

NexGate Location Service — Implementation Spec

“ **For the implementer (Claude Code):** read this whole file before writing code. Follow the existing conventions of the NexGate codebase (package layout, REST path style, auth/security, error format, ID type, DTO style, tests). Where this spec and the codebase conventions differ on *style*, follow the codebase. Where they differ on *behaviour/rules*, follow this spec. Keep it simple — no approval workflows, no extra services, no over-engineering.

0. Summary

The Location Service is a **module inside the existing Spring Boot backend** (not a microservice). It owns everything about *where* things are:

- Tanzania ward boundaries (official NBS 2022 data) and **pin → region/district/ward** resolution
- **Shop location** (one per shop, mandatory before a shop can be published)
- **Buyer saved addresses** (many per buyer, optional) and **recent locations**
- **Order address snapshots** (frozen copy of pickup/drop-off for an order)
- Privacy rules for who may see which part of a location

Shipping (providers, quotes, tracking) is **out of scope** — it will be built later *on top of* this service.

0.1 Where the Location Service is used across NexGate

It is the shared foundation for every feature that cares about *where*. Build the two core calls — `resolve(pin) → ward/district/region` and `areaMatch(areaSet, pin-or-area) → bool` — as reusable functions, because all of these call them:

- **Shop setup** — the shop's mandatory pickup location (pin → ward).
- **Buyer addresses & checkout** — saved/recent delivery addresses; the delivery pin.
- **Order snapshots** — frozen pickup + drop-off for every order.
- **Shipping — coverage (later)** — is a trip inside a shipper's areas / route stops?

- **Shipping — pricing** (*later*) — pin-to-pin distance for distance-based fees.
- **Shipping — trip type** (*later*) — same district / same region / cross-region, from the two wards.
- **Free-shipping zones** (*later*) — seller or NexGate "free within these areas / X km of shop".
- **"Near me" feed & search** (*later*) — show shops/products by the viewer's rough area.
- **Geo-targeted ads / boosting** (*later*) — show a boosted post only in chosen areas, or only where the seller can deliver; match on the viewer's **rough area only**, aggregate counts, no tracking.
- **Reports & analytics** — group orders/activity by ward/district/region codes (never per person).

“ Everything above reuses the **same ward data and the same two functions** — no new map work per feature. Only the two core calls are in scope now; the rest are listed so the API is designed to serve them.

1. Principles (non-negotiable)

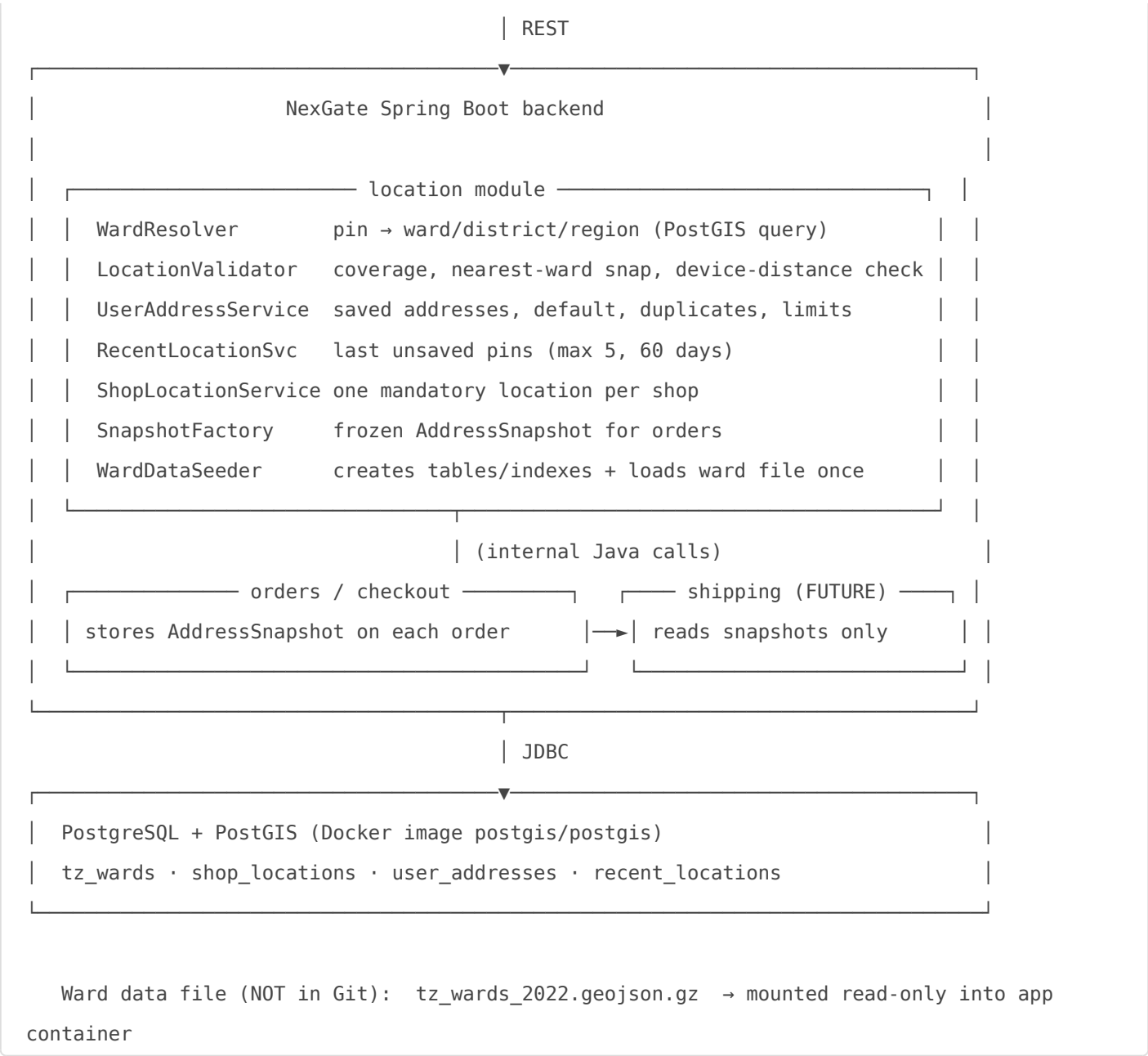
1. **The pin (lat/lng) is the truth.** Every saved location has a pin confirmed by the user on a map. No location is ever saved without coordinates.
2. **We take location only for a reason.** No location request on app open, no background tracking for buyers or shops, no movement trails. Store only the pin the user *confirmed*.
3. **Hierarchy comes from our own data, not Google.** Region/district/ward are computed on the backend from NBS ward shapes. Google is used **only on the client** (map display, place search, live label while placing the pin). The backend **never calls Google** and **never stores Google text**.
4. **Orders keep their own frozen copy** of addresses. Editing or deleting a saved address never changes past orders.
5. **Simple.** One module, plain SQL for spatial work, no Flyway, no version tracking, no JPA entity for the ward table.

2. Architecture

CLIENTS (Android · iOS · Web)

Pin screen: Google Map + Places search + live Google label (display only)

Sends: confirmed lat/lng + user text (landmark/note/phone/label)



Who decides what



3. Ward data

3.1 Source

- National Bureau of Statistics (NBS) Tanzania — *2022 Population and Housing Census, Tanzania Wards (Shapefiles)*.
- Already converted for us to **GeoJSON, WGS84 lat/lng (EPSG:4326)**, 21 self-intersecting shapes repaired.
- File: `tz_wards_2022.geojson.gz` (~6.7 MB gzipped, ~30 MB raw). 4,344 features.
- Attribution required somewhere in the app's legal/about page: "*Ward boundaries © National Bureau of Statistics, Tanzania (2022 PHC)*."

3.2 File format (what the seeder reads)

```
{
  "type": "FeatureCollection",
  "crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
  "features": [
    {
      "type": "Feature",
      "properties": {
        "ward_id": 1, "nbs_code": "010111011",
        "reg_code": "01", "reg_name": "Dodoma",
        "dist_code": "01", "dist_name": "Kondoa",
        "counc_code": "11", "counc_name": "Halmashauri ya Wilaya ya Kondoa",
        "ward_code": "011", "ward_name": "Changaa"
      },
      "geometry": { "type": "Polygon", "coordinates": [[[35.6782, -4.736511], ...]] }
    }
  ]
}
```

3.3 Data facts the code must respect

Fact	Consequence
Coordinates are [longitude, latitude] (GeoJSON order)	<code>ST_MakePoint(lng, lat)</code> — longitude first. Most common bug.
Geometry is a mix of <code>Polygon</code> and <code>MultiPolygon</code>	Always store with <code>ST_Multi(...)</code> into a <code>MultiPolygon</code> column.
<code>ward_code</code> alone is NOT unique; even <code>nbs_code</code> repeats for 5 wards (NBS data issue)	<code>ward_id</code> (1..4344) is the primary key. <code>nbs_code</code> is informational, not unique.

Fact	Consequence
Shapes stop at shorelines (sea, Lake Victoria, etc.)	Pins on beaches/jetties may fall outside every ward → nearest-ward fallback (§6).
31 regions, ~4,344 wards. Dar es Salaam has 5 councils (Ilala, Kinondoni, Temeke, Ubungu, Kigamboni)	Expected counts for sanity tests.

3.4 District & region come from the ward rows — no separate files

There is **no separate district or region boundary table**. Every ward row already carries its council, district and region names and codes, so a single `ST_Contains` lookup returns all levels at once:

pin → ward (Kariakoo) → already has `dist_name=Ilala`, `reg_name=Dar es Salaam`, codes included

- **Coverage checks** ("shipper serves Dar es Salaam region") match `reg_code`/`dist_code` on the resolved ward — no shape needed.
- **Reports** group orders by the stored `reg_code`/`dist_code`.
- **Drawing** a district or region outline on a map (e.g. shipper coverage) is the only case that needs a shape. Build it on demand by dissolving wards — do NOT store a second dataset:

```
-- district outlines (view or materialized view, optional)
SELECT reg_code, reg_name, dist_code, dist_name, ST_Union(geom) AS geom
FROM tz_wards GROUP BY reg_code, reg_name, dist_code, dist_name;

-- region outlines
SELECT reg_code, reg_name, ST_Union(geom) AS geom
FROM tz_wards GROUP BY reg_code, reg_name;
```

Use `counc_code` as well if councils (halmashauri) matter for coverage; the ward rows carry it too.

4. Infrastructure

4.1 Docker

- Database image must include PostGIS: use `postgis/postgis:<PG_MAJOR>-<POSTGIS>` with the **same Postgres major version** currently used (e.g. `postgres:16` → `postgis/postgis:16-3.4`). Same major version keeps existing data volumes working.

- App container mounts the data folder read-only:

```
services:
  db:
    image: postgis/postgis:16-3.4          # match current PG major version
  app:
    volumes:
      - /opt/nexgate/geo:/app/geo:ro      # host folder holding tz_wards_2022.geojson.gz
    environment:
      GEO_WARDS_FILE: /app/geo/tz_wards_2022.geojson.gz
```

- Local dev: put the file in `./data/geo/` and add `data/geo/` to `.gitignore`. Never commit the data file.

4.2 Config

```
# path to .geojson or .geojson.gz ; empty/missing → seeder logs WARN and skips
geo.wards-file=${GEO_WARDS_FILE:./data/geo/tz_wards_2022.geojson.gz}
```

4.3 No Flyway, no version tracking

Schema for this module is created by the **WardDataSeeder** at startup using idempotent SQL (`IF NOT EXISTS`). It runs after Hibernate initialises (use `ApplicationRunner` or `ApplicationReadyEvent`).

- `tz_wards` has **no JPA entity** (read via `JdbcTemplate`/native SQL only), so Hibernate never touches it.
- `shop_locations`, `user_addresses`, `recent_locations` may be JPA entities following project conventions, with **plain** `lat`/`lng` **double columns** (no hibernate-spatial needed). Spatial work is done in native SQL using `ST_SetSRID(ST_MakePoint(lng, lat), 4326)`. The seeder adds the spatial expression indexes afterwards.

5. Database

5.1 Entity relationships

```
erDiagram
    TZ_WARDS ||--o{ SHOP_LOCATIONS : "resolved to"
    TZ_WARDS ||--o{ USER_ADDRESSES : "resolved to"
```

```

TZ_WARDS ||--o{ RECENT_LOCATIONS : "resolved to"
USERS ||--o{ USER_ADDRESSES : owns
USERS ||--o{ RECENT_LOCATIONS : has
SHOPS ||--|| SHOP_LOCATIONS : "has exactly one"
ORDERS ||--|| ADDRESS_SNAPSHOT : "pickup (frozen copy)"
ORDERS ||--|| ADDRESS_SNAPSHOT : "drop-off (frozen copy)"

```

Addresses store `ward_id` **and** copies of the names/codes (denormalised) so reads never need a join and display stays stable.

5.2 `tz_wards` (created + filled by seeder)

```

CREATE EXTENSION IF NOT EXISTS postgis;

CREATE TABLE IF NOT EXISTS tz_wards (
    ward_id      INTEGER PRIMARY KEY,
    nbs_code     VARCHAR(16)  NOT NULL,
    reg_code     VARCHAR(4)   NOT NULL,
    reg_name     VARCHAR(100) NOT NULL,
    dist_code    VARCHAR(4)   NOT NULL,
    dist_name    VARCHAR(100) NOT NULL,
    counc_code   VARCHAR(4)   NOT NULL,
    counc_name   VARCHAR(200) NOT NULL,
    ward_code    VARCHAR(8)   NOT NULL,
    ward_name    VARCHAR(100) NOT NULL,
    geom         geometry(MultiPolygon, 4326) NOT NULL
);

CREATE INDEX IF NOT EXISTS idx_tz_wards_geom ON tz_wards USING GIST (geom);
CREATE INDEX IF NOT EXISTS idx_tz_wards_names ON tz_wards (reg_name, dist_name, ward_name);

```

5.3 Location tables (logical schema — use project ID type & audit conventions)

`shop_locations` — exactly one per shop

column	type	rule
id	project ID type	PK
shop_id	FK → shops	unique

column	type	rule
lat, lng	double	required, rounded to 6 decimals
ward_id	int	resolved by backend
reg_name, dist_name, ward_name, nbs_code	text	copied from ward
match_type	enum <code>CONTAINED</code> / <code>NEAREST</code>	from resolver
landmark	text (≤ 300)	required ("Near Shekilango stop, above Vodashop, 1st floor")
building	text (≤ 100)	optional ("Shop 12")
pickup_phone	text	required , E.164 (+255...)
official_address	text (≤ 100)	optional (Anwani za Makazi)
photo_url	text	optional (shop front photo, uses existing media upload)
created_at, updated_at	timestamp	

`user_addresses` — buyer's saved addresses (0..10 per user)

column	type	rule
id	project ID type	PK
user_id	FK → users	
label_type	enum <code>HOME</code> / <code>WORK</code> / <code>OTHER</code>	HOME and WORK max one each per user
custom_label	text (≤ 40)	required when <code>OTHER</code> ("Mama's place")
lat, lng	double	required, 6 decimals
ward_id, reg_name, dist_name, ward_name, nbs_code, match_type		resolved by backend
note	text (≤ 300)	optional note for rider ("Black gate, call on arrival")
receiver_name	text	defaults to user's name
receiver_phone	text	defaults to user's phone, E.164
is_default	boolean	at most one <code>true</code> per user
last_used_at	timestamp	updated when used in an order
created_at, updated_at	timestamp	

`recent_locations` — unsaved delivery pins (max 5 per user, expire after 60 days)

column	type	rule
--------	------	------

id	project ID type	PK
user_id	FK → users	
lat, lng, ward_id, reg_name, dist_name, ward_name, match_type		as above
note, receiver_name, receiver_phone		as used in that order
last_used_at	timestamp	

Indexes created by seeder after Hibernate (idempotent):

```
CREATE UNIQUE INDEX IF NOT EXISTS ux_shop_locations_shop ON shop_locations (shop_id);
CREATE INDEX IF NOT EXISTS idx_shop_locations_geog ON shop_locations
    USING GIST ((ST_SetSRID(ST_MakePoint(lng, lat), 4326)::geography)); -- for future "near
me"
CREATE INDEX IF NOT EXISTS idx_user_addresses_user ON user_addresses (user_id);
CREATE UNIQUE INDEX IF NOT EXISTS ux_user_addresses_default ON user_addresses (user_id) WHERE
is_default;
CREATE UNIQUE INDEX IF NOT EXISTS ux_user_addresses_home ON user_addresses (user_id) WHERE
label_type = 'HOME';
CREATE UNIQUE INDEX IF NOT EXISTS ux_user_addresses_work ON user_addresses (user_id) WHERE
label_type = 'WORK';
CREATE INDEX IF NOT EXISTS idx_recent_locations_user ON recent_locations (user_id,
last_used_at DESC);
```

(If the project creates these tables itself via its own mechanism, keep the indexes in the seeder anyway.)

5.4 AddressSnapshot (frozen copy stored on orders)

Not a table owned by this module. The orders module stores it (e.g. a `jsonb` column `pickup_snapshot` / `dropoff_snapshot`, or embedded columns — follow project convention).

Immutable once stored.

```
{
  "lat": -6.774512, "lng": 39.241871,
  "wardId": 1131, "nbsCode": "070113062",
  "regionName": "Dar es Salaam", "districtName": "Kinondoni", "wardName": "Kijitonyama",
  "matchType": "CONTAINED",
  "label": "Home", // null if unsaved
```

```

"landmarkOrNote": "Black gate, call on arrival",
"building": null,
"contactName": "Kibuti", "contactPhone": "+2557XXXXXXX",
"sourceType": "USER_ADDRESS", // USER_ADDRESS | NEW_PIN | SHOP_LOCATION
"sourceId": "...", // id of saved address/shop location, null for
NEW_PIN
"capturedAt": "2026-10-08T12:30:00Z"
}

```

6. Pin resolution (core algorithm)

```

input: lat, lng
|
|— basic checks: lat ∈ [-90,90], lng ∈ [-180,180], not (0,0) — fail → 400
INVALID_COORDINATES
|
|— round to 6 decimals
|
|— Query A: ward whose shape CONTAINS the point
|   found → matchType = CONTAINED, distanceM = 0 —————→ return ward
|
|— Query B: nearest ward within 500 m (shoreline, beach, jetty, border gaps)
|   found → matchType = NEAREST, distanceM = n —————→ return ward
|
|— nothing → OUTSIDE_COVERAGE (sea, lake, outside Tanzania)

```

```

-- Query A (uses GIST index)
SELECT ward_id, nbs_code, reg_name, dist_name, counc_name, ward_name
FROM tz_wards
WHERE ST_Contains(geom, ST_SetSRID(ST_MakePoint(:lng, :lat), 4326))
LIMIT 1;

-- Query B (only if A returns nothing)
SELECT ward_id, nbs_code, reg_name, dist_name, counc_name, ward_name,
       ST_Distance(geom::geography, ST_SetSRID(ST_MakePoint(:lng, :lat), 4326)::geography) AS
distance_m
FROM tz_wards
WHERE ST_DWithin(geom::geography, ST_SetSRID(ST_MakePoint(:lng, :lat), 4326)::geography, 500)

```

```
ORDER BY distance_m  
LIMIT 1;
```

Point exactly on a border → whichever ward Query A returns is fine. Constant:

```
NEAREST_WARD_MAX_METERS = 500.
```

6.1 Validation rules

Check	Applies to	Result
Invalid coordinates	all	block <code>INVALID_COORDINATES</code>
No ward within 500 m	all	block <code>OUTSIDE_COVERAGE</code> ("We don't deliver here yet")
Matched by NEAREST	all	allow, <code>matchType=NEAREST</code> (info only)
Pin > 1 km from device position (client sends optional <code>deviceLat/deviceLng</code>)	shop only	allow, return warning <code>FAR_FROM_DEVICE</code> with distance; client asks "Your pin is 3.2 km from where you are. Is this your shop?"
Saved address within 50 m of another saved address of same user	buyer save	<code>409 NEAR_DUPLICATE</code> with the existing address; client asks "This looks like  Home. Use it?"; client may resend with <code>allowNearDuplicate=true</code>
More than 10 saved addresses	buyer save	block <code>ADDRESS_LIMIT_REACHED</code>
Second HOME or WORK	buyer save/edit	block <code>LABEL_ALREADY_USED</code>

Client-only checks (not backend): GPS accuracy circle, minimum zoom before confirm, satellite toggle.

7. Business rules

7.1 Shop location

- Exactly **one** location per shop. **Required to publish a shop** — expose `hasLocation(shopId)` for the shop module.
- Required fields: pin, landmark, pickup phone. Optional: building, official address, photo.
- Only the shop owner (and shop admins per existing roles) can set/edit it. No approval step.
- Editing affects **future** orders only (old orders keep snapshots).

7.2 Buyer addresses

- Saving is **optional** and the buyer's choice (checkbox "Save for next time").
- Max 10 saved. HOME/WORK at most once each; OTHER needs a custom label.
- First saved address becomes default automatically. Buyer can change default. If the default is deleted → no default (client pre-selects most recently used).
- Receiver name/phone default to the buyer's profile; buyer can change ("someone else").
- Edit = change pin (re-resolved), label, note, receiver. Delete = **hard delete**.

7.3 Recent locations

- When an order uses a **new pin that was not saved**, store it in `recent_locations`.
- Keep max **5** per user (delete oldest beyond 5 on insert). Ignore and delete entries older than **60 days** (on read/insert — no scheduler needed).
- If the new pin is within 50 m of an existing recent, update that one's `last_used_at` instead of inserting.
- Buyer can delete one or clear all. A recent can be promoted to a saved address (client posts a normal create).

7.4 Orders / snapshots

- Checkout passes either `addressId` (saved), `recentId`, or a **new pin** payload.
- Location module returns an `AddressSnapshot`; orders module stores it; snapshot never changes.
- Using a saved address updates its `last_used_at`.
- Pickup snapshot comes from the shop location (`sourceType=SHOP_LOCATION`).

7.5 Account deletion

Deletes all `user_addresses` and `recent_locations` of that user. Order snapshots remain (order records).

8. Privacy & visibility

Data	Buyer	Seller	Rider / delivery provider	Third parties
Buyer saved addresses / reents	own only	□	□	□
Order drop-off area (ward, district, region)	□	□	□	□

Data	Buyer	Seller	Rider / delivery provider	Third parties
Order drop-off exact pin + note + phone	☐	only if seller delivers itself (<code>sellerDelivers=true</code>)	☐ for that shipment only	☐
Shop location (pin, landmark, ward)	☐ public	☐	☐	☐
Shop pickup phone	after order	☐	☐	☐

Implement as two snapshot views: `AddressSnapshotPublicView` (area only) and `AddressSnapshotFullView`. Analytics/reports aggregate by ward/district only — never per person. Never log full coordinates together with user identifiers at INFO level.

9. REST API

“ Paths shown as examples — adapt prefix/style/auth to the existing project. All endpoints require authentication except `GET /shops/{id}/location` (public view).

9.1 Resolve (used live by pin screen after the map stops moving — debounce ~400 ms client side)

```
GET /api/v1/locations/resolve?lat=-6.8190&lng=39.2768
```

```
200 { "covered": true, "matchType": "CONTAINED", "distanceM": 0,
      "ward": { "wardId": 1146, "nbsCode": "070214112", "wardName": "Kariakoo",
                "districtName": "Ilala", "councilName": "Halmashauri ya Jiji la Dar es
Salaam",
                "regionName": "Dar es Salaam" },
      "areaLabel": "Kariakoo, Ilala, Dar es Salaam" }
200 { "covered": false, "reason": "OUTSIDE_COVERAGE" }
```

Nothing is stored by this call.

9.2 Buyer addresses

Method	Path	Notes
GET	/api/v1/me/addresses	list, default first, then <code>last_used_at</code> desc
POST	/api/v1/me/addresses	create (body below)
PUT	/api/v1/me/addresses/{id}	update (same body)
DELETE	/api/v1/me/addresses/{id}	hard delete
POST	/api/v1/me/addresses/{id}/default	make default
GET	/api/v1/me/recent-locations	max 5, $\leq$ 60 days
DELETE	/api/v1/me/recent-locations/{id}	remove one
DELETE	/api/v1/me/recent-locations	clear all

```
POST /api/v1/me/addresses
{ "lat": -6.774512, "lng": 39.241871,
  "labelType": "OTHER", "customLabel": "Mama's place",
  "note": "Black gate, call on arrival",
  "receiverName": null, "receiverPhone": null,      // null → use profile
  "makeDefault": false, "allowNearDuplicate": false }
```

9.3 Shop location

Method	Path	Notes
GET	/api/v1/shops/{shopId}/location	public view (no phone unless authorised)
PUT	/api/v1/shops/{shopId}/location	create or replace; owner only

```
PUT /api/v1/shops/{shopId}/location
{ "lat": -6.770901, "lng": 39.229410,
  "landmark": "Near Shekilango stop, above Vodashop, 1st floor",
  "building": "Shop 12", "pickupPhone": "+2557XXXXXXX",
  "officialAddress": null, "photoUrl": null,
  "deviceLat": -6.7712, "deviceLng": 39.2291 }      // optional, for FAR_FROM_DEVICE warning
→ 200 { ...location..., "warnings": [ { "code": "FAR_FROM_DEVICE", "distanceM": 3200 } ] }
```

9.4 Internal Java API (for orders/shop modules — not REST)

```

ResolveResult resolve(double lat, double lng);
boolean hasShopLocation(ShopId shopId);
AddressSnapshot snapshotFromShop(ShopId shopId);
AddressSnapshot snapshotFromAddress(UserId userId, AddressId addressId);    // also bumps
last_used_at
AddressSnapshot snapshotFromRecent(UserId userId, RecentId recentId);
AddressSnapshot snapshotFromNewPin(UserId userId, NewPinRequest req);      // req.saveForLater
→ creates address, else records recent

```

9.5 Error codes

`INVALID_COORDINATES`, `OUTSIDE_COVERAGE`, `NEAR_DUPLICATE` (409, includes existing address),
`ADDRESS_LIMIT_REACHED`, `LABEL_ALREADY_USED`, `SHOP_LOCATION_REQUIRED` (thrown by shop publish),
`NOT_FOUND`, `FORBIDDEN`. Use the project's existing error response format.

10. WardDataSeeder (startup)

```

App starts (after Hibernate)
|
| 1. CREATE EXTENSION IF NOT EXISTS postgis          (log clear error if not permitted)
| 2. CREATE TABLE IF NOT EXISTS tz_wards ... + indexes
| 3. CREATE INDEX IF NOT EXISTS ... on location tables ($5.3)
|
| 4. BEGIN; SELECT pg_advisory_xact_lock(<constant>)  ← only one container seeds
|    SELECT count(*) FROM tz_wards
|      | > 0 → COMMIT, done (normal startup, instant)
|      | = 0 → file exists?
|      | no → WARN "ward file not found at ...; ward lookup disabled", COMMIT,
continue startup
|      | yes → stream features (Jackson streaming parser, gzip if *.gz)
|              batch insert 200 rows:
|              INSERT INTO tz_wards (... , geom)
|              VALUES (... , ST_Multi(ST_SetSRID(ST_GeomFromGeoJSON(?), 4326)))
|              ON CONFLICT (ward_id) DO NOTHING
|              COMMIT; ANALYZE tz_wards
|              log "Loaded N wards in X s" (expect 4344)
|

```

└ 5. If wards were just loaded AND user_addresses/shop_locations/recent_locations already have rows
→ re-resolve all of them (update ward_id + name copies + match_type). Order snapshots untouched.

Rules:

- Stream the file — never load all 30 MB into memory.
- Accept both `.geojson` and `.geojson.gz`.
- Seeder failure must not crash the app; resolve endpoint then returns `503 WARD_DATA_UNAVAILABLE`.
- **Updating to new boundaries later:** replace file on server, `TRUNCATE tz_wards;`, restart. Step 5 re-resolves saved locations.

11. Client behaviour (Android · iOS · Web)

11.1 Location permission

- **Never** ask on app open. Ask only when the user taps "**Use my location**" on the pin screen.
- Show our own explainer first; trigger the OS/browser prompt only if the user agrees.
- **Foreground / "While using the app" only.** Never background location for buyers or shops.
- Handle **approximate** location (Android 12+, iOS 14+): use it to centre the map, ask user to drag map to exact spot.
- iOS: purpose string e.g. "*NexGate uses your location to set your shop or delivery address faster.*"
- Web: only after a click, HTTPS only. If denied, search + map always still work. Location is a shortcut, never a gate.

```
┌ - - - - - ┐
│           │ │           │
│   Use your location?   │
│ We use it once to place your │
│   pin faster. You can also │
│ search or move the map.    │
└ - - - - - ┘
│ │   Use my location   │ │   → OS permission prompt
```

```
L _ _ _ _ _ ]  
|           |  
| Not now   | → stay on map, no prompt  
|           |  
L _ _ _ _ _ ]
```

Behaviour:

1. **Fixed centre pin, the map moves** (Bolt/Uber pattern). Pin lifts while dragging, drops when the map stops.
2. Start position: accurate GPS → last used location → centre of user's city. Never whole-country view.
3. **Map / Satellite toggle** (satellite matters — many streets unnamed).
4. GPS **accuracy circle**; if accuracy > 100 m show "Location is approximate — move the map to your exact spot".
5. **Confirm disabled until zoom ≥ 17** (street level). Hint: "Zoom in to place exactly".
6. Search (Google Places) only **moves** the map: "Now move the map to the exact entrance".
7. When map stops (debounce ~400 ms): show Google label + call `/locations/resolve`. If `covered=false` → disable Confirm, show "We don't deliver here yet".

```

┌ - - - - - ┐
|   ← Shop location      Step 3/4   |
| ┌ - - - - - ┐ |
| |   Search place or area   | |
| └ - - - - - ┘ |
|
| ┌ - - - - - ┐ |
| | [Map|Satellite]         | |
| |                         | |
| |                         | |
| | (  ) ← accuracy        | |
| |   fixed centre         | |
| | move the map           | |
| | [⊙]                    | | [⊙] = use my location
| └ - - - - - ┘ |
|
| ▲ Zoom in to place exactly
| (shown until zoom ≥ 17)
|
| Sinza, Ubungu, Dar es Salaam ← Google label (display only)
| ┌ - - - - - ┐ |
| | Confirm location →    | |
| └ - - - - - ┘ |

```

11.3 Shop flow (mandatory, step in shop creation)

```
┌───────────┐
│ ← Shop location      Step 3/4                     │
├───────────┤
│ [mini map 📍] Change                               │
├───────────┤
│ Makurumla, Ubungo, Dar                             ← from /resolve (NBS)
└───────────┘

Directions / landmark *
├───────────┤
│ Near Shekilango stop,                              │
│ above Vodashop, 1st floor                          │
└───────────┘

Building / shop no. (optional)
├───────────┤
│ Shop 12                                              │
└───────────┘

Pickup contact phone *
├───────────┤
│ +255 7XX XXX XXX                                    │
└───────────┘

Official address (optional)
├───────────┤
│ Anwani za Makazi                                   │
└───────────┘

📷Add shop front photo (optional)
├───────────┤
│ Save & continue →                                  │
└───────────┘
```

If response has `FAR_FROM_DEVICE` → dialog "Your pin is 3.2 km from where you are. Is this your shop?" [Yes, it's correct] [Adjust pin].

11.4 Buyer delivery flow (pin only, Bolt style)

Checkout — choose where

Deliver to

● Home (default)

Mbezi Juu, Ubungo

○ Work

Kivukoni, Ilala

○ Mama's place

Tabata, Ilala

Recent

○ Kijitonyama [Save]

New location (pin)

Manage addresses

First-time buyer with nothing saved → goes straight to the pin screen.

New location — full map + bottom sheet

← Deliver to [Sat]

FULL MAP

[☉]

Search

Home Mbezi Juu

Work Kivukoni

Recent Kijitonyama

Mikocheni, Kinondoni

[Deliver here →]

← tap jumps the pin there

Quick confirm sheet

```
[
┌───────────┐
│ Mikocheni, Kinondoni │
└───────────┘

Note for rider (optional)

┌───────────┐
│ Black gate, call on arrival │
└───────────┘

Receiver: Me (+255 7XX...) ⓘ ← "Someone else" → name + phone

[ ] Save as   ⓘ ⓘ ⓘ ⓘ ⓘ ⓘ ⓘ ← labels shown only when ticked

┌───────────┐
│ Continue → │
└───────────┘
┌───────────┐
```

Profile — manage addresses

```
|  
|← My addresses  
  
| |-----|  
| | Home ✓default      |  
| Mbezi Juu, Ubungo    |  
| |-----|  
| | Work                |  
| Kivukoni, Ilala     |  
| |-----|  
| | Mama's place        |  
| Tabata, Ilala       |  
| |-----|  
|   = Edit pin · Rename ·  
| Set default · Delete  
  
| |-----|  
| | Add address          |  
| |-----|  
| Clear recent locations  
|
```

Optional secondary onboarding (buyer)

```

┌ - - - - - ┐
│           │
│ Faster checkout?
│ Add your home address now
│ and skip it at checkout.
│ ┌ - - - - - ┐
│ │ Add address │ → pin screen → quick confirm (Save always on)
│ └ - - - - - ┘
│
│ Maybe later
└ - - - - - ┘

```

Area lines shown after saving (lists, checkout) are built from **NBS names** (`ward, district`) plus the user's label — never from Google text.

12. Flows

12.1 Shop sets location

```

sequenceDiagram
    actor Owner
    participant App
    participant Google as Google Maps/Places (client)
    participant API as Location module
    participant DB as PostGIS

    Owner->>App: Step 3 "Shop location"
    App->>Owner: Explainer → tap "Use my location"
    App->>App: OS permission (while in use), one GPS reading
    Owner->>App: Move map under fixed pin, zoom ≥ 17
    App->>Google: label for map centre (display only)
    App->>API: GET /locations/resolve?lat&lng
    API->>DB: ST_Contains / nearest ≤ 500 m
    DB-->>API: ward, district, region
    API-->>App: covered=true, areaLabel
    Owner->>App: Confirm → landmark, phone, (building, photo)
    App->>API: PUT /shops/{id}/location (+deviceLat/Lng)
    API->>DB: resolve again, upsert shop_locations
    API-->>App: saved (+ FAR_FROM_DEVICE warning if > 1 km)

```

12.2 Buyer checkout with a new pin

```
sequenceDiagram
    actor Buyer
    participant App
    participant Orders as Orders/Checkout
    participant Loc as Location module
    participant DB as PostGIS
    Buyer->>App: Checkout → "📍 New location"
    Buyer->>App: Place pin, note, receiver, [Save?]
    App->>Loc: GET /locations/resolve (live while moving)
    Buyer->>App: Continue / Pay
    App->>Orders: place order { newPin, saveForLater }
    Orders->>Loc: snapshotFromNewPin(user, req)
    Loc->>DB: resolve ward
    alt saveForLater = true
        Loc->>DB: insert user_addresses (rules §7.2)
    else
        Loc->>DB: upsert recent_locations (max 5, 60 days)
    end
    Loc-->>Orders: AddressSnapshot (frozen)
    Orders->>DB: store order + pickup & drop-off snapshots
```

12.3 Shop publish guard

```
Shop "Publish" → shopModule calls location.hasShopLocation(shopId)
├─ true → publish
└─ false → 400 SHOP_LOCATION_REQUIRED → app opens Shop location step
```

13. Out of scope (do NOT build now)

- Shipping, providers (Bolt etc.), quotes, tracking, recommendation engine
- Delivery zones and hubs (future: same pattern — shapes table + `ST_Contains`)
- Mtaa/street level (no public shapes; future table identical to `tz_wards` if NBS provides them)
- "Near me" feed (index on shop_locations is prepared; feature later — must use rough area only, not stored)
- Any backend call to Google; storing Google labels

14. Acceptance tests

14.1 Ward resolution (integration test against loaded data — these were verified against the file)

Place	lat	lng	Expected ward	District	Region
Kariakoo market	-6.8190	39.2768	Kariakoo	Ilala	Dar es Salaam
Ubungu bus terminal	-6.7889	39.2083	Ubungu	Ubungu	Dar es Salaam
JNIA airport	-6.8781	39.2026	Kipawa	Ilala	Dar es Salaam
Posta / city centre	-6.8160	39.2900	Kivukoni	Ilala	Dar es Salaam
Stone Town	-6.1622	39.1887	Shangani	Mjini	Mjini Magharibi
Arusha clock tower	-3.3700	36.6940	Sekei	Arusha	Arusha
Mwanza centre	-2.5164	32.9006	Nyamagana	Nyamagana	Mwanza
Mbeya town	-8.9094	33.4608	Ruanda	Mbeya	Mbeya
Dodoma centre	-6.1722	35.7395	Majengo	Dodoma	Dodoma
Indian Ocean off Dar	-6.80	39.40	— OUTSIDE_COVERAGE		
Nairobi	-1.2864	36.8172	— OUTSIDE_COVERAGE		

Also: `SELECT count(*) FROM tz_wards` = **4344**; `SELECT count(DISTINCT reg_name)` = **31**. Use Testcontainers with `postgis/postgis` image if the project already uses Testcontainers; otherwise follow project test setup.

14.2 Rules

- Seeder runs twice → still 4344 rows, second run instant.
- Two app instances start together → 4344 rows, no errors (advisory lock).
- Missing ward file → app starts, WARN logged, `/resolve` returns 503.
- Swapped lat/lng (39.2768, -6.8190) → `OUTSIDE_COVERAGE` (guards against order bug).
- 11th saved address → `ADDRESS_LIMIT_REACHED`. Second HOME → `LABEL_ALREADY_USED`.
- Save within 50 m of existing → `409 NEAR_DUPLICATE`; with `allowNearDuplicate=true` → saved.

- Delete default → no default remains. First saved address → becomes default.
 - 6th unsaved pin → only 5 recents remain; entries > 60 days not returned.
 - Edit saved address after an order → order snapshot unchanged.
 - Shop without location cannot publish → `SHOP_LOCATION_REQUIRED`.
 - Shop PUT with device 3 km away → saved + `FAR_FROM_DEVICE` warning.
 - Seller view of order shows area only unless `sellerDelivers=true`.
 - `TRUNCATE tz_wards` + restart → reload + saved addresses re-resolved.
-

15. Build order

1. Docker: switch DB to `postgis/postgis` (same PG major), mount `/opt/nexgate/geo`, add config, `.gitignore data/geo/`.
 2. `WardDataSeeder` + `WardResolver` + `GET /locations/resolve` + tests §14.1. ← **foundation**
 3. Shop location (entity, PUT/GET, `hasShopLocation`, publish guard).
 4. Buyer addresses + recents (+ rules tests).
 5. `AddressSnapshot` + checkout integration + privacy views.
 6. Clients: shared pin-screen component, shop step, checkout picker, manage addresses, optional onboarding.
-

Revision #2

Created 11 October 2026 14:39:15 by Admin Qbit

Updated 11 October 2026 14:42:51 by Admin Qbit