Great progress on G4 Hub! The authentication system and basic dashboard are working perfectly. Users can login and navigate through the sections. Now I need to implement the core business logic for store management and platform connectors.

CURRENT STATE
- ✅ Multi-tenant authentication working
- ✅ Database with tenants, users, stores, sync_logs tables
- ✅ Dashboard UI with navigation (Overview, Stores, Automation, etc.)
- ✅ User registration/login flow complete
- ⏳ All sections are empty placeholders - need business logic

GOAL - PHASE 2: STORE CONNECTORS & BUSINESS LOGIC
Implement the core functionality:
1. Store management (add WooCommerce/Shopify stores)
2. Platform connectors with unified interface
3. Connection testing and validation
4. Basic sync functionality
5. Populate Overview dashboard with real data

DETAILED REQUIREMENTS

1. Store Management System

Store Connection Wizard:
```typescript
// Store connection flow:
1. Platform Selection (WooCommerce vs Shopify)
2. Credentials Input (API keys/OAuth)
3. Connection Testing
4. Store Configuration
5. Initial Data Sync
```

WooCommerce Connection:

```javascript
// WooCommerce credentials format:
{
  platform: 'woocommerce',
  store_url: 'https://mystore.com',
  consumer_key: 'ck_xxxxx',
  consumer_secret: 'cs_xxxxx',
  api_version: 'wc/v3'
}

// Test endpoint: GET /wp-json/wc/v3/system_status
// Products endpoint: GET /wp-json/wc/v3/products
```

Shopify Connection:

```javascript
// Shopify credentials format:
{
  platform: 'shopify',
  store_url: 'https://mystore.myshopify.com',
  access_token: 'shpat_xxxxx',
  api_version: '2023-10'
}

// Test endpoint: GET /admin/api/2023-10/shop.json
// Products endpoint: GET /admin/api/2023-10/products.json
```

2. Backend API Endpoints Needed

Store Management:

```
GET    /api/stores                  # List tenant stores
POST   /api/stores                  # Create new store
GET    /api/stores/:id              # Get store details
PUT    /api/stores/:id              # Update store
DELETE /api/stores/:id              # Delete store
POST   /api/stores/:id/test         # Test connection
POST   /api/stores/:id/sync         # Manual sync trigger
GET    /api/stores/:id/products     # Get store products
```

Dashboard Data:

```
GET    /api/dashboard/overview        # Overview stats
GET    /api/dashboard/recent-activity # Recent sync activity
GET    /api/sync/logs?store_id=X      # Sync history
```

3. Platform Connectors Implementation

Base Connector Class:

```javascript
// backend/src/connectors/BaseConnector.js
class BaseConnector {
  constructor(storeConfig) {
    this.storeId = storeConfig.id;
```

```
    this.credentials = storeConfig.api_credentials;
    this.baseUrl = storeConfig.store_url;
    this.platform = storeConfig.platform;
  }

  // Abstract methods - must be implemented
  async testConnection() { throw new Error('Must implement testConnection()'); }
  async getProducts(page = 1, limit = 100) { throw new Error('Must implement
getProducts()'); }
  async getProduct(productId) { throw new Error('Must implement getProduct()'); }
  async updateProduct(productId, data) { throw new Error('Must implement
updateProduct()'); }
  async getStoreInfo() { throw new Error('Must implement getStoreInfo()'); }

  // Common methods
  async makeRequest(method, endpoint, data = null) {
    // Implement with axios, handle rate limiting, errors
  }
}
```

WooCommerce Connector:

```javascript
// backend/src/connectors/WooCommerceConnector.js
class WooCommerceConnector extends BaseConnector {
  async testConnection() {
    try {
      const response = await this.makeRequest('GET', '/wp-json/wc/v3/system_status');
      return {
        success: true,
        store_name: response.data.settings.title.value,
        wc_version: response.data.version,
        products_count: await this.getProductsCount()
      };
    } catch (error) {
      return { success: false, error: error.message };
    }
  }

  async getProducts(page = 1, limit = 10) {
    const response = await this.makeRequest('GET', '/wp-json/wc/v3/products', {
      params: { page, per_page: limit, status: 'publish' }
    });

    return {
      products: response.data.map(this.transformProduct),
      pagination: {
```

```
      current_page: page,
      total_pages: parseInt(response.headers['x-wp-totalpages']),
      total_items: parseInt(response.headers['x-wp-total'])
    }
  };
}

transformProduct(wooProduct) {
  return {
    id: wooProduct.id,
    name: wooProduct.name,
    sku: wooProduct.sku,
    price: parseFloat(wooProduct.price),
    stock_quantity: wooProduct.stock_quantity,
    manage_stock: wooProduct.manage_stock,
    stock_status: wooProduct.stock_status,
    images: wooProduct.images.map(img => img.src),
    platform: 'woocommerce'
  };
}
}
```

Shopify Connector:

```javascript
// backend/src/connectors/ShopifyConnector.js
class ShopifyConnector extends BaseConnector {
  async testConnection() {
    try {
      const response = await this.makeRequest('GET', '/admin/api/2023-10/shop.json');
      return {
        success: true,
        store_name: response.data.shop.name,
        domain: response.data.shop.domain,
        products_count: await this.getProductsCount()
      };
    } catch (error) {
      return { success: false, error: error.message };
    }
  }

  async getProducts(page = 1, limit = 10) {
    const response = await this.makeRequest('GET', '/admin/api/2023-10/products.json', {
      params: { limit, page }
    });

    return {
```

```
      products: response.data.products.map(this.transformProduct),
      pagination: {
        current_page: page,
        has_next: response.data.products.length === limit
      }
    };
  }

  transformProduct(shopifyProduct) {
    const variant = shopifyProduct.variants[0]; // Main variant
    return {
      id: shopifyProduct.id,
      name: shopifyProduct.title,
      sku: variant.sku,
      price: parseFloat(variant.price),
      stock_quantity: variant.inventory_quantity,
      manage_stock: variant.inventory_management !== null,
      images: shopifyProduct.images.map(img => img.src),
      platform: 'shopify'
    };
  }
}
```

4. Frontend Components Needed

Store Management Pages:

```typescript
// app/dashboard/stores/page.tsx - Store list with Add Store button
// app/dashboard/stores/add/page.tsx - Add store wizard
// app/dashboard/stores/[id]/page.tsx - Store details/management

// Key components:
- StoreCard: Display store info, status, actions
- AddStoreWizard: Multi-step form (platform → credentials → test → save)
- ConnectionTest: Test button with loading/success/error states
- ProductsList: Display store products in table
- SyncStatus: Show last sync, sync button
```

Updated Overview Dashboard:

```typescript
// Show real data instead of placeholders:
- Connected Stores count (from database)
- Products Synced count (from sync_logs)
- Last Sync timestamp
```

- Recent activity list
- Store status indicators
```

## 5. Database Updates Needed

### Additional tables:

```sql
-- Store products cache
CREATE TABLE store_products (
    id SERIAL PRIMARY KEY,
    tenant_id INTEGER REFERENCES tenants(id),
    store_id INTEGER REFERENCES stores(id),
    platform_product_id VARCHAR(100) NOT NULL,
    sku VARCHAR(100),
    name VARCHAR(255),
    price DECIMAL(10,2),
    stock_quantity INTEGER,
    manage_stock BOOLEAN DEFAULT false,
    data JSONB, -- Full product data
    last_updated TIMESTAMP DEFAULT NOW(),
    UNIQUE(store_id, platform_product_id)
);

-- Store connection status
ALTER TABLE stores ADD COLUMN connection_status VARCHAR(20) DEFAULT
'untested';
ALTER TABLE stores ADD COLUMN last_connection_test TIMESTAMP;
ALTER TABLE stores ADD COLUMN store_info JSONB DEFAULT '{}';
ALTER TABLE stores ADD COLUMN products_count INTEGER DEFAULT 0;
```

6. Expected Functionality

Stores Section:

- ✅ List all connected stores with status indicators
- ✅ “Add Store” button opens connection wizard
- ✅ Platform selection (WooCommerce/Shopify radio buttons)
- ✅ Credentials form with validation
- ✅ “Test Connection” button with real API testing
- ✅ Success: Save store and redirect to store details
- ✅ Store details page shows products, sync options

Overview Dashboard:

- ✅ Real connected stores count
```

- ✅ Real products synced count
- ✅ Last sync timestamps
- ✅ Recent activity from sync_logs
- ✅ Quick actions (Add Store, Manual Sync)

Connection Testing:

- ✅ Validate API credentials before saving
- ✅ Show meaningful error messages
- ✅ Display store info on successful connection
- ✅ Handle rate limiting and timeouts gracefully

7. Error Handling Requirements

```javascript
// Comprehensive error handling for:
- Invalid API credentials
- Network timeouts
- Rate limiting (429 errors)
- Invalid store URLs
- Missing required permissions
- API version mismatches

// User-friendly error messages:
"Unable to connect to your WooCommerce store. Please check your Consumer Key and Secret."
"Your Shopify access token doesn't have required permissions. Please regenerate with 'read_products' scope."
```

TECHNICAL CONSTRAINTS

- Use axios for HTTP requests with timeout (30s)
- Implement exponential backoff for rate limits
- Encrypt all API credentials in database
- Add proper TypeScript types for all data
- Use React Query for data fetching on frontend
- Add loading states for all async operations
- Implement proper error boundaries

EXPECTED OUTPUT

Please implement these files in this priority order:

1. Backend Connectors:

- `backend/src/connectors/BaseConnector.js`
- `backend/src/connectors/WooCommerceConnector.js`

- `backend/src/connectors/ShopifyConnector.js`
- `backend/src/connectors/ConnectorFactory.js`

2. Backend Controllers:

- Update `backend/src/controllers/storeController.js` with all CRUD operations
- `backend/src/controllers/dashboardController.js` for overview data
- Add routes in `backend/src/routes/stores.js`

3. Database Updates:

- Migration script for new tables and columns
- Update Sequelize models

4. Frontend Pages:

- Update `app/dashboard/page.tsx` with real overview data
- Create `app/dashboard/stores/page.tsx` - stores list
- Create `app/dashboard/stores/add/page.tsx` - add store wizard
- Create `app/dashboard/stores/[id]/page.tsx` - store details

5. Frontend Components:

- `components/stores/StoreCard.tsx`
- `components/stores/AddStoreWizard.tsx`
- `components/stores/ConnectionTest.tsx`
- `components/dashboard/OverviewStats.tsx`

SUCCESS CRITERIA

- ✅ User can add WooCommerce store with Consumer Key/Secret
- ✅ User can add Shopify store with access token
- ✅ Connection testing works with real API calls
- ✅ Store list shows all connected stores with status
- ✅ Store details page displays real products from API
- ✅ Overview dashboard shows real data from database
- ✅ Error handling provides helpful messages
- ✅ All API credentials are encrypted in database

PRIORITY FEATURES TO IMPLEMENT FIRST:

1. Store CRUD operations (backend)
1. WooCommerce connector with connection testing
1. Add Store wizard (frontend)
1. Store list page with real data
1. Overview dashboard with real stats

Focus on getting the store connection flow working end-to-end first - user should be able to successfully connect a WooCommerce store and see it listed in the dashboard.