Lost & Found Item Tracker
Feature Roadmap & Use Cases

Version 1.0

1. User Authentication System

Allow users to register, log in, and secure their sessions.

- Actors: Guest, Authenticated User
- Use Cases:
  - Register with email, username, password
  - Login to receive access token
  - Logout or revoke token
  - View and update profile (name, phone, location)
- Dependencies: None (foundation)
- Time Estimate: 2–3 days
- JWT via SimpleJWT or token-based auth with djoser

2. Posting a Lost or Found Item

Users can submit a report for a lost or found item.

- Actors: Authenticated user (optional anonymous for "found")
- Use Cases:
  - Submit item with: name, category, description, image, location, date
  - Mark as “lost” or “found”
  - Upload photo
  - Edit or delete own item
  - Anonymous submission is **only allowed for 'found'** items (this is security-sensitive).
- Dependencies: Auth
- Time Estimate: 3–4 days
- Use ImageField with proper validation & upload handling

3. Public Item Browsing & Filtering

Anyone can browse/search lost/found items.

- Actors: Guest, Authenticated User

- Use Cases:
  - Search by item name, category, location, or type (lost/found)
  - Filter by date range
  - Sort by newest first or location proximity (basic)
- Dependencies: Item CRUD
- Time Estimate: 2 days
- Use DRF filters + django-filter
- Optional: If full-text search is used later, SearchVector or TrigramSimilarity may be added to the pipeline.

4. Geo-Based Radius Search (Production-Grade)

Enable users to find items within a radius from a geolocation (e.g., "Find all lost phones within 2 km of Dhaka University").

Actors: Guest users, Logged-in users

Use Cases

- User selects a point on map or enters location manually (converted to lat/lng)
- Selects a distance (e.g. 500m, 1 km, 10 km)
- API returns all matching items (lost or found) within that radius

Implementation Details

- Use **GeoDjango** with **PostGIS** for geospatial queries
- Model Item will include a PointField for geo-coordinates
- Use Distance() for annotation and then distance__lte as a filter.
- Model Requirement : You must explicitly state that location = models. PointField(geography=True) is required in the model.
- Optional: Integrate **reverse geocoding** (Google Maps or Nominatim) to fetch address from lat/lng

Dependencies: Item model, Basic Search/Filter

Estimate: 3 days

5. Matching System (Backend Logic)

System suggests matches between lost and found items.

- Actors: Authenticated user
- Use Cases:
  - If a user posts a lost item, system checks for found items with:
    - Same category
    - Keyword match (in name/description)
    - Date within ±3 days
    - Optional: location similarity
  - Return top 3–5 suggestions
- Dependencies: Item Model
- Time Estimate: 2 days
- Can use TrigramSimilarity, or simple fuzzy match on text

6. Contact / Claim Process

Enable secure communication between poster and interested user.

- Actors: Authenticated user
- Use Cases:
  - If match is found, user can request to contact poster
  - Poster accepts or declines
  - If accepted, contact info is shared (email/phone)
  - System logs the claim attempt
  - Use claim status field (pending, approved, declined) to manage flow cleanly.
- Dependencies: Match System, Auth
- Time Estimate: 2–3 days
- Optional future upgrade: chat system

7. Real-Time Chat System (via WebSockets)

Purpose: Allow secure, real-time communication between the finder and owner.

Actors: Logged-in users with a matched item

Use Cases

- User A finds a match and initiates chat with user B (poster of the item)
- Both users can send/receive messages in real time
- Only matched users can chat

- User can view chat history
- Admin can audit messages

Implementation Details

- Use **Django Channels** (WebSockets) for real-time chat
- Backend stack:
    - **Django Channels** (ASGI)
    - **Redis** for pub/sub layer
    - **PostgreSQL** (or separate store) for message persistence
- Authentication over WebSocket using JWT or session
- Optional: Add is_read flags and typing indicators

Message Model

```
class Message(models.Model):
    sender = models.ForeignKey(User, related_name="sent_messages", on_delete=models.CASCADE)
    receiver = models.ForeignKey(User, related_name="received_messages", on_delete=models.CASCADE)
    item = models.ForeignKey(Item, on_delete=models.CASCADE)
    text = models.TextField()
    timestamp = models.DateTimeField(auto_now_add=True)
    is_read = models.BooleanField(default=False)
```

Dependencies: Auth System, Matching System, Claim Request Feature

Estimate: 4–5 days

8. Mark as Resolved

Users can mark items as resolved (returned or recovered).

- Actors: Authenticated user
- Use Cases:
    - Mark item as "Resolved"
    - Remove it from search results
- Dependencies: Item CRUD
- Time Estimate: 1 day

9. Admin Panel / Moderation

Admin can flag, remove, or monitor reports.

- Actors: Admin
- Use Cases:

    - View all items
    - Flag/report abuse
    - Delete spam/inappropriate content
    - View resolution rate, popular categories
- Dependencies: All prior systems
- Time Estimate: 2 days
- Use Django Admin for Phase 1

10. API Documentation

Make the API easy to consume by frontend/mobile.

- Use Cases:
    - Generate Swagger/OpenAPI documentation
    - Ensure all endpoints are described
- Dependencies: All functional APIs
- Time Estimate: 1 day
- Use drf-spectacular or drf-yasg

Optional Phase 2 (Post-MVP)

These can be Gantt-planned separately later:

- Geo-based radius search (GeoDjango or custom bounding box)
- Email notifications on matched items
- File uploads with S3
- React/Flutter frontend

Summary Table (For Gantt)

No.	Feature	Time (Days)	Depends On
1	Auth System	3	-
2	Post Lost/Found Item	4	Auth
3	Public Search & Filter	2	Post
4	Geo-Based Radius Search	3	Post, Filter
5	Matching Suggestion Engine	2	Post
6	Contact/Claim Request	3	Matching, Auth

7	Real-Time Chat (WebSocket)	5	Claiming, Auth
8	Mark as Resolved	1	Post
9	Admin Panel	2	All
10	API Documentation	1	All