Perfect! Let’s break down a **detailed flow for a single-app self-service printing system** using **React Native CLI (mobile app)**, **Express.js (backend)**, and **MongoDB (database)**. I’ll include **file upload, payment, and printing**.

**1. System Components**

| **Component** | **Purpose** |
| --- | --- |
| React Native App | Runs on both owner and user devices. Handles file selection, payment, and connection. |
| Express.js Backend | Handles file storage, payment verification, user management, and communication with printer. Can run on owner device or cloud. |
| MongoDB | Stores user info, print jobs, payment status, and optionally session tokens. |
| Printer | Connected to owner device via USB or Wi-Fi. Receives print jobs from backend. |

**2. App Role Flow**

**2.1 Owner Mode**

1. Owner logs in and selects **Owner Mode**.
2. Owner device starts **local Express server** (or cloud backend) with:
   * Endpoint to receive PDFs: /upload-print-job.
   * Endpoint to verify payment: /verify-payment.
3. Owner generates **QR code** containing:
   * Server IP / temporary token.
   * Optional Wi-Fi Direct hotspot info if using P2P.
4. App waits for **incoming print jobs** from users.
5. When a user uploads a PDF:
   * Backend verifies payment.
   * Sends PDF to connected printer.
   * Marks job as completed in MongoDB.

**2.2 User Mode**

1. User opens app and selects **User Mode**.
2. User scans **Owner QR code** → connects to owner device:
   * Over **local hotspot** or **Wi-Fi Direct**.
   * Receives server URL / temporary token.
3. User selects file (PDF) using:
   * react-native-document-picker or expo-document-picker.
4. App calculates **printing cost**:
   * Options: B/W or color, number of pages.
   * Shows total amount to pay.
5. User pays via **UPI / Razorpay**.
6. On payment success:
   * App uploads PDF to backend endpoint (/upload-print-job).
   * Backend verifies payment.
7. Backend confirms upload → triggers **printing** on owner device.
8. User sees confirmation of print completion.

**3. Detailed Data Flow**

1. **Connection & Job Creation**

Owner device generates QR → contains backend URL + token

User scans QR → connects to backend

User selects file → backend generates a PrintJob record in MongoDB

1. **Payment Flow**

User selects print options → calculates price

User pays via Razorpay/UPI

Payment success → backend updates PrintJob in MongoDB

1. **File Upload & Printing**

User uploads PDF → backend stores file temporarily

Backend verifies payment → sends PDF to printer

Printer prints PDF

Backend marks PrintJob as completed → optional deletion of file

1. **Job Status & Security**

Each PrintJob has:

- Unique ID

- Payment status

- Temporary token / session

- Timestamp

**4. Backend (Express.js) Structure**

/routes

printJob.js // endpoints: /upload, /status

payment.js // endpoints: /verify

/models

PrintJob.js // MongoDB schema

User.js // MongoDB schema

/controllers

printController.js // file handling + printer logic

paymentController.js

**PrintJob Schema Example (MongoDB)**

const mongoose = require('mongoose');

const printJobSchema = new mongoose.Schema({

userId: String,

filename: String,

filePath: String,

pages: Number,

color: Boolean,

amount: Number,

paymentStatus: { type: String, default: 'pending' }, // pending, success

createdAt: { type: Date, default: Date.now },

printed: { type: Boolean, default: false }

});

module.exports = mongoose.model('PrintJob', printJobSchema);

**5. React Native App Structure**

/screens

OwnerScreen.js

UserScreen.js

FilePickerScreen.js

PaymentScreen.js

/components

QRScanner.js

PrintOptions.js

/utils

api.js // axios requests to backend

**OwnerScreen Flow**

* Start server / connect to printer.
* Generate QR code.
* Show list of incoming print jobs (optional).

**UserScreen Flow**

* Scan QR → get backend URL.
* Pick PDF → select print options.
* Pay → upload PDF.
* Show print status.

**6. Printer Integration**

* **USB printer**: Use Node.js package like printer or system commands (lp/lpr on Linux).
* **Wi-Fi printer**: Send PDF via IPP or vendor SDK.
* Optional: Delete file after print for security.

**7. Security Considerations**

* Temporary **tokens** in QR for secure connection.
* Verify payment **server-side** before printing.
* Limit file size.
* Delete files after printing.

**8. Optional Enhancements**

* Real-time job status using **Socket.IO**.
* Multiple users can connect via **Wi-Fi Direct or hotspot**.
* Allow owner to preview jobs before printing.

If you want, I can **draw a full flow diagram showing React Native app → Express → MongoDB → Printer** so you can see exactly how files, payments, and print jobs flow.

Do you want me to do that?