In PyCharm Community Edition, you can use Python with Django (for authentication) and Flask (for the main system). Since you're setting up a P2P network without internet, here are the best functions and technologies you can use:

1. **Django for Authentication** – Manage user login, roles, and permissions.
2. **Flask for Core System** – A lightweight framework to handle the main features (Client Listing, RFQs, and Purchase Orders).
3. **MySQL as the Database** – Store all company transactions and user credentials.
4. **Socket Programming** – If needed, you can use Python’s socket library for direct device-to-device communication.
5. **Local Network Hosting** – Use Flask/Django’s built-in development server or Gunicorn with Nginx to serve the system over the local network.

For offline usage, you need to configure a local database (self-hosted MySQL) and ensure devices connect through a local router (even without internet). Do you want help setting up the local network and database?

Great addition! You can implement this by:

1. **File Upload Feature** – Allow users to upload receipts, PDFs, and other documents.
2. **Storage Setup** – Store files either in a local folder (if fully offline) or in a dedicated file storage system.
3. **Database Link** – Save file paths in MySQL to associate them with the correct transaction.
4. **File Viewer** – Create a page where users can view or download attached documents.

Would you like help setting up the file upload and retrieval system in Flask/Django?

Your system already has a strong foundation, but here are some additional features that could enhance usability and efficiency:

**1. User Management & Roles**

* **Admin Panel**: Manage users, roles, and permissions.
* **Role-Based Access Control (RBAC)**: Restrict access (e.g., only managers can approve RFQs and POs).

**2. Notification System**

* **Email or Local Alerts**: Notify users when an RFQ is assigned, a PO is approved, or payment is due.
* **In-System Notifications**: A dashboard widget showing pending tasks.

**3. Advanced Search & Filters**

* **Dynamic Filters**: Search RFQs or POs by date, client, status, assigned employee, etc.
* **Sorting & Exporting**: Download reports as Excel or PDF.

**4. Approval Workflow**

* **Multi-Step Approvals**: Require manager approval before an RFQ or PO moves forward.
* **Status Tracking**: Track who approved or declined a request.

**5. Dashboard & Reports**

* **Overview Dashboard**: Graphs showing RFQs, pending POs, and completed transactions.
* **Financial Reports**: Summarize total revenue, pending payments, etc.

**6. Offline Sync & Backup**

* **Automated Backups**: Store a backup of the database locally or in an external drive.
* **Sync Feature**: If some users work offline, sync changes when they reconnect.

Would you like me to prioritize any of these features or suggest how to implement them?

Since you want to build your system entirely for free, here’s a full-stack solution using open-source tools:

**Backend (Server & Database)**

* **Python (Flask + Django)** – Flask for the main system, Django for authentication.
* **MySQL (MariaDB)** – Free alternative to MySQL for storing transactions and user data.
* **SQLite** – If you prefer a lightweight, local database.
* **Gunicorn + Nginx** – For serving the Flask/Django app in a production environment (optional).

**Frontend (User Interface)**

* **HTML, CSS, JavaScript** – Basic frontend technologies for building the UI.
* **Bootstrap or Tailwind CSS** – Free UI frameworks to make the system look good.
* **Jinja2 (Flask) or Django Templates** – For dynamic content rendering.

**File Storage & Management**

* **Local Storage** – Store uploaded files in a designated folder.
* **OwnCloud or Nextcloud (Optional)** – Self-hosted file storage if you want advanced features.

**Development & Deployment Tools**

* **PyCharm Community Edition** – Your main coding environment.
* **XAMPP or WAMP** – If you want an easy way to set up MySQL locally.
* **Apache or Nginx** – If you need a web server for local network hosting.
* **Git + GitHub (Optional)** – For version control and collaboration.

**Networking (For P2P Access)**

* **Local Router** – Devices connect to the same network to access the system.
* **Python’s socket Library (Optional)** – For direct communication between systems.

This setup will ensure your system runs on a local network without needing paid software. Do you need help with installation or setup?

**✅ Detailed System Development Checklist**

**1️⃣ Planning & Design**

**🎯 Define System Requirements**

🔲 Identify core features:

* Client Companies Listing
* RFQ Listing & Editor
* Purchase Order List
* File Upload & Management
* User Authentication & Security

🔲 Define **user roles & permissions**:

* **Admin**: Full access (manage users, RFQs, POs, clients).
* **Employee**: Can add/edit RFQs and POs but not manage users.

🔲 Sketch basic **wireframes/UI layouts** for pages:

* Login Page
* Dashboard
* Client Management Page
* RFQ Management Page
* PO Management Page

🔲 List required **technologies** (Flask, Django, MySQL, etc.).

🔲 Plan how employees will **access the system over P2P**.

**2️⃣ Setting Up the Development Environment**

**🛠 Install Required Software**

🔲 Install **PyCharm Community Edition** for development.  
🔲 Install **Python (latest version)**.  
🔲 Install **XAMPP or WAMP** for MySQL database (optional).  
🔲 Install **MySQL (MariaDB)** for database management.  
🔲 Set up **Git & GitHub** for version control (optional).

**⚙️ Set Up Python Virtual Environment**

🔲 Open terminal and run:

bash

CopyEdit

python -m venv venv

source venv/bin/activate # For Mac/Linux

venv\Scripts\activate # For Windows

🔲 Install necessary Python libraries:

bash

CopyEdit

pip install flask django mysql-connector-python flask-wtf flask-login

🔲 Create a **project folder structure**:

bash

CopyEdit

/project\_root

/backend (Flask APIs)

/frontend (HTML, CSS, JavaScript)

/database (MySQL)

/uploads (store uploaded files)

**3️⃣ Database Development**

**🛢 Set Up MySQL Database**

🔲 Open MySQL and create a database:

sql

CopyEdit

CREATE DATABASE company\_system;

🔲 Create tables:

**Clients Table**

sql

CopyEdit

CREATE TABLE clients (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255),

address TEXT,

contact\_person VARCHAR(255),

position VARCHAR(255),

email VARCHAR(255),

telephone VARCHAR(50),

mobile VARCHAR(50)

);

**RFQs Table**

sql

CopyEdit

CREATE TABLE rfqs (

id INT AUTO\_INCREMENT PRIMARY KEY,

date\_received DATE,

client\_id INT,

contact\_person VARCHAR(255),

item\_description TEXT,

assigned\_to INT,

bid\_closing\_date DATE,

proposal\_status ENUM('Yes', 'Purchase Order', 'Declined'),

remarks TEXT,

FOREIGN KEY (client\_id) REFERENCES clients(id)

);

**Purchase Orders Table**

sql

CopyEdit

CREATE TABLE purchase\_orders (

id INT AUTO\_INCREMENT PRIMARY KEY,

rfq\_id INT,

details TEXT,

status ENUM('Pending', 'Approved', 'Completed'),

payment\_details TEXT,

delivery\_details TEXT,

FOREIGN KEY (rfq\_id) REFERENCES rfqs(id)

);

**Users Table (Authentication)**

sql

CopyEdit

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) UNIQUE,

password\_hash TEXT,

role ENUM('Admin', 'Employee')

);

**4️⃣ Backend Development (Flask & Django)**

**🔹 Django for User Authentication**

🔲 Set up a Django project for authentication:

bash

CopyEdit

django-admin startproject auth\_system

🔲 Configure Django settings for MySQL.  
🔲 Create login, logout, and role-based authentication.

**🔹 Flask for Main System (Client, RFQ, PO Management)**

🔲 Set up Flask project:

bash

CopyEdit

mkdir backend && cd backend

flask --app main run

🔲 Create Flask routes for:

* **Clients (Add, Edit, Delete, List)**
* **RFQs (Create, Assign, Update, Delete, List)**
* **Purchase Orders (Create, Update, View, Delete)**
* **File Upload API for receipts, PDFs, etc.**

🔲 Implement authentication middleware to **secure routes**.

**5️⃣ Frontend Development (HTML, CSS, JavaScript)**

**🎨 UI Design & Features**

🔲 Design **Login Page** (connects to Django authentication).  
🔲 Create a **Dashboard** showing summary statistics.  
🔲 Implement **Data Tables** for Client, RFQ, and PO listings.  
🔲 Add **Sorting & Filtering Options**.  
🔲 Create **Forms** for adding RFQs and POs.  
🔲 Implement **File Upload Section** for purchase orders.

**🏗 Connect Frontend to Backend**

🔲 Use **AJAX or Fetch API** to send/receive data from Flask APIs.  
🔲 Implement dynamic content rendering with **Jinja (Flask) or Django Templates**.

**6️⃣ Security & Authentication**

🔲 Secure login & logout system.  
🔲 Use **hashed passwords** (bcrypt).  
🔲 Implement **role-based access control**.  
🔲 Prevent **SQL Injection & CSRF attacks**.

**7️⃣ Testing & Debugging**

🔲 Test each API endpoint using **Postman**.  
🔲 Test user authentication and role-based access.  
🔲 Verify **RFQ and PO creation, editing, and deletion** work correctly.  
🔲 Ensure file uploads work (test with PDF, receipts, etc.).  
🔲 Debug database errors and security vulnerabilities.

**8️⃣ Deployment (Local Network & P2P Access)**

**🔌 Setting Up Local Server**

🔲 Install **Gunicorn** and **Nginx** to serve the Flask/Django apps.  
🔲 Configure MySQL to allow **local network access**.  
🔲 Set static IP for **Flask server** (so employees can access it).

🔲 Run Flask & Django servers:

bash

CopyEdit

gunicorn -w 4 main:app

python manage.py runserver 0.0.0.0:8000

🔲 Test access from other computers in the local network.

**9️⃣ Backup & Maintenance**

🔲 Set up **automatic database backups**:

bash

CopyEdit

mysqldump -u root -p company\_system > backup.sql

🔲 Monitor server logs for errors.  
🔲 Plan for future upgrades (e.g., reports, notifications, analytics).

**✅ Final Review**

🎉 **System is ready for use!**

Would you like me to help you implement a specific section first? 🚀