# 📱 Flutter Assignment: Request Handling Workflow Prototype

## 🎯 Objective

The goal of this assignment is to build a **prototype mobile application using Flutter** that simulates a real-world **request and confirmation workflow**. The app must support **two roles** with distinct functionalities:

- **End User** → Submits requests containing multiple items.
- **Receiver** → Reviews requests and confirms availability of items one by one.

The system must **track request statuses** (Pending, Confirmed, Partially Fulfilled) and handle **partial confirmations** by reassigning unconfirmed items.

---

## 👤 Roles & Features

### 1. End User

The End User represents a person making a request for multiple items.

**Features:**

- Create a new request by selecting items.
- Submit the request to the system.
- View submitted requests with their statuses:
    - **Pending** → Request submitted, awaiting receiver review.
    - **Confirmed** → All items confirmed by the receiver.
    - **Partially Fulfilled** → Some items confirmed, others reassigned.

- Track real-time progress of request status (without Firebase).

---

## 2. Receiver

The Receiver represents the entity reviewing and fulfilling requests.

**Features:**

- View all new requests assigned to them.
- Open a request and review it item by item.
- Confirm availability of items individually (e.g., mark as Available / Not Available).
- Submit confirmation results back to the system.
- If all items are confirmed → request status becomes **Confirmed**.
- If only some items are confirmed → request status becomes **Partially Fulfilled** and unconfirmed items are **reassigned**.

---

## ⚙️ General System Requirements

- **Backend or Mock Server**
    - A simple backend (Node.js/Express, Django, or mock JSON server) must be used to store requests and manage status updates.
    - Endpoints should include:
        - Create request
        - Fetch requests (per role)
        - Update confirmation status
- **Real-Time Updates (No Firebase)**

    - The app must show real-time updates when statuses change.
    - Acceptable approaches:
        - Polling at regular intervals
        - WebSockets (preferred if possible)
- **Authentication**
    - Basic login system to differentiate **End User** and **Receiver**.
    - Simple role-based authentication (no need for OAuth).
- **State Management**
    - Use a recommended solution like **Riverpod** or **Provider** to ensure clean, maintainable code.
- **Error Handling**
    - Handle edge cases like:
        - Network failure
        - Request submission errors
        - Receiver submitting incomplete confirmation
- **UI/UX**
    - Keep the interface **simple, minimal, and professional**.
    - Focus on clarity over design complexity.

---

## 📂 Deliverables

1. **Flutter Mobile Application**
    - Both End User and Receiver roles implemented.

    - Real-time updates working without Firebase.

2. **Backend / Mock Service**
    - Supports request creation, confirmation, status tracking.
    - Handles partial fulfillment (reassigning unconfirmed items).
3. **GitHub Repository** containing:
    - Complete **source code**.
    - **README.md** with:
        - Setup instructions (how to run backend + Flutter app).
        - Explanation of system design & approach.
        - API endpoints (if backend is implemented).
    - **Short demo video (2–5 min)** showcasing the full workflow (End User submits request → Receiver confirms → Status updates).

---

## 📝 Evaluation Criteria

- ✅ Implementation of both roles and full workflow.
- ✅ Correct handling of partial confirmations (confirmed vs reassigned).
- ✅ Real-time updates without Firebase.
- ✅ Clean, structured code with proper state management.
- ✅ Professional & minimal UI.
- ✅ Quality of documentation & clarity of demo video.

---

## 🔑 Key Features to Implement (Checklist for Intern)

- End User can create and submit requests with multiple items.
- End User can view request statuses (Pending / Confirmed / Partially Fulfilled).
- Receiver can view assigned requests.
- Receiver can confirm items individually.
- System updates request status (Confirmed / Partially Fulfilled).
- Unconfirmed items are reassigned.
- Real-time updates implemented via polling or sockets.
- Basic role-based authentication (End User vs Receiver).
- Error handling for failed network requests.
- Clear project documentation and demo video.