✅Assignment – Backend System Design

🧠 Objective

Design a **store and inventory system** for a gaming platform that supports:

- In-game purchases
- Digital and physical rewards
- Inventory management
- Character customization

Your goal is to design the **relational database schema**, **modular backend architecture**, and outline **optimizations** for scale and performance.

📄 Functional Description

You're designing the backend for the **BattleBucks Store System**.

🎮 Platform Overview:

BattleBucks is a gaming platform where users:

- Earn or buy **gems** (virtual currency)
- Use gems to purchase **store items**, which include:
    - **Game-specific skins**
    - **Platform-wide skins**
    - **Digital rewards** (e.g., voucher codes, coupons)

    - **Physical merchandise** (e.g., BB merch, brand collabs)
    - **Utility items** (e.g., name change cards, extra lives)

🎮 Game Integration:

- Platform supports multiple **games**, each with their own **game items** (e.g., gun skins, emotes)

🧾 User Entities:

Users have:

- A **wallet** (gems balance)
- A **purchase history**
- An **inventory** of owned items (consumables & non-consumables)
- One or more **character profiles** (custom avatars with equipped items)

🛍️ Store Item Types:

- **Consumable** (e.g., utility cards)
- **Non-consumable** (e.g., skins)
- **Delivery Types:**
    - In-game inventory
    - Email (e.g., voucher delivery)
    - Shopify (physical goods)
    - Functional (e.g., apply gems/credits instantly)

🧩 System Must Support:

- Inventory tracking and item usage
- Item equipping to character profiles
- Fulfillment tracking of digital and physical items
- Extensibility (e.g., tournament rewards, brand collabs)

⚙️ API & Performance Optimization (Required)

Assume the system will support:

- **1 million+ users**
- **100,000 daily purchases**
- **10,000 concurrent users**

Answer the following:

1. **Where would you use caching?**
   - (e.g., storefront listings, user inventory, character profiles)
2. **Where would you use background jobs / queues?**
   - (e.g., digital delivery, email sending, Shopify sync)
3. **How would you ensure purchases are transactional and idempotent?**
   - (e.g., duplicate purchase protection, retries, locks)
4. **How would you optimize high-traffic APIs** like:
   - `GET /store/items`
   - `GET /user/inventory`

5. Consider:
    - Pagination & filtering
    - Query batching
    - Data shaping / lightweight responses
    - Read replicas or caching
6. **What database indexing strategies** would you use?
7. **How would you implement rate limiting / abuse prevention?**
8. **Would you use preloading / eager loading / denormalization** for certain flows?

🧾 What to Deliver

Please submit the following:

1. Schema Design Diagram

- Use any tool: drawSQL, dbdiagram.io, Lucidchart, etc.
- Show all tables and relationships (FKs, cardinality)

2. Written Explanation (1–2 pages max)

- Explain key entities (users, store items, purchases, inventory, character profiles)
- Explain how you handle:
    - Delivery types
    - Consumables vs non-consumables
    - Equipping logic
    - Multi-game support and item compatibility

3. (Optional) System Architecture Diagram

- Describe or illustrate how you'd modularize:
    - Store
    - Inventory
    - Purchase flow
    - User & Character Profiles
- Include queues or async workers if applicable

4. Bonus (Optional)

Answer any of the following:

- How would you scale the platform to handle 1M users and 100K purchases/day?
- How would you optimize gem deduction and purchase flow for transactional safety?
- How would you handle Shopify order tracking and webhooks?