BFB 321 Supply Chain Web Application Project – End-to-End Development Guide

Stack: Figma (wireframes) • HTML/CSS or Bootstrap (frontend) • Flask (backend) • MySQL/PostgreSQL (database)

Purpose. This document specifies the required steps, expectations, and marking rubric for delivering a professional, efficient, and scalable supply chain web application. Your work must align with the BFB 321 Proposal Guideline (problem background, problem statement, objectives, web-based solution framework, expected impact & sustainability, professionalism & referencing).

High-Level Deliverables (what you will submit)

- Figma wireframes (key screens + user flows)
- System design pack (ER diagram + API contract + architecture diagram)
- Implemented web application (HTML/CSS/Bootstrap + Flask + SQL schema/seed)
- Test pack (unit, integration, usability results)
- Deployment artefacts (env template, run scripts, DB migration scripts)
- Impact & sustainability analysis + professional documentation

Type	Task	Description
Semester Project	Wireframes (Figma)	UX for key flows; data inputs/outputs; role-based screens
	Frontend Build	Responsive HTML/CSS/Bootstrap; accessibility; forms & tables
	Data Model (SQL)	ERD, normalization, constraints, migrations & seed
Final Project (Exam)	API Design (Flask)	REST endpoints, auth, pagination, validation, error handling
	Integration & Features	End-to-end flows that solve the stated problem
	Testing & Quality	Unit/integration tests; usability; performance baselines

	Deployment & Ops	Runbook, .env template, DB migrations, logging
	Impact, Sustainability & Professionalism	Impact analysis, scalability and report quality

Step 1. Wireframes in Figma (User-centred UX)

What we expect

• Design role-based screens (e.g., buyer, warehouse clerk, supplier).

• Map primary flows: create order, receive shipment, update inventory, view dashboard.

• Show data inputs, validations, and feedback states (loading, empty, error).

Example tied to supply chain proposal

Agriculture input distribution: Figma screens for supplier portal to confirm dispatch dates, a buyer dashboard showing ETA risk, and warehouse inbound scheduling.

Build checklist

- Clickable prototype covering at least 3 key flows
- Annotations for data fields and validation

Step 2. Frontend Implementation (HTML/CSS/Bootstrap)

What we expect

• Responsive layout; accessible forms and tables; client-side validation hints.

• Reusable components for navbars, tables, and cards.

Example tied to supply chain proposal

Logistics delivery board: Bootstrap table with sortable columns for route, vehicle, ETA, and status badges; modal for updating delivery exceptions.

Build checklist

- Mobile-friendly pages for each main flow
- Form validations and helpful error messages
- Consistent typography and spacing scale

Step 3. Data Model (MySQL/PostgreSQL)

What we expect

• Create an ERD with keys, constraints, and indexes for read/write patterns.

• Normalize to 3NF where practical; add audit fields (created_at, updated_at).

• Plan migrations and seed data for demo.

Example tied to supply chain proposal

Manufacturing spare-parts: tables for suppliers, POs, PO lines, shipments, inventory locations, stock movements; indexes for lookups on SKU, location, and status.

Build checklist

- Schema.sql or Alembic/Flask-Migrate migrations
- Seed script (.sql or Python) with realistic demo data
- Performance indices (e.g., composite index on (sku, location))

Step 4. API Design with Flask

What we expect

• Define REST endpoints with clear request/response schemas (JSON).

• Implement auth (Flask-Login/JWT), pagination, filtering, and validation.

• Centralized error handling and logging.

Example tied to supply chain proposal

Education device tracking: endpoints for /devices, /assignments, /returns; role-based auth for teacher vs. admin; pagination for inventory listing.

Build checklist

- OpenAPI/Swagger JSON or README listing endpoints
- Blueprints structure, config by environment
- Input validation (pydantic/marshmallow) and error contracts

Step 5. Integration & Core Features

What we expect

- Wire the frontend to Flask endpoints (fetch/axios) and update DOM states.
- Implement role-based flows that directly address the problem statement.
- Metrics and KPI displayed on the frontend

Example tied to supply chain proposal

Returns management portal: customer NFC return check-in → warehouse inspection → disposition; end-to-end API calls with optimistic UI updates.

Build checklist

- At least 3 end-to-end flows fully working
- Empty/loading/error UI states covered
- Screenshots/GIFs in README

Step 6. Testing & Quality (Optional)

What we expect

• Unit tests (services, utils) and integration tests (API endpoints).

• Usability test with 3–5 users; capture findings and iterate.

• Performance baseline (DB query times).

Example tied to supply chain proposal

Inventory count: integration test covering POST /counts and GET /discrepancies; usability test reveals need for barcode input autofocus.

Build checklist

- pytest coverage report screenshot
- Locust/JMeter quick baseline or simple timing metrics
- Usability summary with 3+ findings and fixes

Step 7. Deployment & Operations (Optional)

What we expect

• Runbook to launch locally and in production; .env template; secrets hygiene.

• DB migrations and rollback plan; basic monitoring/logging.

Example tied to supply chain proposal

Small clinic deployment: Gunicorn + Nginx or Flask's production server behind reverse proxy; systemd service; daily DB backups; error logs retained 14 days.

Build checklist

- README runbook with exact commands
- .env.example (no secrets) and config.py per env
- Migration scripts and backup/restore notes

Step 8. Impact, Sustainability & Professionalism

What we expect

• Quantify impact vs. baseline; explain scalability (multi-site, multi-role).

• Professional packaging (structure, figures, captions).

Example tied to supply chain proposal

Warehouse congestion: report shows dock-to-stock time reduced 32%; discusses scaling the slotting algorithm to additional sites and integrating with TMS.

Build checklist

- Before/after KPI table with charts
- Turnitin similarity ≤ 15% noted

Minimal Technical Scaffold (recommended structure)

• Frontend: /static (css, js), /templates (Jinja2 HTML), base.html with Bootstrap CDN

• Backend: app/__init__.py (create_app), app/blueprints, app/models, app/schemas, app/services

• Config: config.py with environments (DEV/TEST/PROD), .env.example

- Database: Flask-SQLAlchemy + Flask-Migrate, seed.py to populate sample data
- Docs: README.md with runbook, OpenAPI/Swagger JSON, ERD image, Figma link
- Tests: /tests with unit and integration suites (pytest)