Fashion E-Commerce + AI Recommendations + Trend Analyzer + Simulation

Project plan (Shareable, ready to approve & start)

This document collects **everything we agreed** and turns it into a concrete, phased plan your group can review and accept. It covers the **database** part (detailed), **ML features** prioritized into phases, **data acquisition options**, **ETL / pipelines**, **MLOps & deployment** (phased), frontend notes, roles, timeline tied to course deadlines, deliverables, evaluation, risks and mitigations, and immediate next steps.

---

1 — One-line summary & product vision

Build a **mini fashion e-commerce platform** (DBMS) that supports customers & store admins **plus** an **ML layer** for personalized recommendations, demand forecasting, trend analysis, and a **simulation engine** that predicts business impact of discounts/influencers/new styles. Deliver a reproducible, production-style pipeline (ETL → experiments → model registry → container → API → monitoring).

---

2 — Key objectives (DB + ML + Business)

DB objectives

- Design a normalized relational schema (BCNF where applicable) for customers, products, orders, reviews, campaigns & influencers.
- Implement tables, views, stored procs; support INSERT/UPDATE/DELETE and transaction handling for orders.
- Provide queries covering joins, built-ins, nested queries, and at least one program/subprogram per user type (admin, customer).

ML objectives

- Build a working **recommendation engine** (collab & hybrid), **sales/category forecasting**, **customer segmentation**, **sentiment analysis** and **trend detection**.
- Build a **campaign impact (uplift) model** and a simulation UI for "what-if" scenarios.

- Experiment tracking & model versioning; containerize best models and expose via API.

MLOps objectives

- Automated ETL & preprocessing pipelines; experiment tracking (MLflow); CI/CD for model deployment; monitoring for data & model drift; automated retraining trigger.

Business objective

- Produce a tool small/medium fashion retailers can use to personalize storefronts, forecast demand, simulate campaigns and plan inventory.

---

3 — Minimum Viable Product (MVP) — What we must deliver first

(Enough to satisfy DB course and give a solid ML demo.)

DB MVP

- Relational DB implemented (Postgres/MySQL). Tables: Customer, Product, Category, Order, OrderDetails, Review, Campaign, Influencer.
- CRUD + transactions for Order placement that update stock atomically.
- ER diagram + normalization steps up to BCNF in report.
- 5–8 sample queries and at least one stored procedure per user type.

ML MVP

- Baseline collaborative filtering recommender (matrix factorization / ALS).
- Simple time-series forecasting per category (Prophet or ARIMA).
- Small web demo or notebook that shows recommendations on sample user and forecasting results.

Deliver this first. Stretch to hybrid recommender (add content features) if time permits.

---

4 — Full schema (tables & example fields)

(Implement these in DB. Attributes in parentheses are suggested columns.)

1. **Customer** (`customer_id` , name*, email*, gender, age, location, signup_date, style_tags)

2. **Category** (`category_id` , name, parent_category)
3. **Product** (`product_id` , category_id, name, brand, description, price, color, size_options, season, sku, stock, image_url`)
4. **Order** (`order_id` , customer_id, order_date, total_amount, payment_status`)
5. **OrderDetails** (`order_id` , product_id, quantity, unit_price`) — junction table
6. **Review** (`review_id` , customer_id, product_id, rating, review_text, created_at`)
7. **Campaign** (`campaign_id` , type, start_date, end_date, budget, discount_rate, details`)
8. **Influencer** (`influencer_id` , name, niche, reach, cost_per_post`)
9. **Engagement/Event** (`event_id` , session_id, customer_id, product_id, event_type[view/cart/purchase], campaign_id, timestamp`) — for behavior logs

*Store PII only if allowed; prefer anonymized IDs for ML.

---

5 — DB deliverables & requirements (explicit)

- Normalization documentation: FD lists, step-by-step 1NF→2NF→3NF→BCNF.
- ER Diagram (drawn in draw.io / ERDPlus) with PK/FK and cardinalities.
- SQL scripts: `create_tables.sql` , `create_views.sql` , `insert_sample_data.sql` .
- At least one view per useful admin report (top_sellers, daily_revenue, low_stock).
- Queries: user-facing (search products by filters), admin-facing aggregates, join examples.
- Transaction demo: place order → reduce stock → insert payment; commit/rollback flow.
- One subprogram per user: e.g., stored procedure to create order; admin stored proc to run campaign summary.

---

6 — ML Features prioritized (value → complexity)

We'll implement features across **Phases 1–4**. Each phase lists models, inputs, outputs, and evaluation metrics.

Phase 1 — Core ML (MVP) — deliver first

1. **Collaborative Filtering Recommender (ALS / MF)**

- Input: `customer_id` ↔ `product_id` interactions (purchases, add-to-cart, views weighted).
- Output: top-N recommendations per user.
- Metrics: Precision@K, Recall@K, NDCG.

2. **Category / Product Demand Forecasting (Prophet / ARIMA)**

   - Input: time series of sales per product/category.
   - Output: next-period sales forecast.
   - Metrics: MAPE, RMSE.

Phase 2 — Strengthening & Enrichment

3. **Hybrid Recommender (collab + content)**

   - Add product features (category, price, image embeddings).

4. **Customer Segmentation (KMeans/GMM)**

   - Input: RFM + category preferences => segments (VIP, occasional, bargain hunters).
   - Use for targeted campaigns.

5. **Category-level granular forecasting** (weekly/daily).

Phase 3 — Analytics & Trend detection

6. **Sentiment Analysis on Reviews (BERT / DistilBERT or VADER)**

   - Turn free text into sentiment scores. Aggregate by product/category.

7. **Trend Predictor** (combine sales velocity, sentiment, Google Trends / social signals)

   - Predict which categories/styles will trend next season.
   - Metrics: top-K trend recall (did predicted top k contain actual top k).

Phase 4 — Simulation & Uplift (unique selling point)

8. **Campaign Impact / Uplift Model (causal/uplift modeling)**

   - Input: historical campaigns (discount%, influencer type, reach, budget), pre/post sales, control vs treated groups (if available or synthetic).
   - Output: predicted uplift in conversions/sales for a proposed campaign.
   - Metrics: Qini / uplift AUC or uplift accuracy.

9. **Combined Simulation Dashboard**

    - Uses forecasting + uplift + inventory to compute projected revenue & ROI for parameter choices (discount, influencer, budget).

---

7 — Data: exactly what we need & prioritized acquisition plan

You will need **lots of data** for training. Below are prioritized options with short notes and sites/tools.

A — High-priority ready sources (quick wins)

1. **Kaggle datasets** — best for quick bootstrapping and experiments:
    - H&M Personalized Fashion (transactions + item meta) — great for recommenders & forecasting.
    - RetailRocket / E-commerce datasets (events, item properties).
    - Amazon product reviews (fashion subset) — review text + ratings.
    - DeepFashion / DeepFashion2 — image annotations for product images / attributes. *Where:* Kaggle, Github, Hugging Face datasets.

Action: Download these CSVs first to seed DB and ML.

B — APIs (real, legal, production-grade)

- **Shopify API** — pull product catalogs, orders (if you have a store or partner).
- **Etsy API** — boutique listings metadata.
- **Amazon Product Advertising API** — product metadata & images (affiliate key required).
- **Instagram Graph API / YouTube Data API** — influencer metrics & posts (requires app + permissions).
  When to use: When you can legitimately connect to a real store or partner.

C — Web scraping (when APIs not available)

- Scrape product pages, prices, reviews, influencers posts (tools: Scrapy, Playwright, Instaloader).
- **Caution:** check robots.txt and terms; anonymize/personally-identifiable info; prefer public metadata only.

D — Government & macro data (seasonality & exogenous signals)

- Retail indices / clothing sales time series (ONS, FRED, Eurostat) for seasonality signals.
- Google Trends (pytrends) for search interest over time on keywords.

E — Synthetic & generated data (for simulation & to boost volume)

- **Mockaroo** & **Faker** (Python) to create: customers, orders, campaign logs, event streams. Use to build behavioral logs, synthetic campaigns, and to ensure sufficient interactions per user/product.

F — Labeling / annotation (if you need attributes)

- For image attributes (pattern, fabric): use DeepFashion prelabels and refine via Labelbox or Mechanical Turk.

Recommended approach (practical):

1. Seed DB with Kaggle datasets (H&M, RetailRocket, Amazon reviews, DeepFashion) — immediate.
2. Fill gaps with synthetic data generated to match patterns.
3. Enrich later via APIs or polite scraping of a few boutique sites if more specific data needed.
4. For social signals, query Google Trends and optionally Instagram (Graph API) for a small list of influencers.

---

8 — ETL / Data engineering (automation & pipelines)

Design principles

- Keep raw data immutable; store raw files in `raw/` (S3 / local folder).
- Build reproducible ETL notebooks / scripts that produce canonical tables in DB.
- Use DVC or simple versioned folders if DVC is heavy.

Suggested stack

- Orchestration: **Airflow** or **Prefect** for scheduling ETL jobs.
- Processing: Python (pandas), pyarrow for Parquet.
- Feature store (optional): **Feast** for reuse; else store embeddings/features in Parquet / Postgres.
- Storage: Postgres/MySQL for relational canonical data; S3 or local `data/` for large files and image storage.

ETL steps

1. Extract: download CSVs / call API / run scrapers.
2. Validate: Great Expectations checks (schema, missing values).
3. Transform: normalize categories, map brand names, generate event logs (sessionize), compute RFM features.
4. Feature extraction: compute product image embeddings (CLIP/ResNet), text embeddings (Sentence-BERT).
5. Load: store canonical tables and feature artifacts (Parquet + model artifacts).

Automation

- Nightly ingestion of new simulated/scraped data; nightly or weekly feature refresh.

---

9 — MLOps & deployment — phased plan

Phase A — Basic (do for MVP)

- **Experiment tracking:** MLflow to log experiments and models.
- **Model registry:** register best model with version, notes and metrics.
- **Containerize best model:** Docker image that serves predictions via FastAPI endpoints `/recommend` and `/forecast`.
- **Batch job**: daily recompute top-N recommendations and store in DB view.

Phase B — Hardening & Monitoring

- **Serve:** Deploy Docker on a VM or simple Docker Compose.
- **Cache:** Redis for hot recommendations.
- **Monitoring & logs:** Prometheus + Grafana for infra; API logs to a file + Sentry for errors.
- **Model monitoring:** EvidentlyAI / custom scripts to detect data, feature or prediction drift.
- **Retraining trigger:** When drift or metric degradation > threshold, schedule training DAG in Airflow.

Phase C — Production / scalable (stretch)

- **Kubernetes + Helm** for scalable services.
- **CI/CD**: GitHub Actions + docker builds + deploy to k8s.
- **Canary rollout:** deploy new model to subset of users for A/B.
- **Vector search:** FAISS / Milvus for nearest neighbor on embeddings (visual search).
- **Realtime streaming:** Kafka for event ingestion & near-real time updates.

---

10 — Frontend — brief blueprint

Two main UIs

1. **Customer UI** (simple web / Streamlit for prototype): product browse, product page with recommendations, add to cart, checkout (simulate payments), user reviews.

2. **Admin Dashboard**:

    - Inventory & product CRUD.
    - Campaign management UI (create discounts / influencer campaigns).
    - Trend & forecast panels (charts: sales over time, forecasts, top trending categories).
    - Simulation UI: sliders for discount %, influencer reach, and "Run Simulation" to produce predicted revenue & ROI.
      Tech choices: React or plain HTML/CSS/JS for front-end; Streamlit for fast ML dashboards; or simple templates in Flask/FastAPI.

---

11 — Evaluation & success metrics

Recommender

- Offline: Precision@10, Recall@10, NDCG@10.
- Business proxy: click-through rate on recommended items (if demoing with traffic).

Forecasting

- RMSE, MAPE on holdout period.
- Business metric: inventory stockouts avoided (simulated).

Segmentation

- Silhouette score; use segments for targeted campaign and measure uplift.

Simulation

- If historical campaigns exist, measure prediction error; otherwise sanity checks and qualitative business evaluation.

DB

- Correctness (FK constraints satisfied), atomic transactions working, views & stored procedures executed as expected.

---

12 — Team roles & responsibilities (suggested)

- **Project Lead / DB owner** — schema, ERD, normalization docs, SQL scripts, final report (who will submit).
- **Backend / ETL engineer** — ETL pipelines, data ingestion, DB load scripts.
- **ML Engineer** — models (recommendation + forecasting), MLflow experiments, model evaluation.
- **MLOps / DevOps** — containerization, basic deployment, monitoring hooks, experiment registry.
- **Frontend / UX** — small web UI + admin dashboard + simulation UI.
- **QA & Documentation** — test cases, screenshots for report, prepare final zip.

(Adjust roles to group size; a person can hold multiple roles.)

---

13 — Semester timeline (mapped to course deadlines)

Course deadlines: **Proposal by end of Week 4**, **ER by Week 12**, **Full Project by Week 15**. Below is a practical weekly plan for 15 weeks.

Week 1 (kickoff)

- Finalize idea & scope; assign roles. Start proposal doc.

Week 2

- Design DB schema draft; list required datasets; start downloading Kaggle seeds.

Week 3

- Prepare proposal (intro, functions, tech stack). Mock sample data using Faker/Mockaroo.

Week 4 — Proposal due

- Submit proposal + brief sample ERD.

Week 5–6

- Implement canonical DB (create tables, load initial data). Build CRUD + transactional order flow.
- Seed experiments: baseline ALS recommender on seeded data.

Week 7–8

- Build and log baseline forecasting model. Start MLflow tracking.
- Implement front-end prototype (customer browse + simple admin view).

Week 9–10

- Add Reviews table + sentiment pipeline (basic). Implement hybrid recommender (content + collab).
- Add views for admin reports.

Week 11–12 — ER due (end of Week 12)

- Finalize ERD & normalization docs (screenshots).
- Prepare mid-project demo (recommendations & forecasts working).

Week 13

- Build simulation engine prototype: simple uplift/regression model (synthetic campaign experiments).
- Integrate simulation UI.

Week 14

- Containerize model(s) and API endpoints (FastAPI + Docker). Add basic monitoring and logs.

Week 15 — Full project due

- Final report, screenshots, code zip, demo video, and prepare presentation.

(Stretch goals: Airflow pipelines, drift detection, k8s — do if time allows after Week 12.)

---

14 — Deliverables (what you hand in)

1. Project proposal (submitted Week 4).
2. ER Diagram + normalization documentation (Week 12).

3. Full zip (Week 15) containing: SQL scripts, sample data, ETL scripts, notebooks, MLflow exported models, Dockerfiles, frontend code, README (how to run), screenshots of all features.
4. Short demo video (5–10 minutes) showing user & admin flows and ML outputs.
5. Final report (Word) documenting every step, roles, and conclusions.

---

15 — Risks, constraints, and mitigations

- **Data shortage / quality** → Mitigate by starting with Kaggle + synthetic generation; validate with Great Expectations.
- **Legal issues (scraping / PII)** → Avoid harvesting PII; prefer APIs / public datasets; anonymize user data.
- **Compute limits for ML** → Use transfer learning & small batch sizes; use pre-trained models (CLIP, Sentence-BERT).
- **Time constraints** → Prioritize Phase 1+2 only for class; treat Phase 3+4 as stretch goals.
- **Team bandwidth** → Share responsibilities, keep weekly mini-deadlines.

---

16 — Immediate next steps (actionable checklist for your group)

1. **Approve this plan** and assign roles (who is DB lead, ML lead, ETL, frontend).
2. **Download seed datasets** (Kaggle: H&M, RetailRocket, Amazon reviews, DeepFashion) and place in shared repo.
3. **Create repo + branch structure** (code, data, notebooks, docs). Add README & coding conventions.
4. **Implement DB schema** locally and load sample data (Week 5 target). Produce initial ERD screenshot.
5. **Build baseline ALS recommender** on seed data and log experiments to MLflow (Week 6 target).
6. Prepare the **proposal** (use content from this doc) for submission at Week 4.

---

17 — Helpful tech stack suggestions (short)

- **DB:** PostgreSQL (or MySQL)
- **Backend / ETL:** Python, Pandas, SQLAlchemy, Airflow/Prefect

- **ML:** scikit-learn, implicit (ALS), Prophet/ARIMA, PyTorch or TensorFlow for deep models; Sentence-BERT / CLIP for embeddings
- **Experiment tracking:** MLflow
- **Container & serving:** Docker, FastAPI
- **Monitoring:** Prometheus + Grafana; EvidentlyAI for drift
- **Feature store / vector DB (optional):** Feast + Milvus/FAISS
- **Frontend prototyping:** Streamlit or React + Bootstrap

---

18 — How I can help next (I'll do it now if you want)

I can produce any of the following immediately:

- `create_tables.sql` + example normalization steps (1NF→BCNF) for your DB.
- A small notebook that ingests H&M or RetailRocket CSV and runs ALS recommender, logs to MLflow.
- A Dockerized FastAPI template that serves recommendations from a saved model.
- ER diagram image and a one-page proposal doc you can submit.

Tell me which item you want first and I'll produce it right away.

If your group agrees to this plan I'll help convert it into the **final proposal document** (word file) and produce the initial SQL/Notebook artifacts so you can submit by Week 4.