Frontend Specification — DRA-UG (Flutter)

Owner (frontend): Bantrobusa Kazibwe FZ and Jemimah Tendo
Purpose: Clear contract for backend: which endpoints, payloads, response shapes, auth & security expectations, realtime/notification behavior, and offline handling that the Flutter frontend will implement.

---

1. High-level app features (frontend responsibilities)

1. **Authentication**
    - Email/password register & login
    - Social login stubs (Google, Apple) — backend should support OAuth token verification
    - Secure token storage (flutter_secure_storage) and refresh/token expiry handling
2. **Roles / Views**
    - **Normal user** (individual donor)
    - **Donor company** (business donors) — extra profile fields (company name, CSR info)
    - **Receiver organization** (orphanages, kitchens) — verification doc upload and status
    - UI shows different home screen content by role
3. **Donations**
    - Donors (individuals & companies) create donation listings with optional photo(s)

- Receivers see donations intended for them (or browse available donations if public)
- Receivers can request donations
- Support CRUD where appropriate (create, list, read; donors may cancel)

4. **Order/Pickup tracking (Uber-like)**
    - Pickup lifecycle: `pending` → `assigned` → `in_progress` → `picked` → `delivered`
    - Real-time driver location updates so donor/receiver can view progress on map
    - Photo proof on pickup and delivery (camera capture)
    - Recipient confirmation flow (photo + staff name)
5. **Notifications**
    - Push notifications via FCM for assignment, status updates, new donations, verification results
    - In-app notification list
    - Backend must send push payloads; frontend registers FCM token and receives messages
6. **Social share**
    - Option to create a short social post (tweet/share) when donating. Frontend will:
        - Use share intent to post to Twitter/Facebook etc OR call backend `POST /posts` if platform-managed.
7. **Offline**
    - Outbox pattern: queue mutating actions locally (create donation, request donation, confirm delivery) and sync when online.
    - Deduplication / idempotency using `client_id` UUIDs.

---

2. Roles & permissions (frontend expectations)

- **role:** `donor_individual`, `donor_company`, `receiver`, `volunteer`, `admin`
- Frontend will request user role on login response and render UI accordingly.
- Backend MUST enforce these permissions server-side; frontend only uses for display.

---

3. Authentication & session behavior

Endpoints

- `POST /api/v1/auth/register`
    - body: `{ "name","email","password","role", "org_name?" }`
    - returns: `{ "user": {...}, "token": "jwt_or_sanctum_token", "expires_at": "ISO8601" }`
- `POST /api/v1/auth/login`
    - body: `{ "email","password" }`
    - returns same shape as register.
- `POST /api/v1/auth/logout` — invalidate token
- `POST /api/v1/auth/refresh` — optional token refresh

Frontend expects:

- Token-based auth in `Authorization: Bearer <token>` header for all protected routes.

- Token expiry code path (401) — on 401, attempt refresh or redirect to login.

Sample login response

```
{
  "user": {
    "id": 123,
    "name": "Jane Doe",
    "email": "jane@example.com",
    "role": "donor_individual"
  },
  "token": "eyJhbGci....",
  "expires_at": "2025-11-01T12:00:00Z"
}
```

---

4. Core data models (frontend view)

Use ISO8601 timestamps. All IDs are integers or UUIDs (choose one consistently).

Donation (primary model)

```
{
  "id": 456,
  "client_id": "uuid-v4-client-id", // used for offline/outbox mapping
  "title": "Leftover fried chicken (10kg)",
  "description": "Cooked today, sealed.",
  "quantity_est_kg": 10,
  "storage_condition": "room_temp",
  "pickup_window": {
    "from": "2025-10-20T09:00:00Z",
    "to": "2025-10-20T12:00:00Z"
  },
  "donor_id": 123,
  "donor_type": "donor_company",
  "photos": ["https://.../photo1.jpg"],
  "location": { "lat": 0.347596, "lng": 32.582520 },
  "status": "available", // available | requested | reserved | cancelled | completed
  "created_at":"2025-10-20T07:00:00Z",
  "visibility": "public" // or "private" (private => flagged for certain receivers)
}
```

Pickup


```
{
  "id": 789,
  "donation_id": 456,
  "requester_org_id": 555,
  "assigned_driver_id": 321,
  "status": "assigned",
  "pickup_proof": "https://.../pickup.jpg",
  "delivery_proof": "https://.../delivery.jpg",
  "requested_at": "2025-10-20T07:10:00Z",
  "updated_at": "2025-10-20T07:30:00Z"
}
```


Organization (receiver)


```
{
  "id": 555,
  "name": "Little Stars Orphanage",
  "address": "Some market, Kampala",
  "lat": 0.345678,
  "lng": 32.567890,
  "verified": false,
  "verification_doc_url": "https://.../doc.jpg"
}
```


---

5. API endpoints (detailed) — prioritized & sample payloads

> **Note:** all protected endpoints require `Authorization: Bearer <token>`.

Auth

- `POST /api/v1/auth/register` — body:

{ "name":"", "email":"", "password":"", "role":"donor_individual|receiver|donor_company|volunteer" }

- `POST /api/v1/auth/login` — body `{ "email","password" }`

User

- `GET /api/v1/users/me` — returns user profile & role
- `PUT /api/v1/users/me` — update profile (name, phone, avatar_url)

FCM / Push token

- `POST /api/v1/push-tokens` — register device

{ "token": "<fcm_token>", "platform": "android|ios", "device_id": "uuid" }

- `DELETE /api/v1/push-tokens/:id` — remove

Donations

- `POST /api/v1/donations` (multipart/form-data) — create donation
    - fields: `client_id` (UUID), `title`, `description`, `quantity_est_kg`, `pickup_from`, `pickup_to`, `storage_condition`, `lat`, `lng`, `visibility`
    - files: `photos[]` (images)
- `GET /api/v1/donations` — list with query:
    - `?page=1&per_page=20&lat=...&lng=...&radius_km=10&filter=status:available`
- `GET /api/v1/donations/:id` — single donation detail
- `DELETE /api/v1/donations/:id` — cancel (donor authorized)

Sample success response (create)

{ "id": 456, "client_id":"uuid", "status":"available", "created_at":"..." }

Organizations (receiver registration)

- `POST /api/v1/organizations` (multipart) — register receiver
    - fields: `name`, `address`, `lat`, `lng`, contact person, `verification_doc` file
- `GET /api/v1/organizations/:id`
- `POST /api/v1/organizations/:id/verify` — admin only: set `verified: true`

Requests & Pickups

- `POST /api/v1/donations/:id/request` — receiver requests donation
    - body: `{ "org_id": 555, "notes": "" }`
    - returns pickup record or `request_id`
- `POST /api/v1/pickups` — create pickup (internal, usually created by backend on request)
- `POST /api/v1/pickups/:id/assign` — admin/auto: assign driver `{ "driver_id": 321 }`
- `POST /api/v1/pickups/:id/status` — update status
    - body: `{ "status":"in_progress|picked|delivered", "photo": file_or_url, "notes":"", "client_id":"uuid" }`
- `POST /api/v1/pickups/:id/location` — driver sends continuous location updates
    - body: `{ "lat":0.34, "lng":32.5, "bearing":10, "speed":2.5, "timestamp":"ISO8601" }`
    - **Frontend expects** frequent small updates while `in_progress`.

Realtime/Tracking behavior (how frontend will use)

- Frontend will:
    - Poll `GET /api/v1/pickups/:id` for quick fallback every 10s if sockets not available.
    - Listen for FCM messages for status changes (preferred).
    - Use `POST /api/v1/pickups/:id/location` to push driver location.
- Backend should broadcast driver locations via:
    - FCM to subscribed donor & receiver devices OR
    - WebSockets / socket rooms if available (we will support both; start with FCM).

Realtime message structure (FCM)

```
{
  "type":"pickup.location",
  "pickup_id":789,
  "lat":0.34,
  "lng":32.5,
  "status":"in_progress",
  "driver_id":321,
  "eta_minutes":12
}
```

Photo uploads

- Accept `multipart/form-data` uploads on `POST /api/v1/uploads` returning `url` and `thumbnail_url`.
- Or accept photos directly in `POST /api/v1/donations` and `POST /api/v1/pickups/:id/status` as multipart fields.
- Backend should return compressed CDN URLs. Frontend will compress images before upload (target < 500 KB).

Upload response

{ "url":"https://cdn.example.com/abcd.jpg", "thumbnail_url":"..." }

Posts / Social sharing

- Simple endpoint if backend wants to persist posts:
    - `POST /api/v1/posts` — body: `{ "user_id", "donation_id?", "text", "shared_public": true }`
    - returns post object
- Frontend will also support platform-native share (no backend call required).

---

6. Validation & error format

- Use consistent error response structure:

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Validation failed",
    "details": {
      "title": ["Title is required"],
      "pickup_from": ["Invalid date"]
    }
  }
}
```

- 401 for unauthenticated, 403 for forbidden, 404 for not found, 409 for conflict, 429 for rate limiting.
- For create endpoints, if `client_id` duplicates server should return 409 with `existing_resource` object OR idempotent success.

---

7. Offline & idempotency contract

- **Client behavior**
    - Generate `client_id` UUID for any mutating action (create donation, pickup status) and persist action in local `outbox`.
    - Retry sync on network regained.
- **Server behavior**
    - Accept `client_id` as dedupe key. If operation already processed, return the existing resource with `200` (idempotent).
    - Return created resource with `server_id`.
- **Fields required for offline operations**:
    - `client_id` (UUID), `created_at` timestamp.

---

8. Tracking & UX details (how frontend will display)

- **Home screen variations**
    - Donor: quick create donation button, list of active donations, recent pickup statuses for their donations.
    - Receiver: nearby donations list (distance sorting), requests, "my requests", verification status.
    - Volunteer: list of assigned pickups with ETA + accept button.
- **Donation card**
    - Title, est kg, pickup window, distance, donor name, small photo thumbnail, CTA (request / cancel).
- **Tracking screen**

    - Map view centered on driver marker with polyline (if provided) and ETA.
    - Status timeline at bottom (pending → assigned → in_progress → picked → delivered).
    - Button to upload delivery proof (camera), confirm receipt (receiver enters staff name).
- **Delivery confirmation**
    - Required: uploaded photo, staff name, optional signature text.
    - On confirm: frontend posts to `/pickups/:id/status` with `status=delivered`+`photo`.

---

9. Notification types (frontend expects)

`donation.created` — for nearby receivers

{ "type":"donation.created", "donation_id":456 }

1.
2. `pickup.assigned` — to driver
3. `pickup.location` — driver → donor/receiver (realtime)
4. `pickup.status` — status update with `status` field
5. `organization.verified` — verification result for receiver
6. `message.system` — admin messages

Frontend will display notifications and also generate local in-app notification list.

---

10. Security, rate-limits & expectations

- All endpoints over HTTPS. Backend should force HTTPS.
- Tokens stored securely; use short expiry + refresh token or rotate tokens.
- Rate-limit location updates from drivers (e.g., 1 req/sec from each device). Backend should enforce and respond with `429` if exceeded.
- Limit uploaded image size (frontend will compress to <500KB) and dimensions (max 1920px).

---

11. Acceptance criteria for each major flow

1. **Login/Register**
    - Successful login returns user with `role` and `token`.
    - Token works on protected endpoints; 401 handled gracefully.
2. **Create Donation (online)**
    - Frontend uploads photo(s), gets back URLs, donation visible on `GET /donations`.
3. **Create Donation (offline)**
    - Donation queued with `client_id`. On sync, backend returns server id and status `available`.
4. **Request donation & pickup lifecycle**
    - Receiver requests donation ⇒ backend creates pickup and notifies volunteers.
    - Driver assignment triggers `pickup.assigned` push.
    - Driver can update location; donor & receiver see movement on map.
    - Photo proofs accepted and returned in pickup object.
5. **Delivery confirmation**

- Receiver confirms with photo + name; pickup `status` becomes `delivered`.
- Audit trail available in pickup record.

6. **Notification & push**
   - Frontend registers FCM token and receives push notifications for events relevant to the logged-in user.

---

12. Example OpenAPI-like snippet (reference)

Create donation (multipart)

```
POST /api/v1/donations
Headers:
  Authorization: Bearer <token>
Body (multipart/form-data):
  client_id: uuid
  title: string
  description: string
  quantity_est_kg: number
  pickup_from: ISO8601
  pickup_to: ISO8601
  lat: number
  lng: number
  photos[]: file
Response 201:
{
  "id": 456,
  "client_id":"uuid",
  "status":"available",
  "created_at":"..."
}
```

Update pickup status

```
POST /api/v1/pickups/{id}/status
Body (multipart or json):
  status: "picked" | "delivered"
  client_id: "uuid"
```

photo: file OR photo_url
staff_name: "..."

---

13. Developer notes & suggestions for backend team

- Provide a minimal OpenAPI / swagger with these endpoints and exact field names — that speeds integration.
- Return diff-friendly, minimal objects on list endpoints (avoid huge nested objects).
- Add query parameters for filters and pagination:
    - `?page=&per_page=`, `?lat=&lng=&radius_km=`, `?status=available`
- Provide CORS headers and mobile-friendly JWT tokens (long enough expiry for intermittent networks).
- Provide dev/test API key and a sandbox environment for the Flutter app.
- Provide sample test accounts for each role (donor, receiver, volunteer, admin).
- Provide a small `realtime` or `websocket` spec if backend supports sockets. If not, rely on FCM for real-time events and HTTP polling for location fallback.

---

14. Data privacy & audit

- Backend should store audit logs for pickup lifecycle and verification steps.
- Photo URLs should be time-limited or protected; include owner id/ACL metadata.
- Minimal personal info retention; provide endpoint to request deletion if needed.

---

15. Priority roadmap for API delivery (recommended)

1. Auth + `users/me` + push token endpoints (Sprint 2)
2. Donations create/list + upload endpoint + offline dedupe (Sprint 3)
3. Organizations registration + verification (Sprint 4)
4. Request/pickup lifecycle + notifications (Sprint 5)
5. Location updates + tracking UI support (Sprint 5)
6. Admin reports + audit logs (Sprint 6)

---

16. Example test cases the frontend will run during integration

- Create donation online → verify appears in GET /donations
- Create donation offline (store in outbox) → toggle network → sync → verify server id returned
- Request donation as unverified org → backend must reject (403)
- Assign driver → driver gets push; donor/receiver see status changes
- Driver sends location updates → donor map updates via FCM or polling
- Upload pickup & delivery photos → photos accessible via returned URLs

---

17. Appendix — small handy lists

Status enums

- donation.status: `available`, `requested`, `reserved`, `cancelled`, `completed`
- pickup.status: `pending`, `assigned`, `in_progress`, `picked`, `delivered`, `cancelled`

Date format

- ISO8601 UTC (e.g., `2025-10-20T07:00:00Z`) everywhere.

Image sizes

- Frontend compress to max 1920px longest side, target < 500 KB.
-