) and a tiny page to create/run a saved search.

5.5 – What you now have (E2E)

- **Favorites**: add/remove/list working from DB → API → UI
- **Saved searches**: create/list/toggle + “run now” for matches
  (Alert delivery can be added later via Azure Queue + email/push—your data model supports it.)

5.6 – Next (pick one)

- **Deliverer flow** (deliveries table + required comment + status changes)
- **Image uploads** to Azure Blob (SAS pre-signed URLs)
- **Auth (Entra ID B2C)**: secure endpoints, roles (buyer/admin/deliverer)
- **Deploy**: SWA (frontend) + Functions (backend) with GitHub Actions

Tell me which track you want next, and I’ll wire it up.