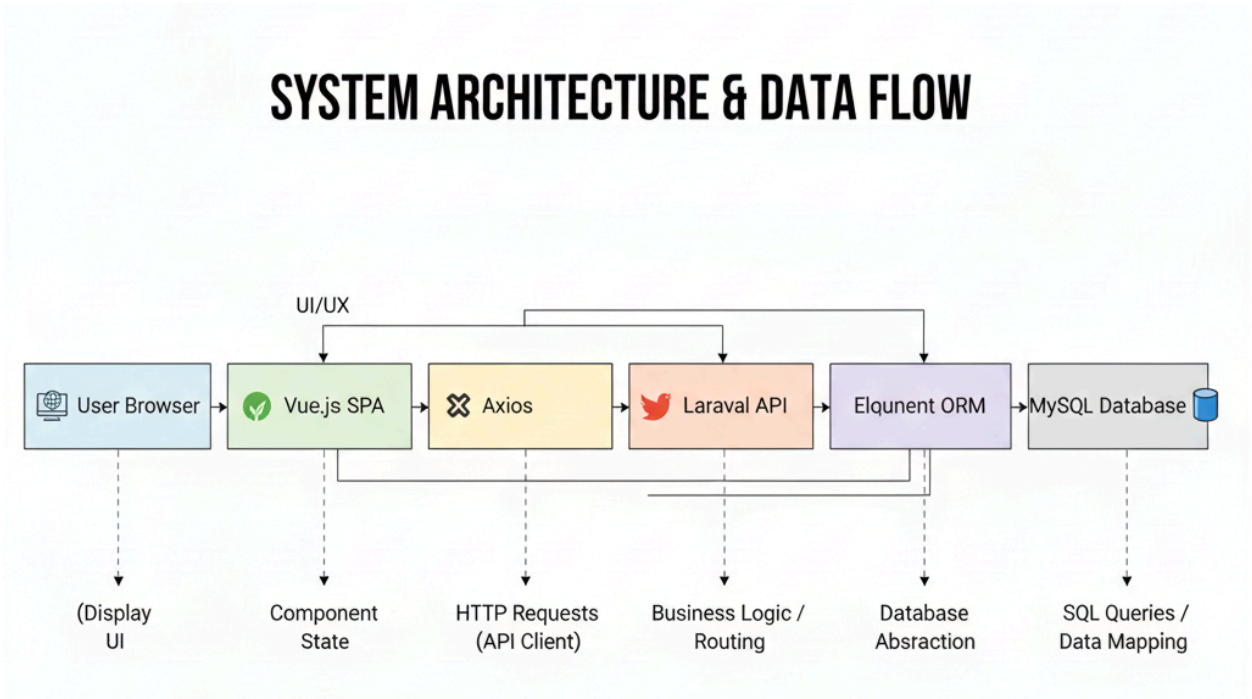
- Components such as POS.vue, Inventory.vue, Sales.vue, and Dashboard.vue render dynamic interfaces using Vue Router for navigation.

2. **API Communication Layer (Axios HTTP Client):**
   - Vue.js components make asynchronous HTTP requests to the Laravel backend using Axios.
   - All requests include an authentication token (Laravel Sanctum) in the header for secure access.
3. **Application Logic Layer (Laravel Backend):**
   - Requests are routed through Laravel's API routes (routes/api.php) to dedicated controllers (e.g., SalesController, InventoryController).
   - Laravel handles business logic: validating input, calculating totals, updating inventory, and enforcing business rules.
4. **Data Persistence Layer (MySQL Database):**
   - Laravel Eloquent ORM interacts with the structured MySQL database.
   - Key tables include users, products, sales, sales_items, inventories, suppliers, and inventory_logs, which maintain relational integrity and audit trails.
5. **Response Flow:**
   - After processing, Laravel returns a JSON response (success/error) to the frontend.
   - Vue.js updates the UI state reactively, providing immediate feedback (e.g., updated stock levels, new sale confirmation).

High-Level Architecture Diagram Concept:



Tools and Technologies Used

- **Frontend:** Vue.js 3, Vue Router, Vuex/Pinia (for state management), Axios, Tailwind CSS/Bootstrap (for styling)
- **Backend:** Laravel 10.x, Laravel Sanctum (API authentication), Eloquent ORM
- **Database:** MySQL 8.0
- **Development Tools:** Composer, npm, VS Code

b. Backend Implementation

Database Structure

The database is normalized across 10 core tables to ensure data integrity, avoid redundancy, and support auditability:

- **users**: Stores administrator credentials and profile (linked via id to sales.ClerkID and inventory_logs.created_by).

- **categories & suppliers:** Reference tables for product categorization and supplier details.
- **products:** Central product catalog with CostPrice, SellingPrice, ReorderLevel, and foreign keys to categories and suppliers.
- **inventories:** Tracks real-time QuantityOnHand per product, updated transactionally with each sale or stock adjustment.
- **sales & sales_items:** Master-detail tables recording each transaction with TotalAmount, DiscountAmount, PaymentMethod, and links to customers and users.
- **inventory_logs:** Audit trail for all stock movements (stock_in, stock_out, adjustment) with timestamps and user references.
- **transactions:** Financial ledger capturing cost and revenue impact per product movement.
- **purchase_records:** History of procurement from suppliers.

API Integration and Authentication Logic

- **API Endpoints:** RESTful endpoints under /api/ namespace (e.g., /api/products, /api/sales, /api/inventory).
- **Authentication:** Laravel Sanctum provides token-based authentication. Upon login, the backend issues a secure API token stored in the frontend's HTTP-only cookies or authorization headers for subsequent requests.
- **Authorization:** Middleware ensures only authenticated users can access protected routes.
- **Validation:** Form requests and policy classes validate input (e.g., stock availability before sale, positive quantities).

b.1 Sample Backend Logic

Pseudocode: Point-of-Sale (POS) Controller

Display POS Interface

```
FUNCTION displayPOS()
    RETRIEVE all product categories

    RETRIEVE all products
        INCLUDE category and inventory data
        FILTER products where QuantityOnHand > 0

    DISPLAY POS view with categories and products
END FUNCTION
```

Process POS Checkout

```
FUNCTION processCheckout(request)

    BEGIN database transaction

    TRY
        CREATE new Sale record using:
            customer name
            subtotal
            VAT
            discount
            total amount
            payment method
            amount received
            change
```

```
        FOR EACH item in request cart items
            CREATE SaleItem record with:
                sale ID
                product ID
                quantity
                price
                total amount

            RETRIEVE product with inventory

            IF inventory exists THEN
                DECREASE QuantityOnHand by item quantity
                SAVE updated inventory
            END IF
        END FOR

        COMMIT database transaction

        RETURN success message with sale ID

    CATCH any error
        ROLLBACK database transaction
        RETURN error message
    END TRY

END FUNCTION
```

b.2 Sample Frontend API Request

```
INITIALIZE API client

SET base URL to local server address
ENABLE sending credentials with requests
SET CSRF cookie name to "XSRF-TOKEN"
SET CSRF header name to "X-XSRF-TOKEN"

ADD request interceptor
    BEFORE sending request:
        READ CSRF token from browser cookies
        IF token exists THEN
            ATTACH token to request headers
        END IF
    RETURN updated request configuration
END interceptor

FUNCTION readCookie(cookieName)
    SEARCH browser cookies for cookieName
    IF found THEN
        RETURN cookie value
    END IF
END FUNCTION

EXPORT configured API client
```

c. Integration & Execution

System Environment

- The application was executed within a **local development environment** using a local web server, database, and PHP runtime. The Laravel backend and Vue.js frontend were integrated within the same environment to support system functionality and testing.

- The Vue.js frontend was built and served alongside the Laravel application, allowing the backend to process API requests while the frontend handled user interaction through the browser.

Frontend-Backend Connection

- The Vue.js frontend communicates with the Laravel backend through RESTful API endpoints. Axios is used to send asynchronous HTTP requests, while Laravel Sanctum manages authentication for protected operations.

- Both frontend and backend operate within the same local environment, allowing seamless API communication without cross-origin restrictions. API requests are routed through relative endpoints, and responses are returned in JSON format for reactive UI updates.

System Setup and Execution Overview

The project was developed and executed within a **local development environment** for testing and demonstration purposes. System setup involved placing the project files in the local server directory, configuring environment settings, installing required dependencies, and setting up the database.

Database migrations were performed to generate the required tables, and the system was accessed through a web browser for functional testing. All configurations and executions were handled manually throughout the