Can you help me build a layout for the inventory management part of the project and give me a top down view of the sections i will have to code

Capstone Inventory System — Top-Down Architecture & Layout

> Scope: QR-driven inventory & supply ordering for campus printing (toner, waste baskets, parts), w audit history. Designed for Berea College printers (Toshiba MFPs, Lexmark, etc.).

---

1) Product Overview (Top-Down)

Goals

- Track on-hand quantities of consumables (by SKU), receive stock, decrement on issues, and keep a
- Let users scan a QR or visit a link to request supplies or report a device issue.
- Auto-notify staff when items fall below par; show a dashboard of what needs action today.

Primary Roles

- **Requester** (faculty/staff/student): submits supply requests & issue reports via QR web form.
- **Technician** (print team): views queue, fulfills requests, performs receiving, adjusts inventory.
- **Manager** (you): approves exceptions, sets thresholds, manages SKUs, locations, users.

Key Workflows

1. **Request**: User scans QR → prefilled device/location → selects item(s) and quantity → submits.
2. **Triage/Fulfill**: Technician sees new request → picks stock → delivers → marks fulfilled (deducts stoc
3. **Receiving**: Shipment arrives → receive against PO (optional) → adds to stock with lot/cost → audit e
4. **Reorder Signal**: Any SKU below threshold → appears in Reorder list → export/CSV/email to vendor.
5. **Issue Report** (optional for MVP): QR → device issue form → ticket created → status tracked; parts u

---

2) System Architecture (Layers)

```
[Client (Web + Mobile Web)]
  ├─ Public QR Forms (Request Supply, Report Issue)
  └─ Authenticated App (Dashboard, Inventory, Requests, Receiving, Admin)

[API Layer]
  ├─ Auth & RBAC (JWT or session)
  ├─ Inventory Service (SKUs, stock, transactions)
  ├─ Request Service (supply requests, fulfillment)
  ├─ Receiving Service (POs, receipts, lots)
```

```
  ├─ Notification Service (email/Teams webhook)
  └─ Device/Location Service (printers, buildings, rooms)

[Data Layer]
  ├─ PostgreSQL (core relational data)
  └─ Redis (optional: rate limits, queues, caching)

[Background Jobs]
  ├─ Low-stock scan & notifications
  ├─ Daily digest (what needs attention)
  └─ QR short-link generator (if used)
```

Tech Stack Suggestion

- **Backend**: Flask (or FastAPI) + SQLAlchemy + Alembic migrations
- **DB**: PostgreSQL
- **Frontend**: Minimal server-rendered templates (Jinja2) *or* React single-page app (SPA) if you prefer
- **Auth**: Flask-Login (sessions) or JWT (if SPA)
- **Infra**: Gunicorn/Uvicorn, Nginx (later); GitHub Actions for CI

---

3) Data Model (ERD-ish)

```
Users (id, name, email, role, is_active, created_at)
Roles: MANAGER, TECH, REQUESTER (public requests are guest records)

Locations (id, building, room, notes)
Devices   (id, cpo_tag, make, model, serial, location_id, status)

Skus      (id, sku, name, category, unit, color, vendor_id, is_active,
           reorder_point, reorder_qty, notes)
Vendors   (id, name, contact_email, phone, account_no)

InventoryLots (id, sku_id, qty_on_hand, unit_cost, lot_code, expiry_date, created_at)
Transactions (id, sku_id, lot_id, delta, type, ref_type, ref_id, user_id, created_at)
  types: RECEIVE, ISSUE, ADJUST+, ADJUST-, CONSUME

Requests  (id, device_id, requester_name, requester_email, status,
           submitted_at, fulfilled_at, fulfilled_by, notes)
RequestItems (id, request_id, sku_id, qty)

POs       (id, vendor_id, status, created_at)
POItems   (id, po_id, sku_id, qty, est_unit_cost)
Receipts  (id, po_id, received_by, received_at)
ReceiptItems (id, receipt_id, sku_id, lot_id, qty, actual_unit_cost)

Issues    (id, device_id, reporter_name, reporter_email, description,
           status, created_at, closed_at)
IssueParts (id, issue_id, sku_id, qty)
```

Notes

- Keep **Transactions** append-only for a complete audit trail.
- Use **InventoryLots** if you want FIFO and true unit cost history; otherwise keep a simple aggregat
- **Devices** tie QR codes to a location and give context to requests/issues.

---

4) API Surface (first pass)

Auth

- `POST /auth/login` (email+password) → session/JWT
- `POST /auth/logout`

Reference Data

- `GET /locations`, `POST /locations`
- `GET /devices`, `POST /devices`
- `GET /skus`, `POST /skus` (manager only)
- `GET /vendors`, `POST /vendors`

Inventory

- `GET /inventory/summary` (current on-hand by SKU)
- `GET /inventory/sku/{id}` (lots, transactions)
- `POST /inventory/adjust` (delta with reason) — manager/tech

Receiving

- `POST /pos` (optional for MVP)
- `POST /receipts` with items → creates lots + transactions(RECEIVE)

Requests (Supply)

- `POST /public/requests` (from QR)
- `GET /requests?status=new|in_progress|fulfilled`
- `POST /requests/{id}/fulfill` (deducts stock, adds transactions ISSUE)

Issues (Service) (optional MVP)

- `POST /public/issues`
- `POST /issues/{id}/close`
- `POST /issues/{id}/use-part` (consumes stock)

Alerts & Reports

- `GET /alerts/low-stock` (computed by reorder_point)
- `GET /reports/transactions?from&to`

---

5) UI Layout (screens & components)

A. Public QR Forms (no login)

- **Supply Request Form**
  - Header: Device (CPO-tag, model, location)
  - Items: multi-select SKU + quantity
  - Contact: name, email (optional), notes
  - Submit → success page with request ID
- **Issue Report Form** (optional)
  - Device context
  - Description, severity, optional photo upload (later)

B. Authenticated App

- **Dashboard**
  - Low-stock cards, New requests table, Today’s pick list, Recent transactions
- **Requests**
  - Table: status, device, items, submitted_at
  - Detail drawer: fulfill action (deduct), partials (later)
- **Inventory**
  - SKUs grid: on-hand, reorder point, vendor
  - SKU detail: lots, recent transactions, quick adjust
- **Receiving**
  - Receive items: choose SKU, qty, unit cost, lot code → commit
  - (PO pick list optional)
- **Devices & Locations**
  - Search by CPO/tag; view QR link, regenerate QR
- **Admin**
  - Users/Roles, Thresholds, Vendors, Export CSV

---

6) Permissions (RBAC)

- **Requester (public)**: create Requests/Issues only.
- **Technician**: view/fulfill requests, receive stock, adjust inventory (with reason), view reports.
- **Manager**: all technician rights + manage SKUs, thresholds, users, vendors; approve negative stoc

Route guards at UI level + server-side checks.

---

7) Inventory Math (single-source of truth)

- On-hand per SKU = Σ(lot.qty_on_hand)

- Transactions are append-only; on-hand is derived (and optionally cached on SKU for speed).
- Fulfillment decrements lots by FIFO (preferred) or by single aggregate if lots are disabled.

---

8) Notifications

- **Low-stock**: daily (and on change) email/Teams to Manager + Techs.
- **Request received**: email confirmation to requester (if provided).
- **Request fulfilled**: email to requester.

---

9) QR Code Strategy

- QR encodes a short URL: `/r?d=<device_id>&k=<nonce>`
- Server looks up device and renders prefilled form.
- Rotate nonces yearly; rate-limit public posts.

---

10) Folder Structure (Flask + Jinja2 option)

```
backend/
  app.py
  config.py
  requirements.txt
  /app
    __init__.py
    models.py
    schemas.py
    auth.py
    routes/
      inventory.py
      requests.py
      receiving.py
      devices.py
      admin.py
    services/
      inventory_service.py
      request_service.py
      receiving_service.py
      notify_service.py
    jobs/
      low_stock.py
    templates/  # if server-rendered
    static/
  /migrations  # Alembic

frontend/ (optional if React SPA)
  src/
    main.tsx
    App.tsx
    components/
    pages/
      Dashboard.tsx
      Requests.tsx
```

```
Inventory.tsx
Receiving.tsx
```

11) MVP vs. Stretch

MVP (2–4 weeks)

- Auth (Manager/Tech)
- SKUs & on-hand (simple aggregate)
- Public Request form + Request queue
- Fulfillment deducts stock + audit transaction
- Receiving adds stock + audit transaction
- Low-stock view (no emails yet)

Stretch

- Lots + FIFO costing
- Vendor/PO flows, exports
- Email/Teams notifications
- Issue reporting with parts usage
- Role self-service & SSO
- Barcode/QR label generation page

12) Test Plan (critical paths)

- Create SKU → Receive stock → Request item → Fulfill → On-hand decremented; transactions corr
- Low-stock threshold triggers once and clears after receiving.
- Public form spam controls (rate limit, CAPTCHA if needed).
- Access control: Requester can’t view internal pages.

13) Definition of Done (DoD)

- All user stories in MVP pass end-to-end tests.
- DB migrations reproducible; README covers setup.
- Basic audit report exports to CSV.
- Linting + type checks (mypy/pyright if using pydantic/FastAPI).

14) Initial User Stories (Backlog Seeds)

1. As a **Requester**, I can scan a QR and submit a supply request for the device I'm at.
2. As a **Technician**, I can see new requests, pick items, fulfill them, and auto-deduct stock.
3. As a **Technician**, I can receive a shipment and increase stock on selected SKUs.
4. As a **Manager**, I can set a reorder point per SKU and view a list of items below par.
5. As a **Manager**, I can export transactions to CSV for finance.

---

15) Sample Schemas

```
// POST /public/requests
{
  "device_id": 42,
  "items": [{"sku_id": 7, "qty": 2}],
  "requester_name": "Prof. Smith",
  "requester_email": "smith@berea.edu",
  "notes": "Need before 3pm"
}
```

```
// Fulfill
{
  "items": [{"sku_id": 7, "qty": 2}],
  "delivered_by": 3,
  "notes": "Delivered to office"
}
```

---

16) Coding Task Breakdown (Week 1–2)

1. Repo & scaffold (Flask, SQLAlchemy, Alembic, .env).
2. Models: Users/Roles, Locations, Devices, Skus, Inventory (aggregate), Requests, RequestItems, Tran
3. Basic routes: auth, skus (CRUD), requests (create/list/fulfill), receiving (add stock).
4. Jinja pages or minimal React with Requests/Inventory views.
5. Seed data + QR test page for a device.
6. Low-stock computation + dashboard cards.

---

17) Risks & Mitigations

- **Scope creep** → Strict MVP, defer POs/issues.
- **Data integrity** → Transactions append-only; DB constraints; service layer handles math.
- **Adoption friction** → Keep public forms fast and mobile-friendly; prefill device context.
- **Security** → CSRF on forms, rate limiting, server-side validation.

---

18) Nice-to-Haves Later

- Mobile "Pick & Deliver" flow with scanner support.

- Per-department cost centers; GL export mapping.
- SSO via Microsoft Entra ID.
- Attachment uploads for issues (photos of jams/errors).

You’re set. Build MVP against these models & routes first, then layer in lots/POs/notifications.

Absolutely, Zeke — I sketched the whole thing for you.

I dropped a full top-down architecture and screen/layout plan into a canvas you can build from (roles, DB schema, API routes, page flows, QR strategy, MVP vs stretch, and a week-1/2 coding checklist).

If you want, I can spin up a starter Flask repo structure next with the models and a couple of the routes stubbed out.