



Hey Zephyr, I would like some capstone project ideas. I am a senior computer science student and i was hoping to tie it to my position with our colleges printing services. An initial idea i had that i would also like you to include and build on is an inventory system that one can manage and also a way for users to scan QR codes or follow a link to order supplies for printers and each link and QR code would be specific to each printer. this has to be done in 14 weeks and rember i have other classes and responsibilities

Got it, Zeke — this is a perfect way to blend your **CS capstone** with your **real-world job as Campus Printing Equipment Fleet Manager**. Since you have 14 weeks and other classes, we'll need something that's ambitious enough for a senior capstone but scoped so it's realistic. I'll give you **project concepts (including your QR inventory idea expanded)**, plus notes on scope, technologies, and possible deliverables.

Capstone Project Ideas (Printing Services Focused)

1. Smart Inventory & Supply Reordering System (your idea, expanded)

- **Core Idea:** Each printer has a unique QR code/link that faculty/staff can scan to request supplies (toner, waste baskets, staples). Requests automatically log into a central dashboard.
- **Features:**
 - Admin dashboard: view current supply levels, outstanding requests, and usage history.
 - Inventory management: track toner levels, waste baskets, etc. (you could even integrate with FM Audit or PaperCut email alerts if you want).
 - QR Code Generator: system generates & maintains QR codes unique to each device.
 - Faculty/Staff Portal: simple “submit request” form via QR link.
- **Stretch Goals (optional):**
 - Auto-notifications (email/SMS) when inventory is low.
 - Analytics: see which devices consume the most toner, supply request patterns, etc.
- **Tech Stack Possibilities:**
 - Backend: Flask/Django (Python), Node.js/Express.
 - Database: PostgreSQL/MySQL or even lightweight SQLite if you want simple.
 - Frontend: React or just Bootstrap templates for simplicity.
 - QR Code: Python `qrcode` or Google Chart API.
- **Capstone Deliverables:** working system, demo video, database schema, QR label prototypes.

2. Printer Fleet Usage Analytics Dashboard

- **Core Idea:** A web dashboard that visualizes data from PaperCut MF or FM Audit (you already deal with these).
 - **Features:**
 - Pulls monthly usage reports (via CSV export or API, if available).
 - Graphs showing toner consumption, number of pages, duplex vs simplex printing.
 - Predictive model for when toner will run out (basic regression or time-series forecasting).
 - **Why Good:** Connects **CS data skills + your real job**. Could show sustainability impact (aligns with Berea's environmental goals).
 - **Scope:** manageable since data pipelines and visualization are well-documented.
-

3. Printer Service Ticket Automation System

- **Core Idea:** Faculty/staff scan a QR code → system auto-logs a service ticket (e.g., jam, low toner, error code).
 - **Features:**
 - Web form tied to each printer.
 - Tickets logged in a database with printer ID, issue, time.
 - Admin dashboard to see open/closed tickets.
 - Optional: auto-email to Ed or IS&S with details.
 - **Why Good:** Saves your team time, makes reporting printer issues structured and trackable.
-

4. "Follow-Me" Print Job Tracker (Lite Version)

- **Core Idea:** Build a lightweight dashboard that students/faculty can use to check print job status without going to the release station.
 - **Features:**
 - Simple API or script that checks PaperCut queue.
 - User login → shows active jobs, estimated wait time.
 - **Why Good:** Relates directly to your fleet role but focuses more on **user experience improvements**.
 - **Risk:** Might require PaperCut API integration (check feasibility first).
-

5. Sustainability & Cost Impact Tracker

- **Core Idea:** System calculates how much paper, toner, and money is being used, and shows potential savings from duplex printing or reduced color printing.
- **Features:**
 - Data viz dashboard (charts, comparisons).
 - "Impact reports" for departments (could even auto-generate PDFs).

- **Why Good:** Capstone aligns with **Berea's mission of sustainability.**

My Recommendation for You

Since you want something that **ties to your position + is doable in 14 weeks:**

➡ Go with **Idea #1 (Smart Inventory & Supply Reordering System)** as your **main project.**

- It's **tangible, directly useful, and scalable.**
- QR codes + ordering links are easy to demo.
- You can keep the core small (inventory + QR ordering), and only add analytics/automation if time allows.

Suggested 14-Week Timeline

- **Week 1-2:** Requirements gathering, tech stack selection, simple schema design.
- **Week 3-4:** Build QR code generator + database for printers/supplies.
- **Week 5-6:** Faculty/staff request form → logs requests into system.
- **Week 7-8:** Admin dashboard (basic inventory tracking).
- **Week 9-10:** Notifications (low inventory alerts, email confirmations).
- **Week 11-12:** Testing with a few real printers (pilot).
- **Week 13:** Final refinements, documentation, presentation prep.
- **Week 14:** Deliver + present capstone project.

⚡ Question for you, Zeke: do you want this system to be **just a prototype for the class** (works on your laptop, fake data), or do you want to **build something Berea Printing Services could actually use** after your capstone? That will help me scope tech stack and features properly.



counter question would it be feasible in that time frame to also have them report errors through the same form they would use to reorder items and just have it send out an email with the details to me or Ed and the hope i had would be that my job could use it after i leave but that is not required

Short answer: yes—totally feasible. You can use the **same QR→form** for both **supply requests** and **issue reports**, route them differently on the backend, and auto-email you/Ed with the details. Below is a tight, 14-week plan that still fits your class load and yields something Berea can keep using, even if you graduate.

Plan at a glance

- **MVP (Weeks 1–8):** Printers table + QR links → one form with a **Type** selector (“Order supplies” / “Report an issue”), requests saved in DB, confirmation page, and **email notifications**.
- **Polish (Weeks 9–12):** Simple **Admin Dashboard** (search/sort/filter, status changes), low-inventory rules, CSV export, basic auth/roles.
- **Pilot & Handoff (Weeks 13–14):** Run a small pilot (5–10 printers), fix bugs, finalize docs, handoff guide.

Architecture (simple + durable)

- **Backend:** Python (Flask or Django) or Node (Express). Keep it monolith.
- **DB:** PostgreSQL (printer, request_ticket, inventory_item, user tables).
- **Frontend:** Server-rendered pages (Jinja/EJS) or light React if you want.
- **Email:** SMTP (campus relay) or a service (e.g., Mailgun/SES). Send transactional emails on create/update.
- **Hosting:** One box (Linux VM) or managed PaaS (Railway/Render/Heroku-style). Static QR labels printed once.
- **Auth:** Campus SSO nice-to-have; start with admin login only. Submitters don’t need accounts.

Single form, dual purpose (UX)

- **Landing (via QR or short link):**
 - Shows printer name/location (from URL token).
 - **Radio:** “Order supplies” / “Report an issue”.
 - Conditional fields:
 - **Order:** supply type, quantity, urgency.
 - **Issue:** category (jam/quality/error code), description, photo upload (optional).
 - Contact name/email (autofill if SSO later).
 - Submit → writes to DB → shows thank-you.
- **Email routing:**
 - If Type = Order → subject **[SUPPLY] PR0101 – Black toner x1**
 - If Type = Issue → subject **[ISSUE] ST0102 – 2503 Paper Jam**
 - Body includes: printer ID, location, time, details, and link to admin page.

Minimal schema (start here)

- **printer**(id, asset_tag, location, model, qr_token UNIQUE, is_active)
- **request_ticket**(id, printer_id FK, type ENUM('SUPPLY', 'ISSUE'), payload JSONB, status ENUM('NEW', 'IN_PROGRESS', 'FULFILLED', 'CLOSED'), created_at, updated_at, requester_name, requester_email)
- **inventory_item**(id, name, sku, unit, min_threshold, current_qty)

- **user**(id, name, email, role ENUM('ADMIN', 'TECH'), password_hash or SSO_sub)

Admin dashboard (Week 9+)

- Table view with filters (type/status/location/date).
- Ticket detail page (notes, status changes, internal comments).
- **CSV export** for reports.
- **Rules:** email reminders for NEW > 48h; low-inventory alerts nightly.

Revised 14-week timeline (with error reporting)

- **W1:** Requirements, workflow map, data model, choose stack/hosting.
- **W2:** Project scaffold, DB migrations, seed printers, basic auth for admin.
- **W3:** QR token & label generator; printer detail page by token.
- **W4:** Unified submit form (conditional fields), validation.
- **W5:** Create tickets in DB; confirmation pages; basic list in admin.
- **W6: Email notifications** (NEW ticket), environment secrets, audit log.
- **W7:** Inventory basics (items, thresholds), order flow writes payload.
- **W8:** Usability pass; print QR sheets; hallway test with 1–2 printers.
- **W9:** Admin filters/search; ticket status updates; CSV export.
- **W10:** Low-inventory nightly check + email; issue categories.
- **W11:** Access controls (roles), basic rate limiting, attachment uploads.
- **W12:** Pilot (5–10 printers), collect feedback/metrics, bug fixes.
- **W13:** Documentation (runbook, admin guide, deployment), presentation deck.
- **W14:** Final demo; handoff to Printing Services (account + QR pack).

What makes it “capstone-worthy”

- Solid **systems design** (tokens, auth, email reliability).
- **Data model** and **ops** (migrations, seeding, backups).
- **Security** choices (signed QR tokens, input validation, minimal PII).
- Optional **analytics** module (requests per device, MTTR, supply burn rate).
- Thoughtful **handoff** (SOPs, admin guide, maintenance plan).

Risks & safeguards

- **Email deliverability:** test against campus spam filters; add reply-to.
- **Printer identity:** use **qr_token** not asset tag in URL; map server-side.
- **Scope creep:** keep PaperCut/FMAudit integrations as **stretch goal** (CSV import first, API later).

- **Sustainability:** pick hosting your department can own (campus VM) and write a 1-page deploy/runbook.

Stretch goals (only if time allows)

- Attach photo of error page; parse **error codes** for quick triage tips.
- Department-scoped dashboards and monthly PDFs.
- SSO for admins; requester autofill for faculty.

If this looks good, I can spin up:

1. a **detailed task breakdown** (issues checklist),
2. a **starter DB schema & migrations**, and
3. a **QR label template** you can print on Avery sheets.

Want me to assume **Flask + Postgres** and draft the initial project structure and schema next?



i dont know what flask + postgres is so if you could explain that first then i want you to do a deep check and make sure the main points of this project are feasable because something to also remeber is that i have not been in a coding academic setting in a year and have only done minor coding projects nothing huge like this before so my skills will be a little rusty

Hey Zeke! Here's the plain-English version, then a reality check on feasibility with guardrails for a 14-week capstone while you're a bit rusty.

What “Flask + Postgres” means (super simple)

- **Flask** — a tiny Python web framework. It lets you define web pages and APIs with very little code. Think: “routes” (URLs) → Python functions → HTML pages/forms.
- **Postgres (PostgreSQL)** — a reliable database where you store tables like **printers**, **tickets**, and **inventory**. It’s great for real apps and easy to back up.
- Why they’re often paired: Flask handles the **web**; Postgres handles the **data**. You talk to Postgres using an “ORM” (e.g., SQLAlchemy) so you can write Python instead of raw SQL most of the time.

Is your project feasible in 14 weeks? (Yes—if we scope it right)

Below is your feature list with a difficulty rating and a fallback if something takes longer than expected.

Must-have (core MVP)

1. **QR → One Web Form** that already knows the printer
 - **Do:** URL like `/p/<token>` loads printer name + location; radio buttons:

- “Order supplies” (supply type, qty)
- “Report an issue” (category, description, optional photo)
- **Store:** Create a `request_ticket` row in DB
- **Email:** Send an email with details to you/Ed
- **Difficulty: Medium** (forms, DB write, email)
- **Fallback:** Skip photo upload first; add later

2. Admin list of tickets (search/filter by status/type/location)

- **Do:** Simple table view, update status (NEW → IN_PROGRESS → CLOSED/FULFILLED)
- **Difficulty: Medium**
- **Fallback:** Start with a bare list + status dropdown; add filters later

3. Printer & QR management

- **Do:** `printers` table, unique `qr_token`, a page that outputs printable QR labels
- **Difficulty: Easy → Medium**
- **Fallback:** Generate QR codes offline with a script and store the PNGs

Nice-to-have (only if time remains)

- Inventory thresholds + low-stock email (Nightly job) — **Medium**
- CSV export for reports — **Easy**
- Attach photos to tickets — **Medium**
- Role-based access (Admin/Tech) — **Medium**
- PaperCut/FMAudit CSV import — **Medium** (API later if ever)

Non-goals (defer)

- Full SSO integration, deep PaperCut API work, fancy analytics/predictions. You can describe these as **future work** in your capstone paper.

Stack choice (given you're rusty)

You asked about Flask+Postgres; that's a fine choice. **However**, for speed and built-in admin tools, I strongly recommend:

Recommendation: Django (Python) + SQLite in dev, Postgres in prod

- **Why Django?** It gives you a production-quality **Admin UI out of the box**. You'll get CRUD pages for `Printer`, `RequestTicket`, and `InventoryItem` on day 1—huge time-saver.
- **Why SQLite first?** Zero setup while building. Flip to Postgres when deploying (Django makes that swap smooth).
- **Email:** Django's email utilities are straightforward.
- **Forms:** Django Forms handle validation + CSRF for you.

If you really want Flask, we can still do it—just expect a bit more work wiring up auth, admin, forms, and CSRF yourself (or via extensions like Flask-Admin, WTForms, Flask-Login).

Minimal architecture (works for Django or Flask)

URLs

- `GET /p/<qr_token>` → preloads printer info + shows one form (Order / Issue)
- `POST /p/<qr_token>` → creates `request_ticket`, sends email, shows thank-you
- `GET /admin/tickets` → list + filters (type, status, location)
- `POST /admin/tickets/<id>/update` → change status, add internal note
- `GET /admin/printers` → CRUD for devices, generate QR sheet

Tables

- `printer(id, asset_tag, location, model, qr_token UNIQUE, is_active)`
- `request_ticket(id, printer_id FK, type ENUM('SUPPLY', 'ISSUE'), payload JSON/text, status ENUM('NEW', 'IN_PROGRESS', 'FULFILLED', 'CLOSED'), requester_name, requester_email, created_at, updated_at)`
- `inventory_item(id, name, sku, unit, min_threshold, current_qty)` (*optional for MVP*)
- `user(id, name, email, role ENUM('ADMIN', 'TECH'), password_hash or SSO_id)` (*admin only; submitters don't need accounts*)

Security basics

- Use **opaque tokens** in QR URLs (e.g., 22–32 chars).
- Server looks up printer by token; never trusts user-supplied IDs.
- CSRF protection on forms; validate file uploads (if any).
- Keep emails minimal (printer info + request text); no sensitive data.

Project plan tuned for a 14-week semester (with ramp-up)

Week 1: Pick Django (or Flask), install, “Hello World”. Write a 1-page requirements doc + data model.

Week 2: Create models/migrations (`Printer`, `RequestTicket`). Admin working. Seed a few printers.

Week 3: `/p/<qr_token>` route that displays the printer and a single combined form (no upload yet).

Week 4: Save tickets to DB, thank-you page, basic validation.

Week 5: Email notifications on ticket create; environment secrets for SMTP.

Week 6: Admin: ticket list + status updates.

Week 7: QR code generation + printable label sheet (Avery layout).

Week 8: Hallway test with 2–3 printers (supplies + issues). Fix rough edges.

Week 9: Filters/search on admin table; CSV export.

Week 10: (Optional) Inventory min-thresholds + nightly email.

Week 11: (Optional) Photo upload for issues; size/type validation.

Week 12: Pilot 5–10 printers in different buildings. Gather feedback.

Week 13: Docs: README, runbook, handoff guide, security notes, limitations & future work.

Week 14: Final polish + presentation.

Built-in buffers: Weeks 8, 12, and 14 are intentionally lighter to catch up or stabilize.

"Rusty skills" risk plan (how we keep it manageable)

- **Use Django Admin** to avoid building an admin UI from scratch.
- **Start with SQLite** to remove database setup friction; switch to Postgres only if/when deploying to a server.
- **Defer uploads & inventory** until the core flow (QR → form → email + DB) is solid.
- **Feature flags:** Keep optional features behind simple booleans so you can cut scope without breaking things.
- **Definition of Done (MVP):**
 - For any printer with a QR, a user can submit a supply order or issue report.
 - You/Ed receive an email with details.
 - You can see and update ticket status in the admin.
 - You can generate/print the QR labels.

If you absolutely want a no-code first pilot (fastest validation)

- **Phase 0 (1-2 days):** Google Form per printer (prefilled link) → Google Sheet → Email notifications.
 - You'd prove the workflow on campus **this week**, then replace with your Django app for the capstone proper.
 - This can also serve as your baseline in the paper ("we replaced a fragile no-code solution with a maintainable system").

If you're good with **Django + SQLite (dev) → Postgres (deploy)**, I can immediately draft:

1. a tiny data model,
2. the URL plan, and
3. the exact steps to scaffold the project on your laptop.

Want me to generate that starter plan and commands now?



Can you also make a fallback plan because my boss said he will be talking to Toshiba soon who is a vendor and may be purchasing more services so in case there is overlap I need a fallback idea

That's smart thinking, Zeke — vendors like Toshiba sometimes roll out service portals that overlap with custom in-house tools. Having a **fallback plan** ensures your capstone is safe even if Toshiba suddenly

offers supply ordering or ticket tracking.

Here are three **backup project paths** (all still tied to your Printing Services role, but different enough from vendor services that they won't be redundant). Each is scoped for 14 weeks and works even if Toshiba adds overlap.

Fallback Project Ideas

1. Printer Usage & Sustainability Analytics Dashboard

- **Problem solved:** Right now, you (and Ed) get usage from PaperCut/FMAudit in raw reports. They're messy and not actionable. Toshiba may provide service, but not sustainability insights tailored to Berea.
 - **Core Features:**
 - Import CSV exports from PaperCut/FMAudit.
 - Visualize per-device, per-department trends (pages printed, duplex %, color vs. B/W).
 - Calculate estimated savings from switching to duplex or reducing color usage.
 - Generate simple PDF reports for supervisors or sustainability committees.
 - **Stretch:** Predict toner replacement dates using regression.
 - **Why safe from overlap:** Vendors rarely give **custom, sustainability-focused insights**. You're adding Berea-specific value.
-

2. Printer Fleet Health Tracker (Error & Maintenance Logging)

- **Problem solved:** Right now, errors get lost in email chains or word-of-mouth. Toshiba might handle repairs, but they don't track *campus-internal support workflows*.
 - **Core Features:**
 - Centralized system for logging printer issues (e.g., error codes, jams, fading color).
 - Each issue tied to printer, time, and resolution notes.
 - Dashboard for average time to resolution (MTTR), error frequency.
 - **Stretch:** Attach troubleshooting guides per error code (your knowledge base).
 - **Why safe from overlap:** Even if Toshiba takes service calls, Berea benefits from **internal tracking** (patterns, repeat offenders, accountability).
-

3. Toner/Waste Cartridge Lifecycle Optimizer

- **Problem solved:** You're managing toner and waste cartridges manually. Vendors may deliver supplies, but they won't optimize your local workflow.
- **Core Features:**
 - Track toner/waste cartridge swaps (date, person, printer).
 - Estimate lifetime per cartridge model.

- Show alerts if a cartridge died prematurely.
- Generate reorder lists based on actual campus consumption (not vendor estimates).
- **Stretch:** Link with QR sticker on cartridges → scan to mark installed/removed.
- **Why safe from overlap:** This is a **campus operations tool**, not a vendor-facing system.

Fallback Strategy

- **Primary (preferred):** Smart Supply/Issue Request System with QR links.
- **If Toshiba announces overlap (Weeks 1-4):** Pivot to **Fallback 1 (Analytics Dashboard)**, because it's easiest to start with CSV data and still ties neatly to your role + capstone requirements.
- **If Toshiba only partly overlaps:** Adjust scope so your project complements theirs — e.g., if Toshiba handles supply requests, you keep the **error-reporting + admin tracking** part as your core deliverable.

Next Step

When you present your proposal to your professor, you can frame it like this:

"My primary project is a QR-based supply and issue reporting system for our printer fleet. If our vendor Toshiba rolls out overlapping services during this semester, my fallback will be to pivot toward an analytics dashboard that imports PaperCut/FMAudit data and provides sustainability insights and printer health tracking. This ensures the project remains feasible, relevant, and unique to Berea."

Do you want me to **draft a one-page “capstone proposal doc”** with both the primary and fallback ideas spelled out, so you can give it to your professor and Ed right away?



something else my boos sugested that wont be taken over in the near future is a type of ticketing system that we would use for clothing orders as there are lots of different types of designs, material, sizes, etc would that sufficently show my skill or bascially be reinventing the wheel and not be a good project

That's a **really good suggestion from your boss**, and it's not "reinventing the wheel" if you frame it right. Let me break it down:

Clothing Order Ticketing System – Feasibility & Skill Showcase

Why It Can Work as a Capstone

- **Complexity:** Clothing orders are naturally more complex than toner reorders. You have multiple attributes (design, size, material, quantity, department) → this means relational database design,

form validation, and structured workflows.

- **Skills Demonstrated:**

- Database schema design (orders, products, variations, users).
- Form design with conditional logic (choose material → show available colors).
- Admin dashboard for tracking, updating, and fulfilling orders.
- Optional email notifications or status tracking for requesters.
- File uploads (mockups, logos).

- **Unique Value:** Toshiba (or any vendor) won't be stepping in here—it's a **campus-specific process**. You're filling a gap for internal workflow.

Why It Might Be Seen as "Reinventing the Wheel"

- Ticketing/order systems already exist (e.g., ServiceNow, Shopify, even Google Forms).
- If you just build a **basic "order form → email" flow**, it may look too simple for a senior CS capstone.
- To avoid that, you'll want to emphasize **custom tailoring** to Berea's needs:
 - Multiple clothing items (shirts, jackets, hats).
 - Multiple attributes (sizes, colors, embroidery options).
 - Potential for bulk orders by departments.
 - Reports (most ordered items, order history, by department).
 - Role separation (requester vs. approver vs. admin).

How to Make It Capstone-Worthy

If you go this route, frame it as:

"A lightweight, customizable order management and ticketing system designed for internal use at Berea College, where existing vendor solutions are too heavy or too generic."

Then include:

1. **Scalable Data Model:** Items, attributes, orders, status (Pending, Approved, Fulfilled).

2. **Admin Tools:** Search, filter, export orders.

3. **Workflow:** Submitter → Approval step (if needed) → Fulfillment.

4. **Stretch:**

- Barcode/QR tracking for large batches.
- Auto-generated order summaries (PDF/email).
- Analytics dashboard (popular sizes, most used designs).

That combination shows database design, workflow automation, and system design thinking — exactly what a capstone should demonstrate.

✓ My Recommendation

- Yes, it can definitely be a solid capstone if you emphasize **workflow + data management + Berea-specific needs** (not just a glorified Google Form).
- It's probably less risky than the QR printer supply project because there's no chance of Toshiba stepping on it.
- If you're worried about being rusty, this project might even be *easier to scope* (no PaperCut/FMAudit integration, no printer tokens, just order → ticket → track).

👉 Question for you, Zeke:

Do you want me to **line up both options (Printer QR/Ticketing vs Clothing Order Ticketing)** into a side-by-side comparison — strengths, risks, and feasibility — so you can pick which one to propose as your **primary project** and which as the **fallback**?



yes please line up both options

Perfect — let's put them side by side so you (and Ed or your professor) can clearly see which fits best as **primary vs fallback**.

⚖️ Capstone Project Options

Option A: QR-Based Printer Supply & Issue Ticketing System

(Primary Idea)

Core Concept

Each printer gets a unique QR/link → staff scan to submit either a **supply request** (toner, waste baskets) or an **issue report** (jam, error code). Requests flow into a central dashboard and email you/Ed.

Strengths

- Direct tie to your current job (Printing Fleet Manager).
- Real-world impact: could reduce email chaos and streamline ordering.
- Capstone-worthy tech:
 - QR code mapping to devices.
 - Database design (printers, requests, inventory).
 - Web form with conditional logic.
 - Admin dashboard for tracking.
 - Email automation.

- **Stretch:** inventory thresholds, CSV exports, error analytics.
- **Unique:** Most vendors won't provide this exact mix (especially error logging tied to your workflow).

Risks

- **Vendor overlap:** Toshiba may launch services that duplicate supply ordering or issue reporting.
- **Technical challenge:** You're rusty; building both frontend & backend might be a learning curve.
- **Maintenance:** If Berea adopts Toshiba's system later, your tool may be sidelined.

Feasibility (14 weeks)

- ✓ Feasible with Django/SQLite, focusing on **core flow** (QR → form → DB → email → admin view).
⚠ May need to cut stretch goals if you hit snags.
-

Option B: Clothing Order Management & Ticketing System

(Fallback Idea — but could be promoted to primary if vendor overlap hits)

Core Concept

Campus users submit clothing orders (shirts, hoodies, jackets, embroidery, etc.) with attributes like **design, material, size, quantity**. Orders are logged into a system with statuses (Pending, Approved, Fulfilled), viewable by admins.

Strengths

- **Safe from vendor overlap:** Toshiba won't touch this.
- **Complex enough for capstone:**
 - Multiple entities (orders, products, sizes, requesters).
 - Conditional form logic.
 - Workflow automation (status changes, approvals).
 - Admin dashboard with filters, reports.
 - Email notifications.
- **Flexibility:** Could expand later into other campus ordering systems (not just clothing).
- **Skills shown:** Database design, CRUD workflows, validation, reporting.

Risks

- **Perception:** Could look like "reinventing the wheel" unless you emphasize Berea-specific needs.
- **Engagement:** Less directly tied to your current daily role (not as obvious a link to Printing Services).
- **Data complexity:** More attributes/variants per item (sizes, materials, designs) → needs careful schema design.

Feasibility (14 weeks)

-  Also very feasible. Admin UI for orders/statuses + forms for submissions is straightforward in Django.
-  Trickiest part will be making the order form **intuitive** with many product variations.

Recommendation

- **Primary Proposal: Option A – QR-Based Printer Supply & Issue Ticketing System**
 - Best tie-in to your current role.
 - Shows applied skills in web + database + automation.
 - High impact on daily campus ops.
- **Fallback Proposal: Option B – Clothing Order Management System**
 - Ready to pivot if Toshiba overlaps.
 - Still capstone-level if you emphasize **workflow complexity + custom tailoring for Berea**.
 - Easier to keep relevant long-term since no vendor will replace it soon.

 In your capstone proposal doc, you can phrase it like:

"My primary project will be a QR-based printer supply and issue ticketing system. If our vendor Toshiba expands services that overlap with this idea, my fallback will be a clothing order management system, which also demonstrates system design, database modeling, and workflow automation skills but targets a different campus need."

Do you want me to go ahead and **draft that 1-page proposal document** with both options laid out (problem, solution, scope, feasibility, risks), so you can hand it directly to Ed and your professor?



I talked it over with my boss it would be much more cost effective for me to build it as through some digging we found at other universities it would cost \$10 per device per month and we have over 170 devices which would be rather expensive so can you expand option A again so that way i can write up a proposal

Got it, Zeke — that's actually great news. If Berea is looking at $\$10/\text{device/month} \times 170+\text{ devices}$ = **\$20,400 per year**, then your capstone solution has a **clear cost-savings justification** on top of being practical.

Here's a more **expanded version of Option A** you can lift into your proposal draft. I'll break it down into **Problem, Proposed Solution, Features, Benefits, Risks/Mitigations, and Timeline**.

Expanded Capstone Proposal:

QR-Based Printer Supply & Issue Ticketing System

1. Problem Statement

Currently, supply requests (e.g., toner, waste containers, staples) and printer issue reports (e.g., paper jams, print quality problems, error codes) are managed through informal channels such as emails, phone calls, or in-person conversations.

- These methods lead to **lost requests, delays, and inefficiencies**.
- Vendors like Toshiba offer similar services but at a **significant cost** (~\$10/device/month). With Berea's fleet of 170+ printers, this equates to **over \$20,000 annually**—an expense that can be avoided with an in-house solution.
- An internal, lightweight system tailored to Berea's needs would improve efficiency and reduce cost while giving Printing Services greater control over data and workflows.

2. Proposed Solution

Develop a **web-based ticketing and inventory request system** where each printer has a **unique QR code or URL**.

- Faculty/staff scan the QR code → open a simple form tied directly to that printer.
- They select either:
 1. **Order Supplies** (toner, wastebasket, staples), or
 2. **Report an Issue** (error type, description, optional photo).
- Requests automatically log in a **centralized database** and trigger **email notifications** to Printing Services staff.
- Admins (you, Ed, or future managers) use a secure **dashboard** to view, update, and track requests.

This solution would be **built in-house**, leveraging open-source technologies to avoid licensing fees.

3. Key Features

User Side (Faculty/Staff)

- Scan QR code → printer-specific form.
- Select request type (Supplies or Issue).
- Conditional fields (supply type & quantity, or issue category & description).
- Optional: attach a photo of error screen.
- Receive confirmation message (and optional email copy).

Admin Side (Printing Services Staff)

- Dashboard with **list of open requests** (sortable by printer, type, status, date).

- Status tracking (New → In Progress → Fulfilled/Closed).
- Ability to add notes to tickets (e.g., "Toner replaced 9/10").
- Search/filter tickets by building, request type, or date range.
- Export requests as CSV for reporting.

Additional Features (Stretch Goals, if time allows)

- Inventory tracking with automatic low-stock alerts.
- Analytics (usage patterns, most common supply types, frequent error codes).
- QR code label generator (for Avery label sheets).
- Multi-role access (Admin vs. Technician).

4. Benefits

- **Cost Savings:** Avoid \$20,000+ in annual vendor subscription fees.
- **Efficiency:** Streamlined, centralized requests reduce lost tickets and email clutter.
- **Accountability:** Clear logs of issues and fulfillment improve service tracking.
- **Scalability:** Works for 170+ devices now; can expand to new devices easily.
- **Sustainability:** Data collected can inform usage trends and support campus environmental goals (e.g., duplex printing adoption).
- **Longevity:** Built in-house; maintainable even after student turnover with proper documentation.

5. Risks & Mitigations

- **Risk: Vendor overlap in future**
 - *Mitigation:* System is campus-specific and flexible; if vendor services are added, Berea can still use this system for internal tracking or as a backup.
- **Risk: Developer skill ramp-up (student is returning after a year away)**
 - *Mitigation:* Use Django framework with built-in admin UI and rapid prototyping tools. Focus on MVP (core form + DB + email) first, then expand.
- **Risk: Maintenance after graduation**
 - *Mitigation:* Provide thorough documentation, database schema, and admin guide. Train at least one other staff member to use/update the system.

6. Proposed Timeline (14 Weeks)

(flexible but structured to ensure MVP is delivered early)

- **Weeks 1-2:** Requirements gathering, system design, choose tech stack, draft DB schema.

- **Weeks 3-4:** Implement printer database + QR code generator.
 - **Weeks 5-6:** Build ticket submission form (supplies/issues), tie into DB.
 - **Week 7:** Add email notifications (ticket creation → email to staff).
 - **Week 8:** Basic admin dashboard (view tickets, update status).
 - **Week 9:** Add filters/search, CSV export.
 - **Week 10:** Usability testing with 2-3 pilot printers.
 - **Week 11:** Expand pilot to 10 printers, collect feedback, fix bugs.
 - **Week 12:** Implement stretch goals (inventory tracking, attachments).
 - **Week 13:** Documentation, runbook, training guide for Printing Services staff.
 - **Week 14:** Final testing, presentation, and system handoff.
-

7. Technologies

- **Backend:** Python + Django framework.
 - **Database:** SQLite (development), PostgreSQL (production).
 - **Frontend:** Simple Django templates (Bootstrap for styling).
 - **Email:** SMTP relay through Berea College servers.
 - **QR Codes:** Python QR libraries (e.g., `qrcode`).
 - **Hosting:** Campus-provided VM or cloud provider (Heroku/Render/Railway).
-

8. Deliverables

- Fully functioning web application.
 - Printable QR code labels for printers.
 - Documentation package:
 - Technical setup guide (install, run, maintain).
 - Admin user manual (how to manage requests).
 - Capstone report with system design, testing results, and future work.
-

👉 This framing emphasizes **real cost savings**, **practical value**, and **capstone-level technical skills**.

Would you like me to now **draft the actual one-page proposal document** (formatted like something you'd turn into your professor), using this outline as the backbone? That way you'll have a ready-to-submit version you can polish with Berea's required formatting.



can you also give an expansion on this idea and how it wouldn't just be copying because it

would be more personalized for the college

Absolutely — this is an important point to stress in your proposal: you're not just **reinventing the wheel** but creating a **customized, lightweight, and cost-effective solution** that fits Berea's exact needs better than a generic vendor platform.

Here's an expanded section you can drop into your proposal (or adapt into the "justification" section):

Why This System Is More Than a Vendor Copy

While similar ticketing and supply-request systems exist commercially, this project is not a simple replication. Instead, it is designed to be **tailored specifically to Berea College's printing environment, workflows, and cost constraints**.

1. Cost-Effectiveness and Scalability

- Vendors like Toshiba provide cloud-based fleet management and ticketing services, but at a cost of **\$10/device/month**. For Berea's 170+ devices, this would exceed **\$20,000 annually**.
- The proposed system eliminates ongoing licensing fees and replaces them with a **one-time development cost**, creating immediate and long-term savings.
- The system is designed to be lightweight and can scale easily if new devices are added, without additional per-device costs.

2. Berea-Specific Customization

Unlike generic vendor tools, this system is designed around **how Berea College actually operates**:

- **Printer ID + Location Awareness:** Each QR code is tied to both the device ID and its building/room location, ensuring requests are instantly contextualized for staff without extra input.
- **Dual-Purpose Form:** A single form allows users to both **order supplies** and **report issues**—a flow designed around Berea's actual pain points, rather than vendor-standard workflows that separate them into different systems.
- **Campus-Specific Supplies:** The supply ordering form can be pre-populated with Berea's actual toner models, waste cartridge types, and staple cartridges, rather than generic supply categories.
- **Error Reporting in Plain Language:** Faculty/staff can select simple issue categories (e.g., "Paper Jam," "Print Quality," "Error Code") that map to Berea's internal triage processes, instead of vendor-level technical codes that many users don't understand.

3. Flexibility for Campus Processes

- **Internal Routing:** Requests can be routed directly to the Printing Services team (e.g., you or Ed), bypassing vendor support queues that may not reflect the college's immediate needs.
- **Future Integration:** The system can later incorporate **PaperCut/FMAudit data imports**, sustainability reporting, or even campus authentication (BCSecure), giving it the ability to evolve as Berea's needs change.

- **Departmental Reporting:** Data can be grouped by **academic department** or **building** to inform budgeting, resource allocation, and sustainability goals—something vendor tools rarely provide in a college-specific way.

4. Longevity and Maintainability

- Because this system is **built in-house**, it remains under Berea's control and can be updated as workflows, staff, or technologies change.
- Documentation and handoff materials will be included to ensure **future students or staff can maintain and expand** the system, preserving Berea's investment.

5. Educational and Mission Alignment

- This project demonstrates Berea's commitment to **cost stewardship** (saving \$20k+/year), **sustainability** (tracking supplies and reducing waste), and **student-led innovation** (a capstone that directly benefits the campus).
- It goes beyond a capstone assignment to become a **real, living tool** used by Berea College after the semester ends.

👉 In short: **Yes, there are vendor tools, but they are expensive, generic, and not tailored for Berea. This project provides a low-cost, campus-specific, and customizable solution designed to directly meet Berea's unique operational needs.**

Do you want me to **weave this expansion directly into the one-page proposal draft** I offered earlier, so you'll have a polished, professor-ready version that already addresses the "reinventing the wheel" concern?