
📖 Project Overview

This project is a fully functional furniture e-commerce platform supporting product browsing, cart management, and order checkout. The backend uses the lightweight Spark Java framework, while the frontend is built with vanilla JavaScript + jQuery + Bootstrap for a responsive interface.

Key Features

Module	Description
🛍️ Products	Product display, category filtering, keyword search, price sorting
👤 Users	Registration, login, profile management, multi-address support
🛒 Cart	Add/remove items, quantity updates, selective checkout
📦 Orders	Order creation, stock validation, order history

🛠️ Tech Stack

Backend

- **Java 21** - Programming language
- **Spark Java 2.9.4** - Lightweight web framework
- **Gson 2.10.1** - JSON serialization/deserialization
- **SLF4J** - Logging framework

Frontend

- **HTML5 / CSS3** - Page structure and styling
- **JavaScript (ES6+)** - Interaction logic
- **jQuery 3.7.1** - DOM manipulation and AJAX
- **Bootstrap 5** - UI framework

Data Storage

- **JSON Files** - Lightweight data persistence (products.json, users.json, carts.json, orders.json)

🏗 Architecture

MVC Structure

This project adopts a modified MVC (Model-View-Controller) architecture with some adaptations:

Layer	Responsibility	Directory
View	Frontend UI and user interaction	`src/main/resources/public/`
Controller	HTTP request handling, route registration	`src/main/java/.../controller/`
Model	Data entity definitions	`src/main/java/.../model/`
Data	Data persistence and business logic	`src/main/java/.../data/`

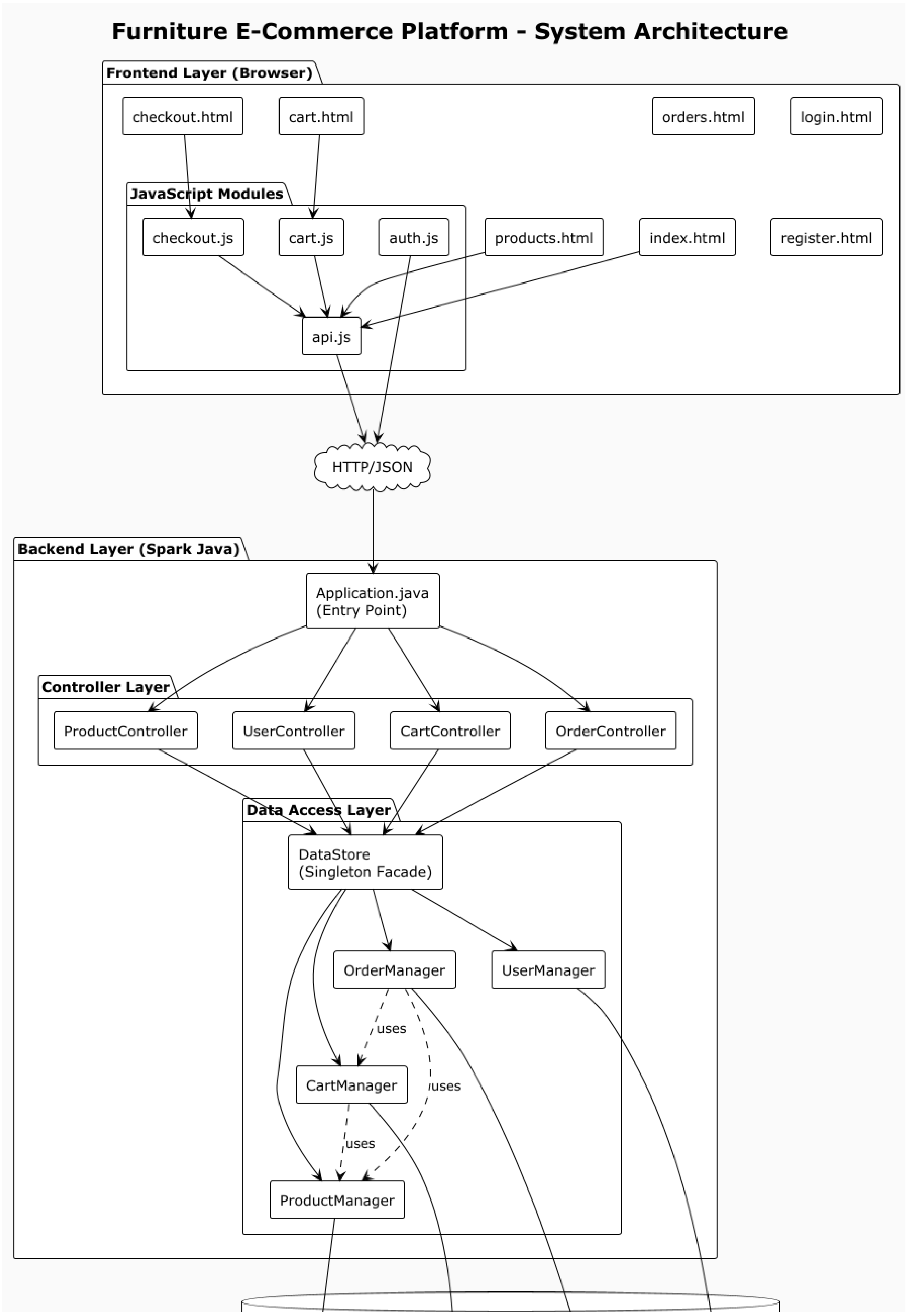
> 💡 **Note**: This project merges the traditional Service and DAO layers into a unified Data layer (Manager classes), simplifying architecture complexity for small-scale projects.

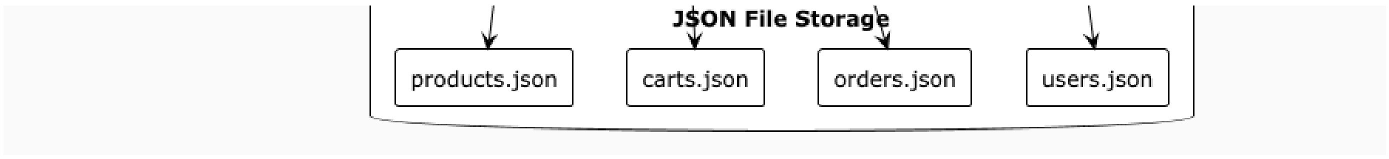
📁 Project Structure

```
ecommerce/
├── pom.xml                          # Maven configuration
├── README.md                        # Project documentation
├── docs/
│   └── diagrams/                    # PlantUML diagram files
│       ├── architecture.puml
│       ├── class-diagram.puml
│       ├── user-registration-sequence.puml
│       ├── user-address-management.puml
│       ├── product-search-filter.puml
│       ├── product-stock-check.puml
│       ├── cart-add-display.puml
│       ├── cart-to-checkout.puml
│       └── order-creation.puml
│
├── src/main/
│   ├── java/com/furniture/
│   │   ├── Application.java         # Entry point
│   │   │
│   │   ├── controller/              # Controller layer
│   │   │   ├── ProductController.java
│   │   │   ├── UserController.java
│   │   │   ├── CartController.java
│   │   │   └── OrderController.java
│   │   │
│   │   ├── model/                   # Model layer
│   │   │   ├── Product.java
│   │   │   ├── User.java
│   │   │   ├── Address.java
│   │   │   ├── CartItem.java
│   │   │   ├── Order.java
│   │   │   └── ApiResponse.java
│   │   │
│   │   └── data/                    # Data access layer
│   │       ├── DataStore.java       # Singleton facade
│   │       ├── ProductManager.java
│   │       ├── UserManager.java
│   │       ├── CartManager.java
│   │       └── OrderManager.java
│   │
│   └── resources/
│       ├── data/                   # JSON data files
│       │   ├── products.json
│       │   ├── users.json
│       │   ├── carts.json
│       │   └── orders.json
│       │
│       └── public/                 # Frontend static resources
│           ├── *.html              # HTML pages
│           ├── js/                 # JavaScript modules
│           ├── css/                # Stylesheets
│           ├── bootstrap/          # Bootstrap framework
│           └── JQuery/             # jQuery library
│
└── target/                         # Build output
```

Architecture Diagram

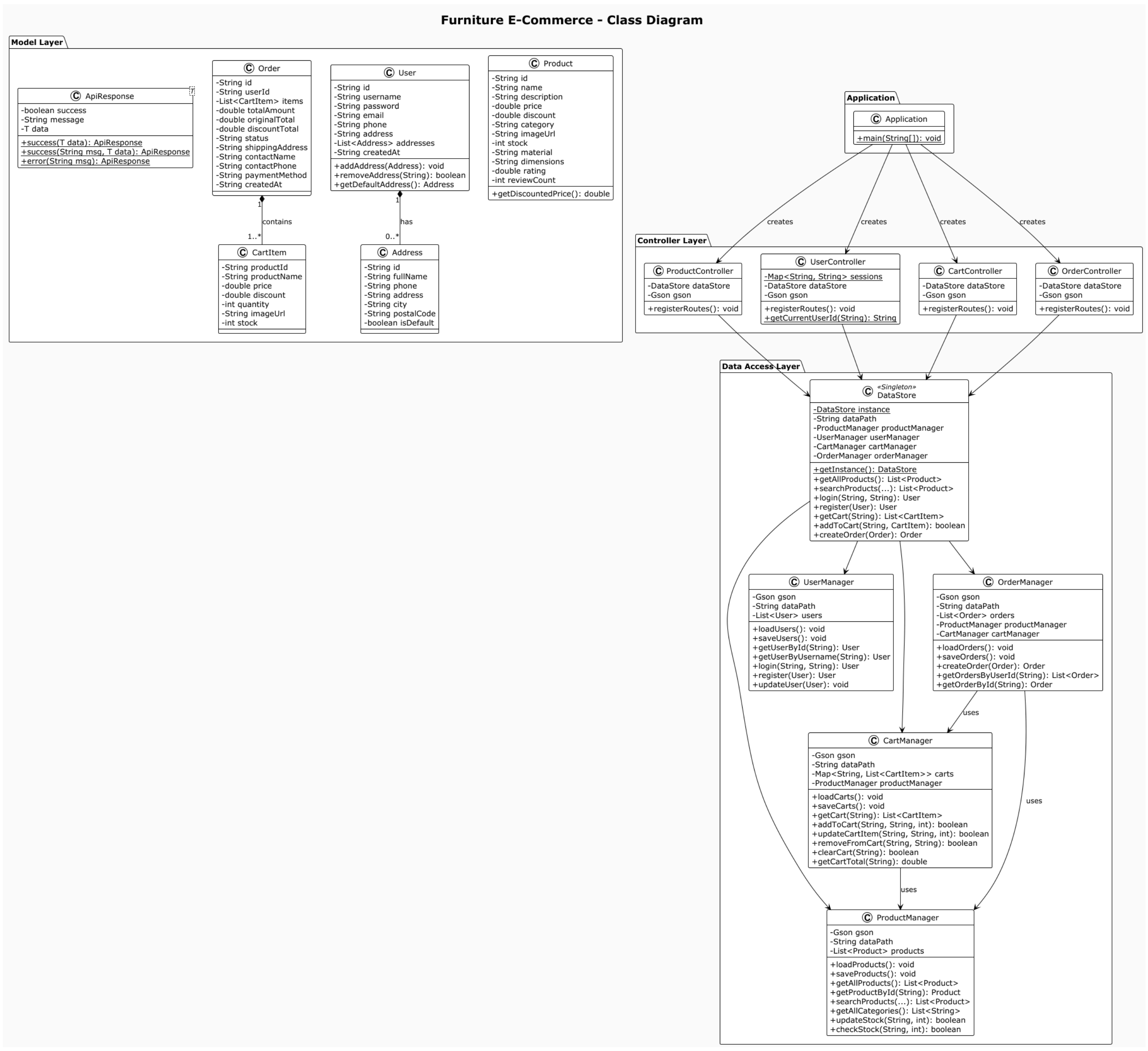
SVG Image Link: Furniture E-Commerce Platform - System Architecture


Furniture E-Commerce Platform - System Architecture
Frontend Layer (Browser)
checkout.html
cart.html
orders.html
login.html
JavaScript Modules
checkout.js
cart.js
auth.js
api.js
products.html
index.html
register.html
HTTP/JSON
Backend Layer (Spark Java)
Application.java
(Entry Point)
Controller Layer
ProductController
UserController
CartController
OrderController
Data Access Layer
DataStore
(Singleton Facade)
OrderManager
UserManager
uses
CartManager
uses
uses
ProductManager



Class Diagram

SVG Image Link:[Furniture E-Commerce - Class Diagram]



Data Flow & Interaction

Request-Response Flow

```
┌────────────────────────────────────────────────────────────────────────────┐
│                          Data Flow Diagram                                 │
├────────────────────────────────────────────────────────────────────────────┤
│                                                                            │
│  User Action       Frontend            Backend             Storage         │
│      │                 │                   │                   │           │
│      │ ① Click         │                   │                   │           │
│      │ ──────────────► │                   │                   │           │
│      │                 │ ② Build Request   │                   │           │
│      │                 │ (api.js)          │                   │           │
│      │                 │ ─────────────────►│                   │           │
│      │                 │                   │ ③ Route Dispatch  │           │
│      │                 │  HTTP Request     │ (Controller)      │           │
│      │                 │  (JSON Body)      │ ─────────────────►│           │
│      │                 │                   │                   │           │
│      │                 │                   │ ④ Data Operation  │           │
│      │                 │                   │ (DataStore/       │           │
|      |                 |                   |   Manager)        |            |
|      |                 |                   |◄─────────────────|            |
|      |                 |  ⑤ Return Result  |                   |            |
|      |                 |◄─────────────────  |                   |            |
|      |                 |  HTTP Response    |                   |            |
|      |                 |  (JSON)           |                   |            |
|      |  ⑥ Update UI    |                   |                   |            |
|      |◄───────────────  |                   |                   |            |
└─────────────────────────────────────────────────────────────────────────────┘
```

Communication Protocol

Feature	Description
Protocol	HTTP/HTTPS
Data Format	JSON
Authentication	Bearer Token (stored in localStorage)
Request Methods	RESTful API (GET/POST/PUT/DELETE)
CORS	Configured via middleware

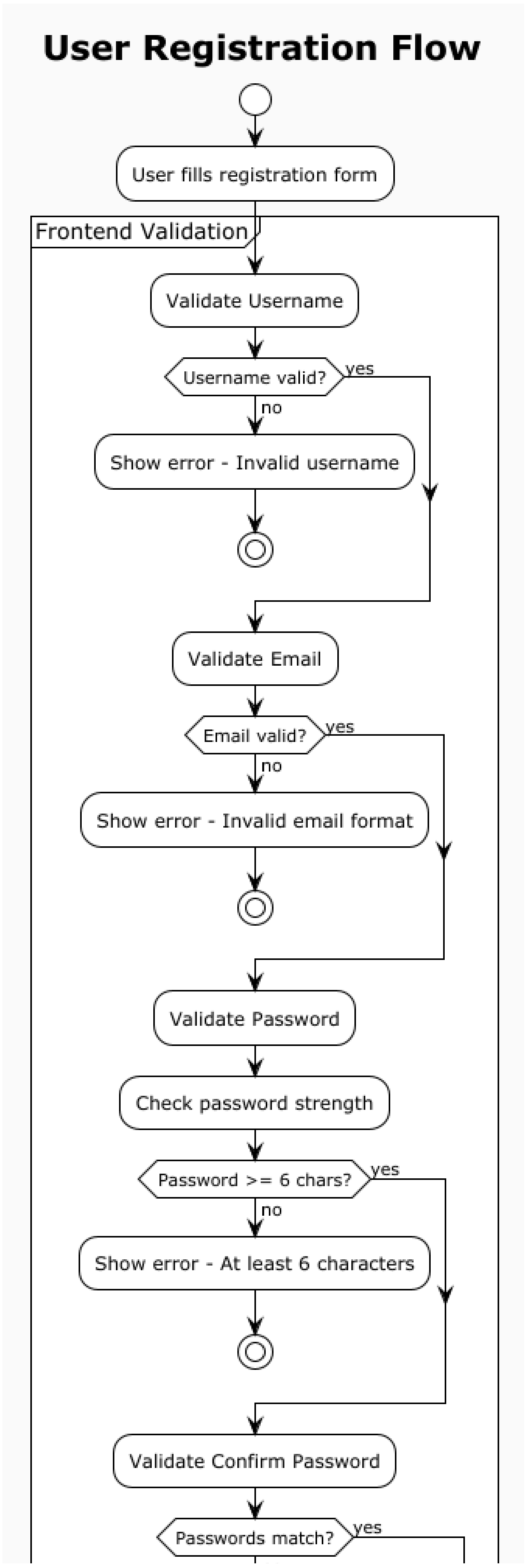
Unified Response Format

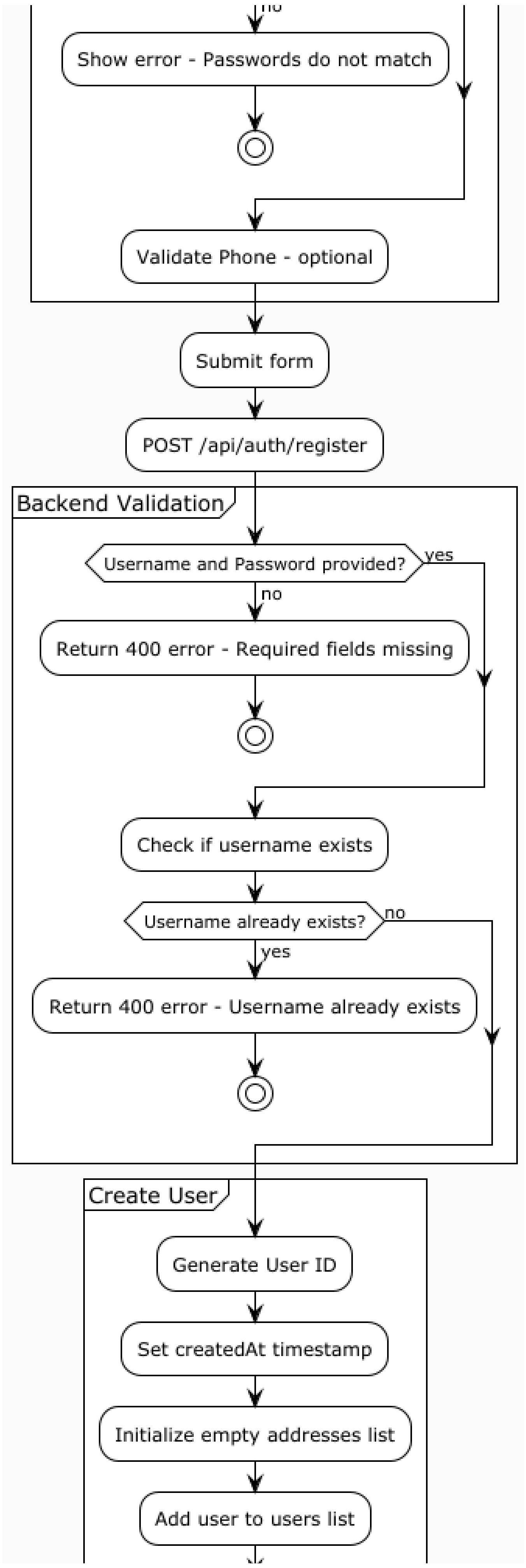
```json
{
    "success": true,
    "message": "Operation successful",
    "data": { ... }
}
```

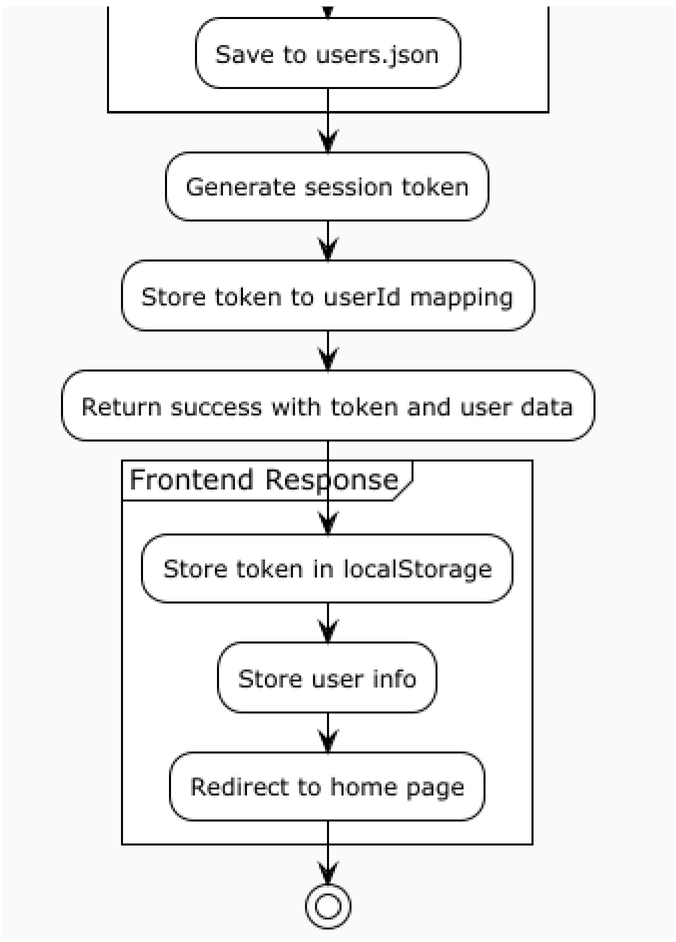
🔧 Core Module Implementation

User Module

1. Registration with Field Validation


User Registration Flow
User fills registration form
Frontend Validation
Validate Username
Username valid?
yes
no
Show error - Invalid username
Validate Email
Email valid?
yes
no
Show error - Invalid email format
Validate Password
Check password strength
Password >= 6 chars?
yes
no
Show error - At least 6 characters
Validate Confirm Password
Passwords match?
yes


no
Show error - Passwords do not match
Validate Phone - optional
Submit form
POST /api/auth/register
Backend Validation
Username and Password provided?
yes
no
Return 400 error - Required fields missing
Check if username exists
Username already exists?
no
yes
Return 400 error - Username already exists
Create User
Generate User ID
Set createdAt timestamp
Initialize empty addresses list
Add user to users list


Save to users.json
Generate session token
Store token to userId mapping
Return success with token and user data
Frontend Response
Store token in localStorage
Store user info
Redirect to home page

Frontend Validation (Client-Side)

The registration form implements real-time validation using regex patterns and state tracking:

```
// Validation patterns defined in register.js
const patterns = {
    username: /^[a-zA-Z0-9_]+$/,          // Alphanumeric + underscore only
    email: /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/,
    phone: /^[0-9+\-\s()]*$/              // Numbers and phone symbols
};

// Validation state tracking
const validationState = {
    username: false,
    email: false,
    password: false,
    confirmPassword: false,
    phone: true  // Optional field, default valid
};
```

Validation Flow:

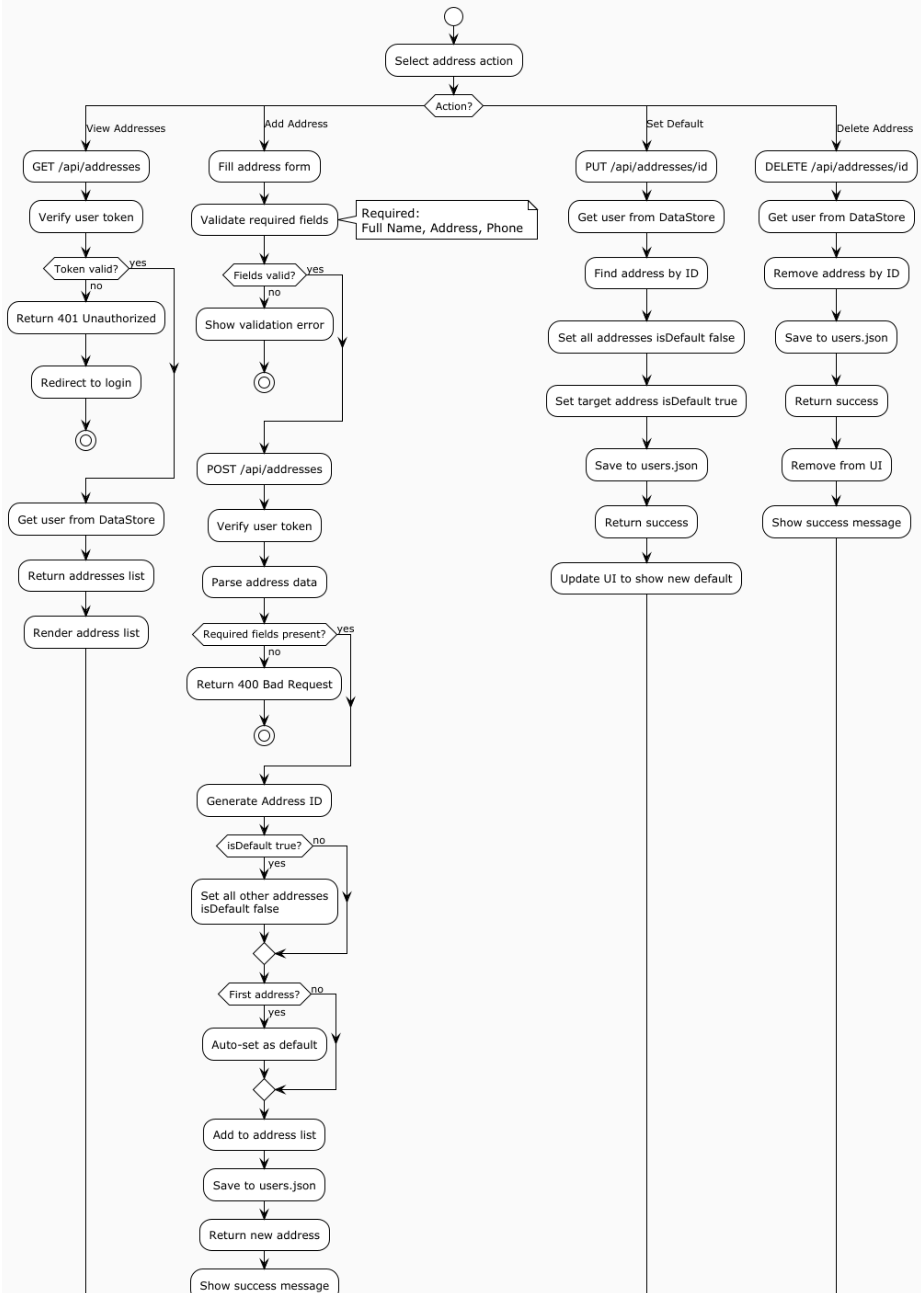
1. **On Blur Events**: Each field validates when focus leaves
2. **Password Strength**: Real-time strength indicator (Weak/Medium/Strong)
3. **Confirm Password**: Validates match with password field
4. **Submit Prevention**: Form only submits when all `validationState` flags are `true`

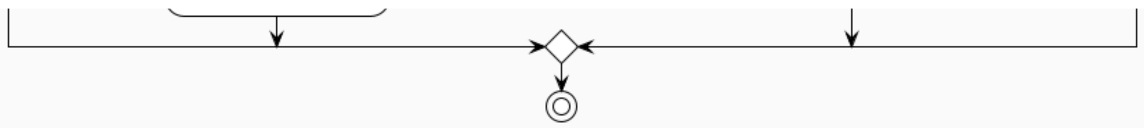
Backend Validation (Server-Side)

```java
// UserController.java - POST /api/auth/register
post("/api/auth/register", (req, res) -> {
    User newUser = gson.fromJson(req.body(), User.class);

    // Null check for required fields
    if (newUser.getUsername() == null || newUser.getPassword() == null) {
        res.status(400);
        return gson.toJson(ApiResponse.error("Username and password required"));
    }

    // Delegate to UserManager
    User user = dataStore.register(newUser);
    // ...
});

// UserManager.java - register()
public User register(User newUser) {
    // Check for duplicate username
    if (getUserByUsername(newUser.getUsername()) != null) {
        return null;  // Username exists
    }

    // Generate sequential ID: U001, U002, ...
    String newId = "U" + "%03d".formatted(users.size() + 1);
    newUser.setId(newId);
    newUser.setCreatedAt(LocalDateTime.now().toString());

    // Initialize empty address list
    if (newUser.getAddresses() == null) {
        newUser.setAddresses(new ArrayList<>());
    }

    users.add(newUser);
    saveUsers();  // Persist to JSON
    return newUser;
}
```

2. Address Management

User Address Management Flow





The User model supports multiple addresses with default address handling:

```java
// User.java - Address management methods
public void addAddress(Address addr) {
    if (addresses == null) {
        addresses = new ArrayList<>();
    }

    // Generate unique ID using timestamp
    if (addr.getId() == null || addr.getId().isEmpty()) {
        addr.setId("ADDR" + System.currentTimeMillis());
    }

    // If setting as default, clear other defaults
    if (addr.isDefault()) {
        for (Address a : addresses) {
            a.setDefault(false);
        }
    }

    // First address is auto-set as default
    if (addresses.isEmpty()) {
        addr.setDefault(true);
    }

    addresses.add(addr);
}

public Address getDefaultAddress() {
    if (addresses == null || addresses.isEmpty()) {
        return null;
    }
    // Return marked default, or first address as fallback
    return addresses.stream()
        .filter(Address::isDefault)
        .findFirst()
```

```
            .orElse(addresses.getFirst());
}
```

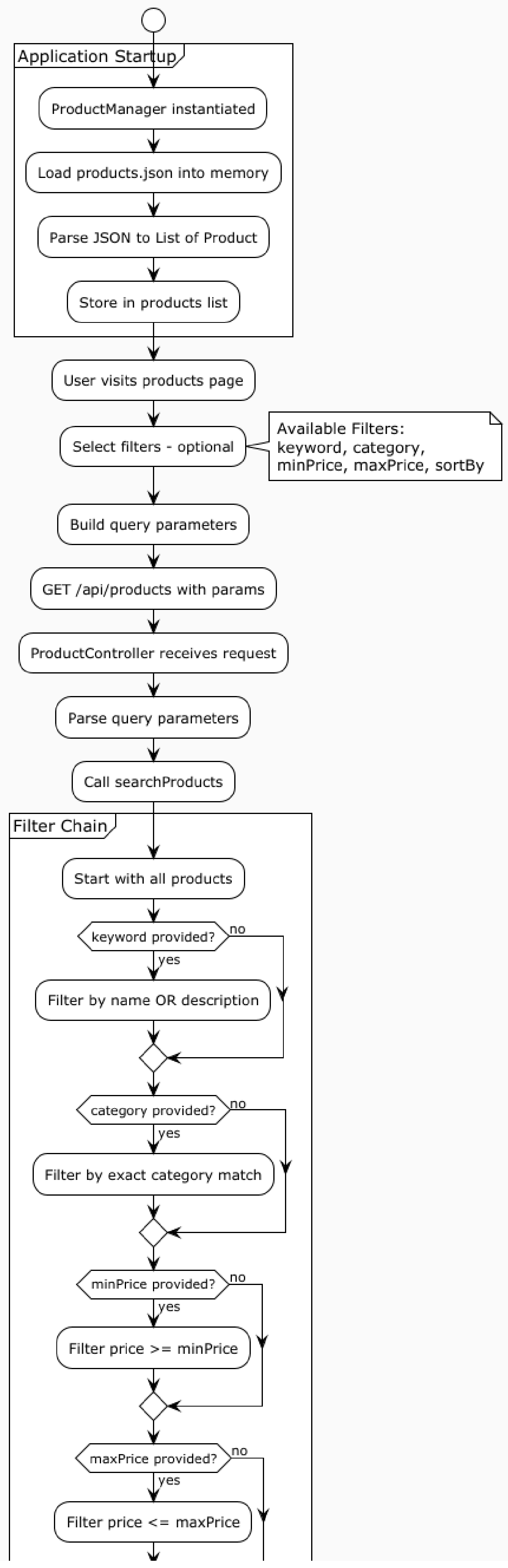
API Endpoints for Address Management:

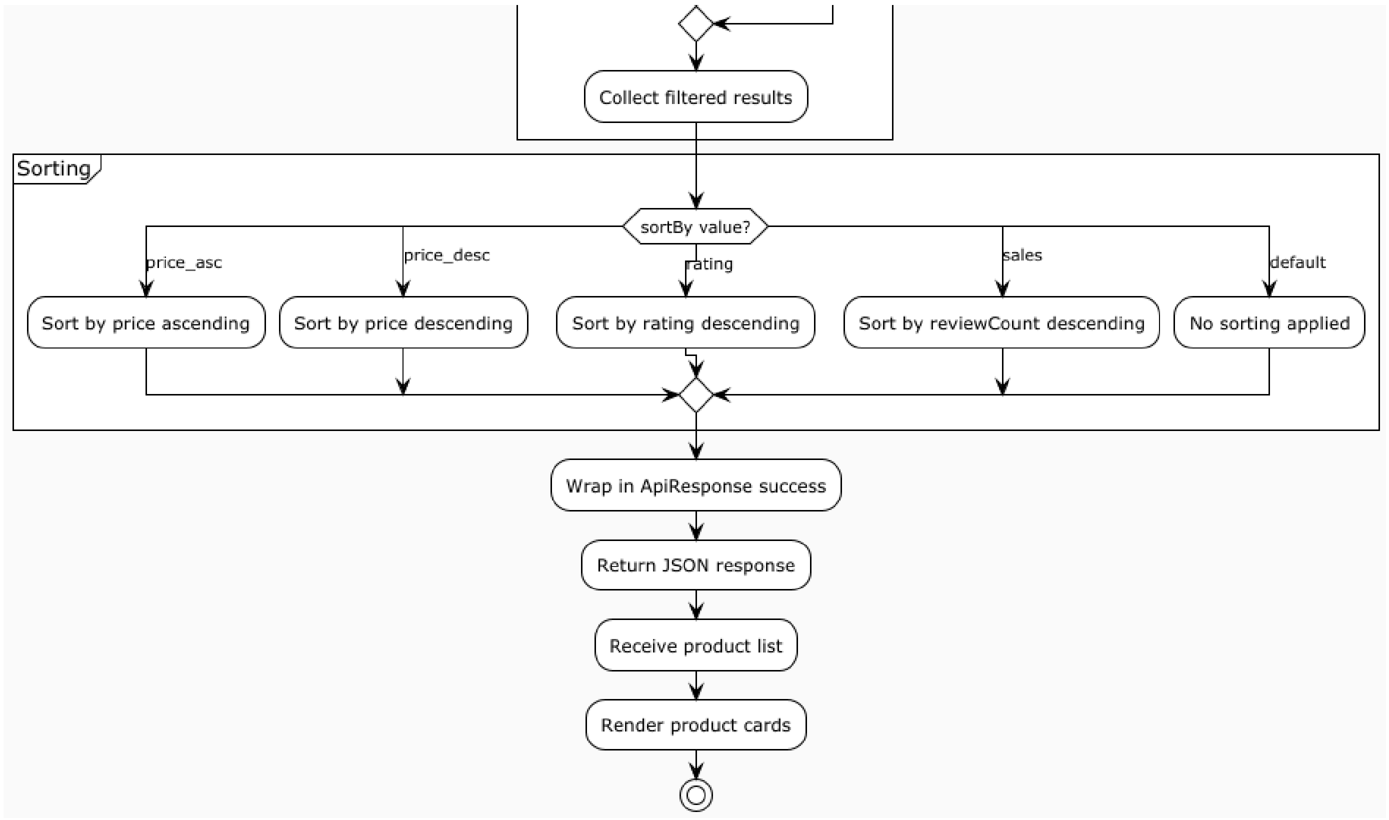
Method	Endpoint	Description
GET	`/api/addresses`	Get all user addresses
POST	`/api/addresses`	Add new address
PUT	`/api/addresses/:id`	Update address / Set default
DELETE	`/api/addresses/:id`	Delete address

Product Module

1. Data Loading and Search/Filter

Product Search & Filter Flow





Data Loading (Application Startup)

Products are loaded into memory when `ProductManager` is instantiated:

```java
// ProductManager.java
public ProductManager(String dataPath) {
    this.gson = new GsonBuilder().setPrettyPrinting().create();
    this.dataPath = dataPath;
    loadProducts();  // Load on construction
}

private void loadProducts() {
    try {
        String json = new String(Files.readAllBytes(
            Path.of(dataPath + "products.json")), StandardCharsets.UTF_8);
        Type type = new TypeToken<Map<String, List<Product>>>(){}.getType();
        Map<String, List<Product>> data = gson.fromJson(json, type);
        this.products = data.get("products");
    } catch (Exception e) {
        this.products = new ArrayList<>();
    }
}
```

Search and Filter Implementation

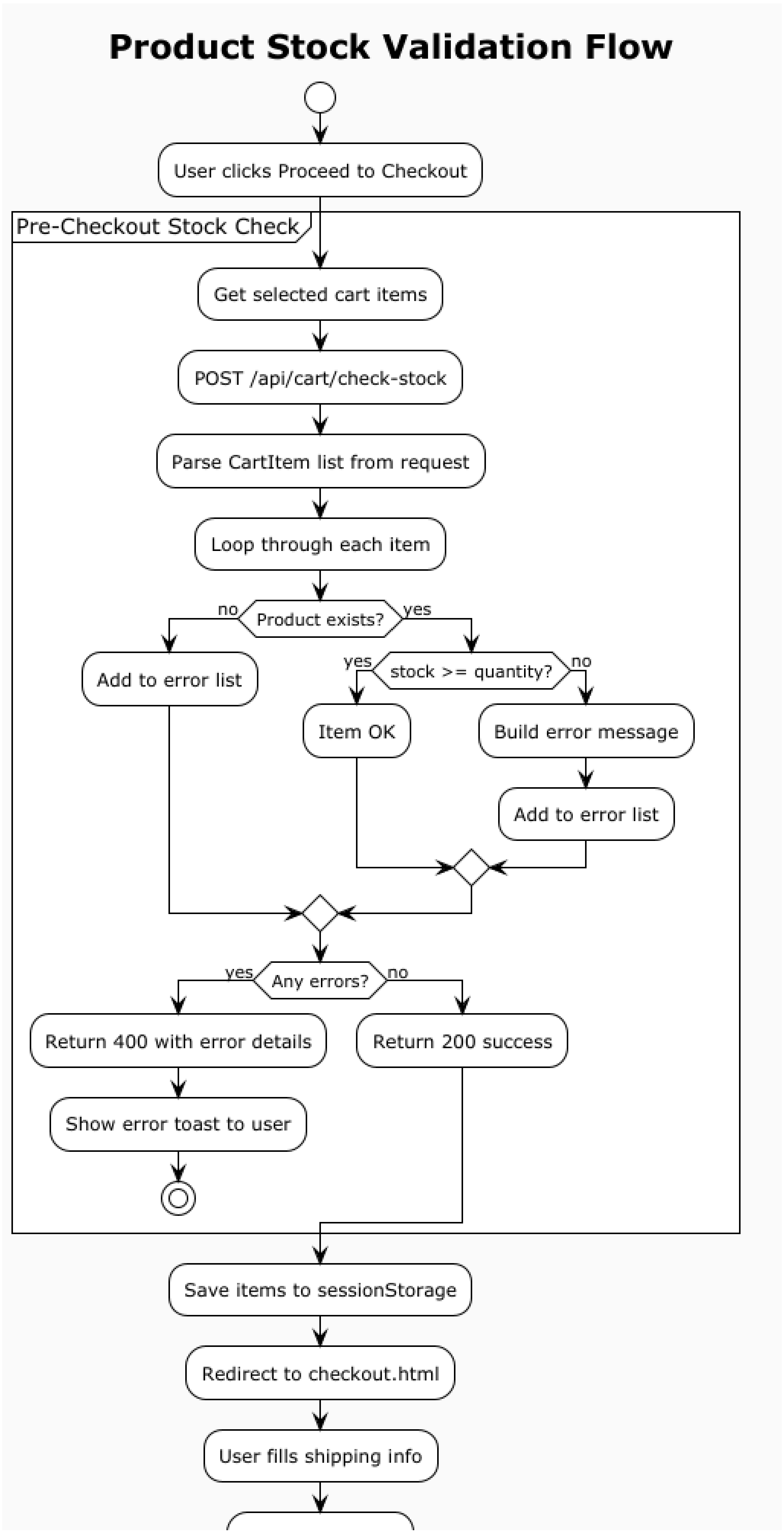
The search uses Java Streams for flexible filtering and sorting:

```java
// ProductManager.java - searchProducts()
public List<Product> searchProducts(String keyword, String category,
                                     Double minPrice, Double maxPrice, String sortBy) {
    List<Product> result = products.stream()
        .filter(p -> {
            boolean match = true;

            // Keyword filter (name OR description)
            if (keyword != null && !keyword.isEmpty()) {
                match = p.getName().toLowerCase().contains(keyword.toLowerCase()) ||
                        p.getDescription().toLowerCase().contains(keyword.toLowerCase());
            }

            // Category filter (exact match)
            if (category != null && !category.isEmpty()) {
                match = match && p.getCategory().equals(category);
            }

            // Price range filters
            if (minPrice != null) {
                match = match && p.getPrice() >= minPrice;
            }
            if (maxPrice != null) {
                match = match && p.getPrice() <= maxPrice;
            }

            return match;
        })
        .collect(Collectors.toList());

    // Apply sorting
    if (sortBy != null) {
        switch (sortBy) {
            case "price_asc" -> result.sort(Comparator.comparingDouble(Product::getPrice));
            case "price_desc" -> result.sort(Comparator.comparingDouble(Product::getPrice).reversed());
            case "rating" -> result.sort(Comparator.comparingDouble(Product::getRating).reversed());
            case "sales" -> result.sort(Comparator.comparingInt(Product::getReviewCount).reversed());
        }
    }

    return result;
}
```

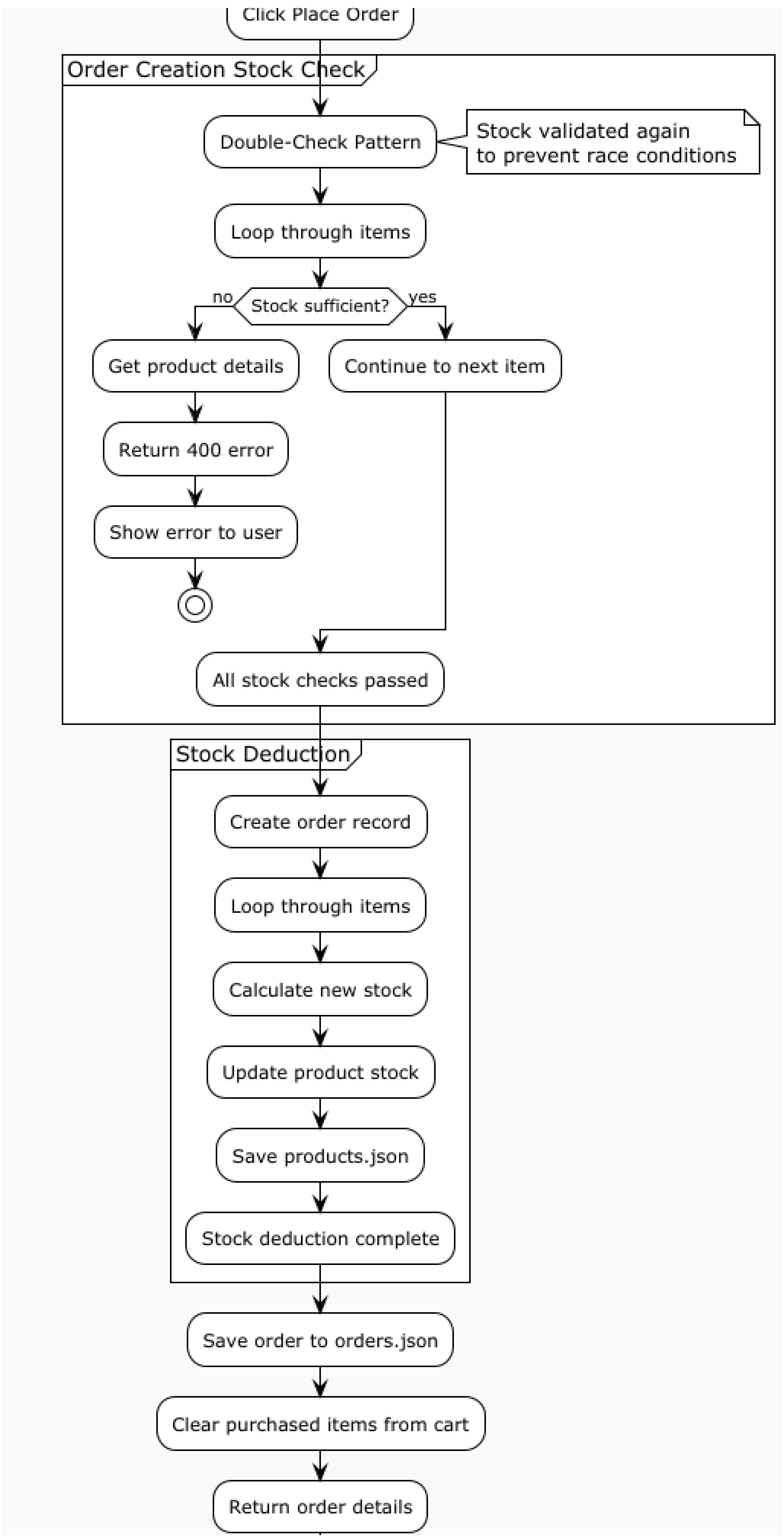
Frontend Filter Request:

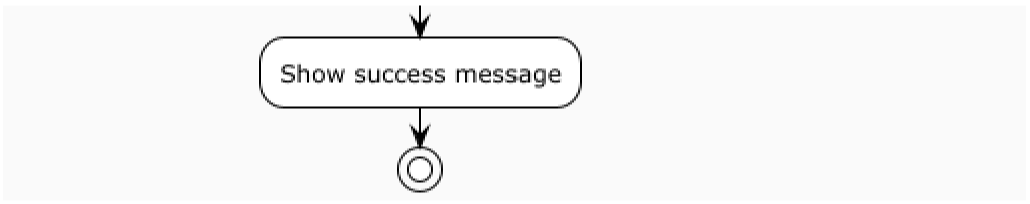
```javascript
// products.js - Building filter query
async function loadProducts() {
    const params = {
        category: $('#categoryFilter').val(),
        keyword: $('#searchInput').val(),
        minPrice: $('#minPrice').val(),
        maxPrice: $('#maxPrice').val(),
        sortBy: $('#sortBy').val()
    };

    const result = await API.getProducts(params);
    // Render products...
}
```

2. Stock Validation



Product Stock Validation Flow
User clicks Proceed to Checkout
Pre-Checkout Stock Check
Get selected cart items
POST /api/cart/check-stock
Parse CartItem list from request
Loop through each item
no
Product exists?
yes
Add to error list
yes
stock >= quantity?
no
Item OK
Build error message
Add to error list
yes
Any errors?
no
Return 400 with error details
Return 200 success
Show error toast to user
Save items to sessionStorage
Redirect to checkout.html
User fills shipping info


Click Place Order
Order Creation Stock Check
Double-Check Pattern
Stock validated again
to prevent race conditions
Loop through items
Stock sufficient?
no
yes
Get product details
Continue to next item
Return 400 error
Show error to user
All stock checks passed
Stock Deduction
Create order record
Loop through items
Calculate new stock
Update product stock
Save products.json
Stock deduction complete
Save order to orders.json
Clear purchased items from cart
Return order details



Stock validation occurs at two critical points:

Point 1: Before Checkout (User Experience)

```javascript
// cart.js - checkoutBtn click handler
$('#checkoutBtn').on('click', async function() {
    const itemsToCheck = cartData.items.filter(item =>
selectedItems.has(item.productId));

    // Pre-checkout stock validation
    const stockResult = await API.checkStock(itemsToCheck);
    if (!stockResult.success) {
        Utils.showToast(stockResult.message, 'error');
        return;  // Block checkout
    }

    // Proceed to checkout
    sessionStorage.setItem('checkoutItems', JSON.stringify(itemsToCheck));
    window.location.href = '/checkout.html';
});
```

Point 2: During Order Creation (Data Integrity)

```java
// OrderController.java - Stock check in order creation
for (CartItem item : selectedItems) {
    if (!dataStore.checkStock(item.getProductId(), item.getQuantity())) {
        Product p = dataStore.getProductById(item.getProductId());
        String stockInfo = p != null ? "(Stock: " + p.getStock() + ")" : "";
        res.status(400);
        return gson.toJson(ApiResponse.error(
            "Product " + item.getProductName() + " insufficient stock" +
stockInfo));
    }
}
```

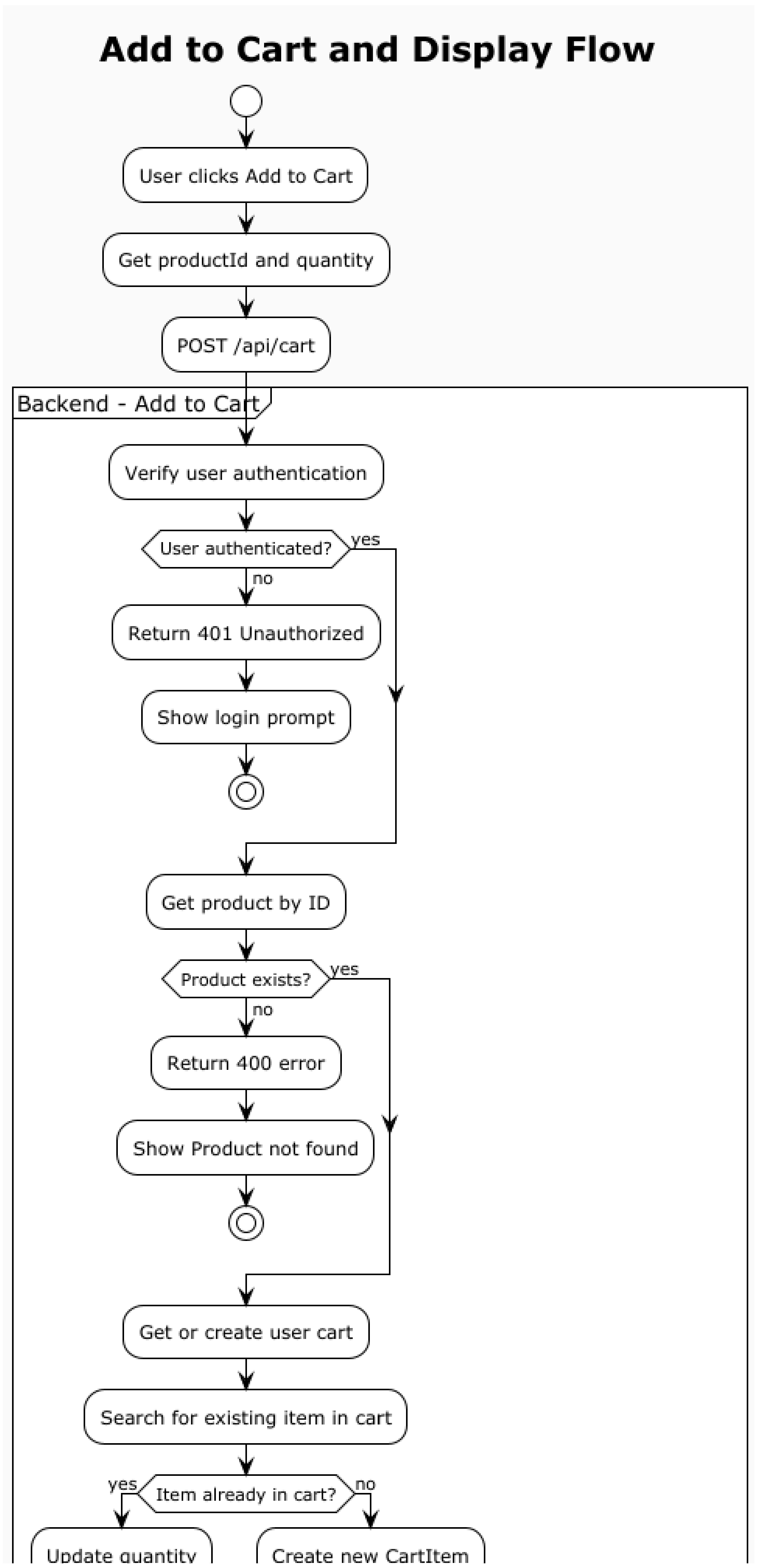
```java

// ProductManager.java - checkStock()
public boolean checkStock(String productId, int quantity) {
    Product product = getProductById(productId);
    return product != null && product.getStock() >= quantity;
}
```

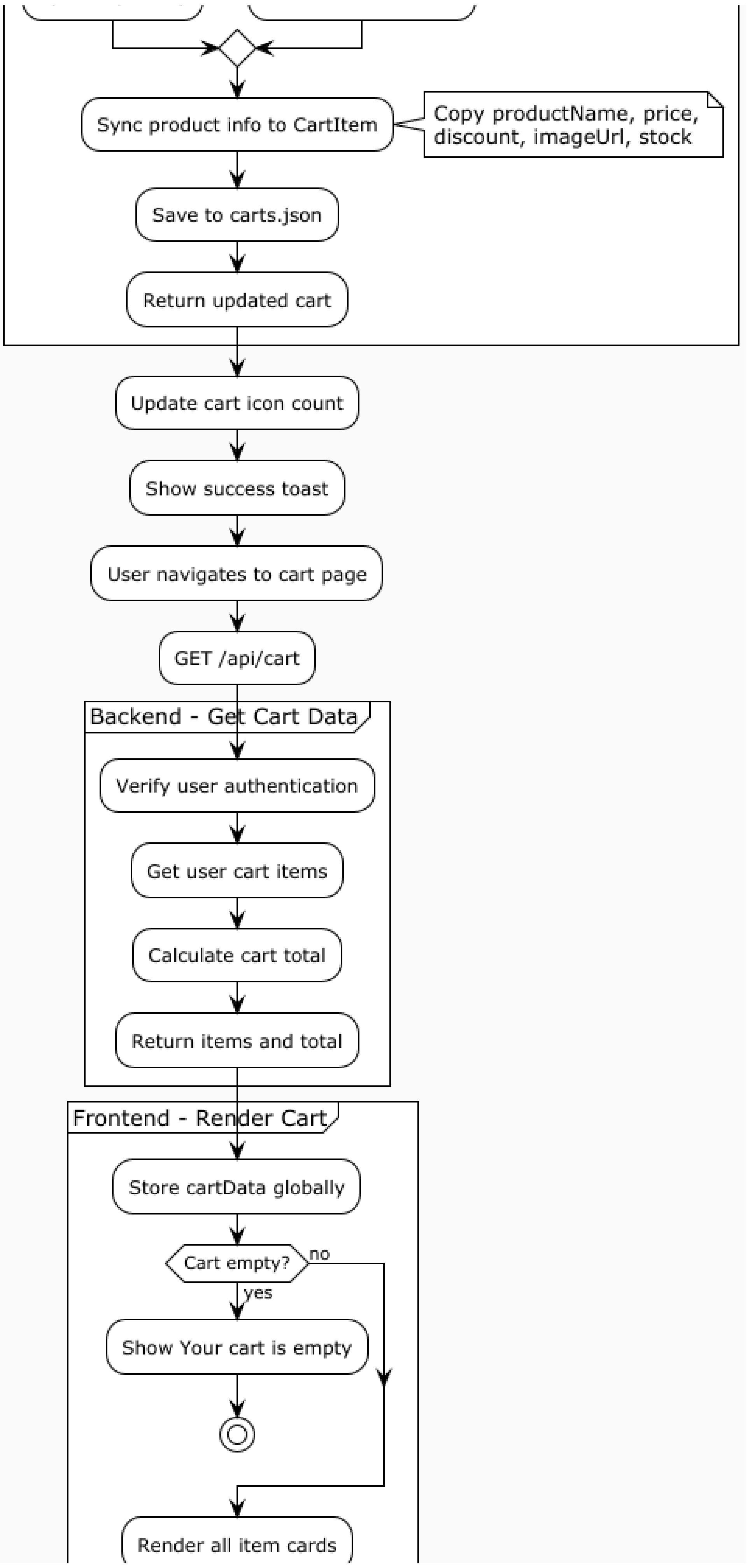
Stock Deduction After Order:

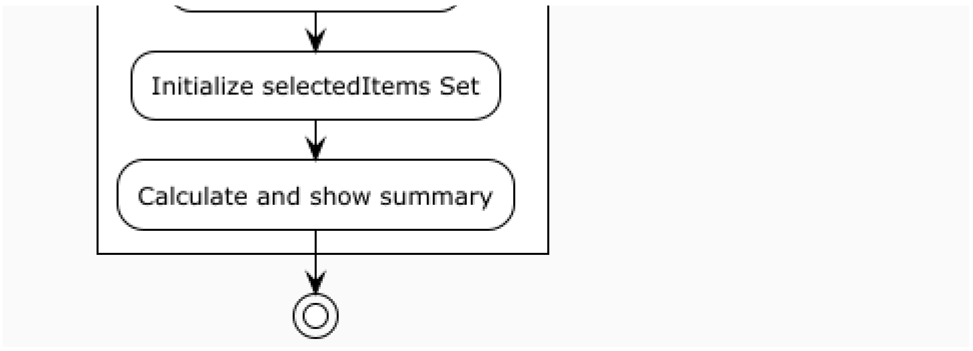
```java
// OrderManager.java - createOrder()
for (CartItem item : order.getItems()) {
    productManager.updateStock(item.getProductId(), item.getQuantity());
}

// ProductManager.java - updateStock()
public boolean updateStock(String productId, int quantityToReduce) {
    Product product = getProductById(productId);
    if (product == null) return false;

    int newStock = product.getStock() - quantityToReduce;
    if (newStock < 0) return false;

    product.setStock(newStock);
    saveProducts();  // Persist change
    return true;
}
```

Cart Module

1. Adding Products to Cart and Display


Add to Cart and Display Flow
User clicks Add to Cart
Get productId and quantity
POST /api/cart
Backend - Add to Cart
Verify user authentication
User authenticated?
yes
no
Return 401 Unauthorized
Show login prompt
Get product by ID
Product exists?
yes
no
Return 400 error
Show Product not found
Get or create user cart
Search for existing item in cart
Item already in cart?
yes
no
Update quantity
Create new CartItem


Sync product info to CartItem
Copy productName, price, discount, imageUrl, stock
Save to carts.json
Return updated cart
Update cart icon count
Show success toast
User navigates to cart page
GET /api/cart
Backend - Get Cart Data
Verify user authentication
Get user cart items
Calculate cart total
Return items and total
Frontend - Render Cart
Store cartData globally
Cart empty?
no
yes
Show Your cart is empty
Render all item cards



Add to Cart Flow

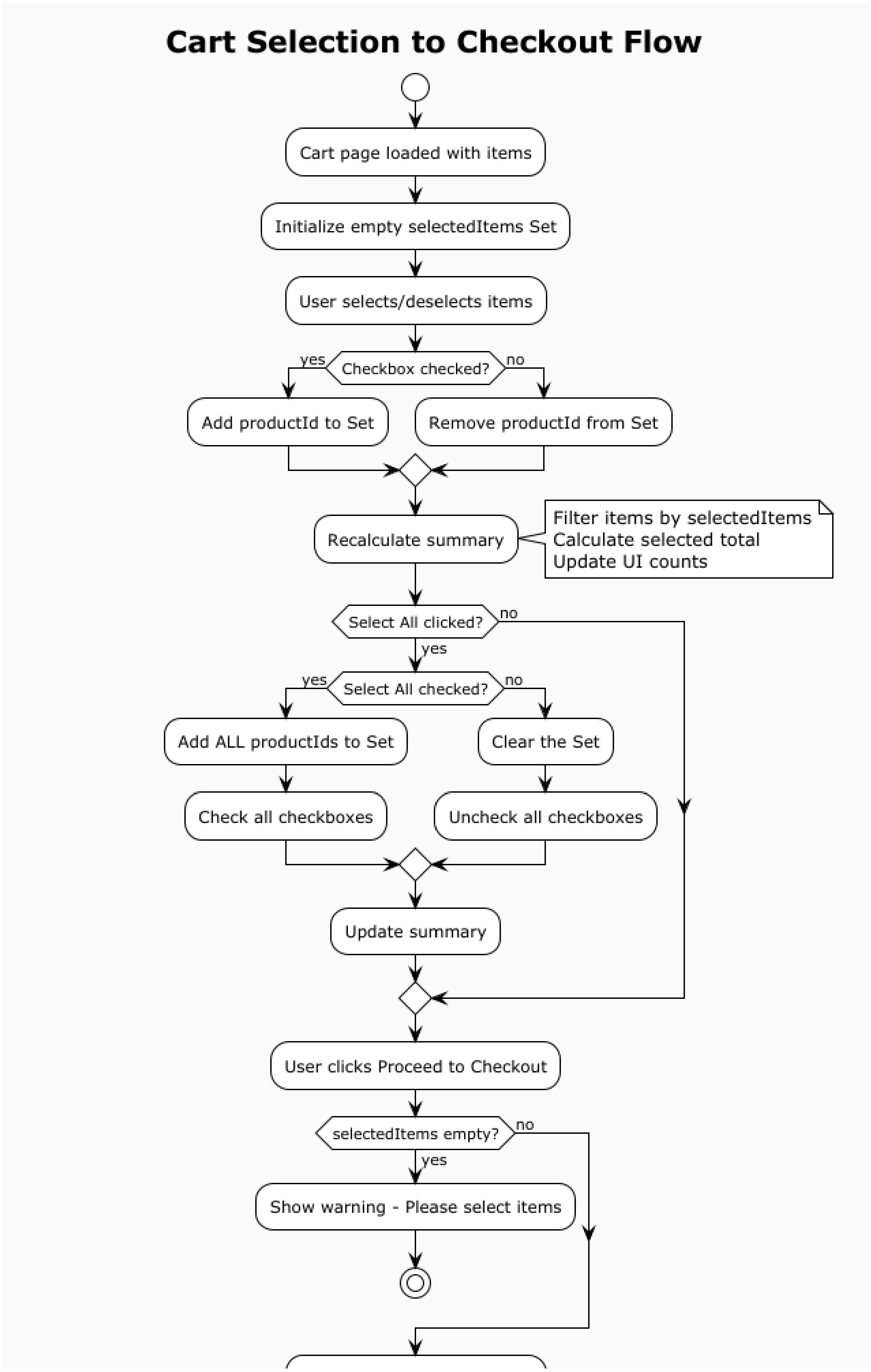
```java
// CartManager.java - addToCart()
public boolean addToCart(String userId, String productId, int quantity) {
    // Fetch current product info
    Product product = productManager.getProductById(productId);
    if (product == null) return false;

    // Get or create user's cart
    List<CartItem> cart = carts.computeIfAbsent(userId, k -> new ArrayList<>());

    // Check if item already exists
    for (CartItem item : cart) {
        if (item.getProductId().equals(productId)) {
            // Update quantity for existing item
            item.setQuantity(item.getQuantity() + quantity);
            updateCartItemFromProduct(item, product);  // Sync product info
            saveCarts();
            return true;
        }
    }

    // Add as new item
    CartItem newItem = new CartItem();
    newItem.setProductId(productId);
    newItem.setQuantity(quantity);
    updateCartItemFromProduct(newItem, product);  // Copy product details
    cart.add(newItem);
    saveCarts();
    return true;
}
```

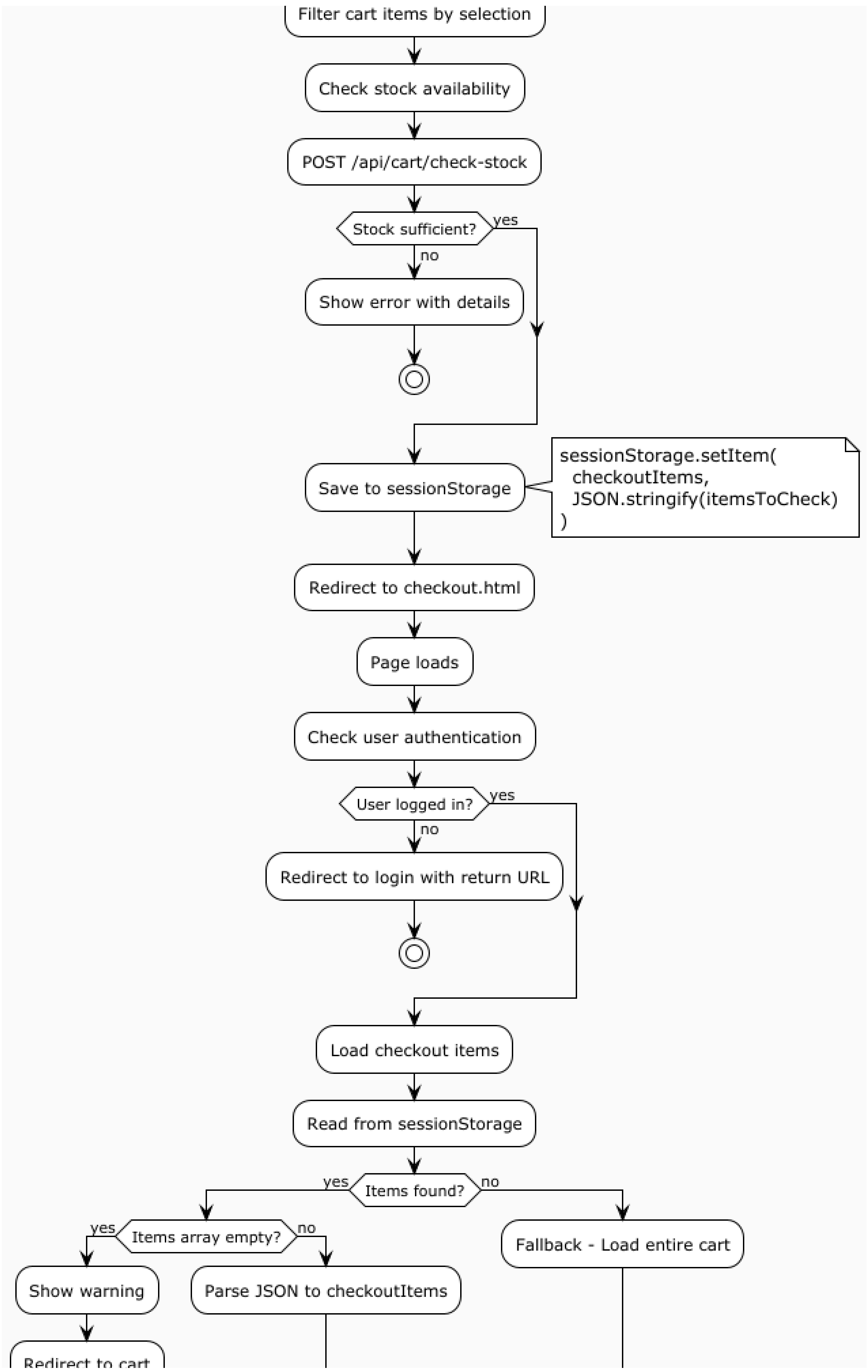
```java

// Sync cart item with current product data
private void updateCartItemFromProduct(CartItem item, Product product) {
    item.setProductName(product.getName());
    item.setPrice(product.getPrice());
    item.setDiscount(product.getDiscount());
    item.setImageUrl(product.getImageUrl());
    item.setStock(product.getStock());
}
```

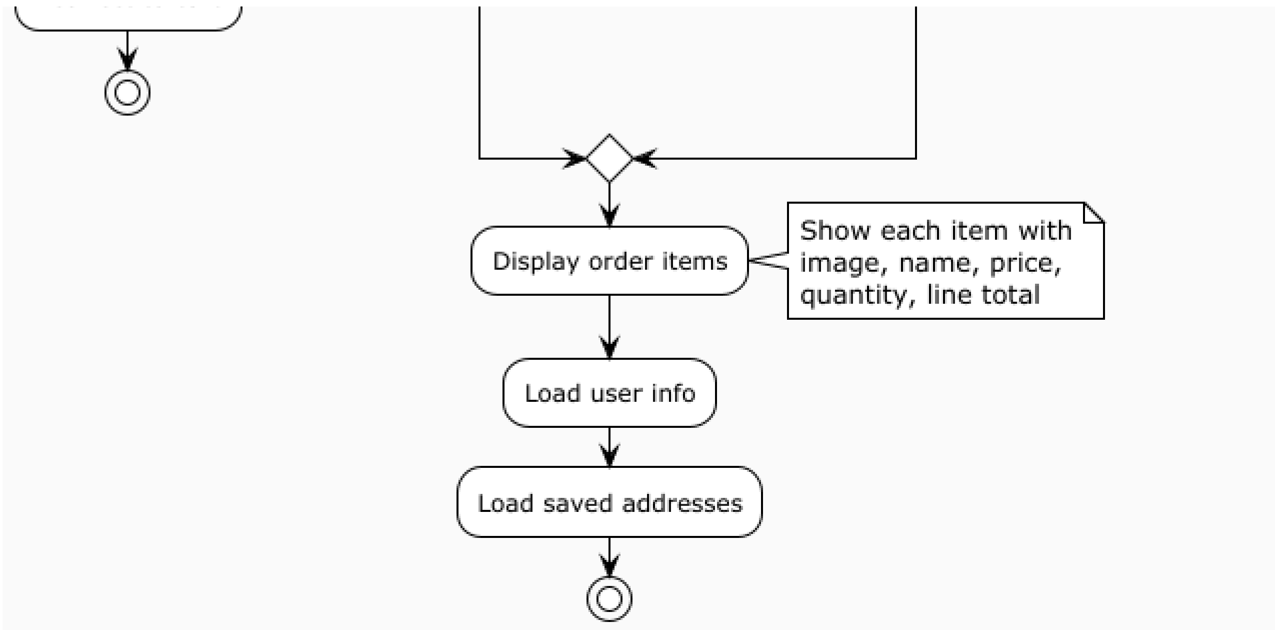
Cart Display with Totals

```javascript
// cart.js - loadCart()
async function loadCart() {
    const result = await API.getCart();

    if (result.success && result.data) {
        cartData = result.data;  // Store globally

        if (cartData.items.length === 0) {
            $('#emptyCart').show();
        } else {
            renderCartItems(cartData.items);
            updateSummary();
        }
    }
}

// Calculate totals for selected items
function updateSummary() {
    let selectedTotal = 0;
    let selectedDiscount = 0;

    cartData.items
        .filter(item => selectedItems.has(item.productId))
        .forEach(item => {
            const originalPrice = item.price * item.quantity;
            const discountedPrice = item.price * (1 - item.discount) * item.quantity;
            selectedTotal += discountedPrice;
            selectedDiscount += (originalPrice - discountedPrice);
        });

    $('#selectedCount').text(selectedItems.size);
    $('#selectedTotal').text(selectedTotal.toFixed(2));
    $('#discountAmount').text(selectedDiscount.toFixed(2));
}
```

2. Passing Selected Items to Checkout


Cart Selection to Checkout Flow
Cart page loaded with items
Initialize empty selectedItems Set
User selects/deselects items
Checkbox checked?
yes
no
Add productId to Set
Remove productId from Set
Recalculate summary
Filter items by selectedItems
Calculate selected total
Update UI counts
Select All clicked?
no
yes
Select All checked?
yes
no
Add ALL productIds to Set
Clear the Set
Check all checkboxes
Uncheck all checkboxes
Update summary
User clicks Proceed to Checkout
selectedItems empty?
no
yes
Show warning - Please select items


Filter cart items by selection
Check stock availability
POST /api/cart/check-stock
Stock sufficient?
yes
no
Show error with details
Save to sessionStorage
sessionStorage.setItem(
  checkoutItems,
  JSON.stringify(itemsToCheck)
)
Redirect to checkout.html
Page loads
Check user authentication
User logged in?
yes
no
Redirect to login with return URL
Load checkout items
Read from sessionStorage
yes
Items found?
no
yes
Items array empty?
no
Fallback - Load entire cart
Show warning
Parse JSON to checkoutItems
Redirect to cart


Display order items
Show each item with image, name, price, quantity, line total
Load user info
Load saved addresses

The cart uses a **Set-based selection system** and **sessionStorage** for checkout transfer:

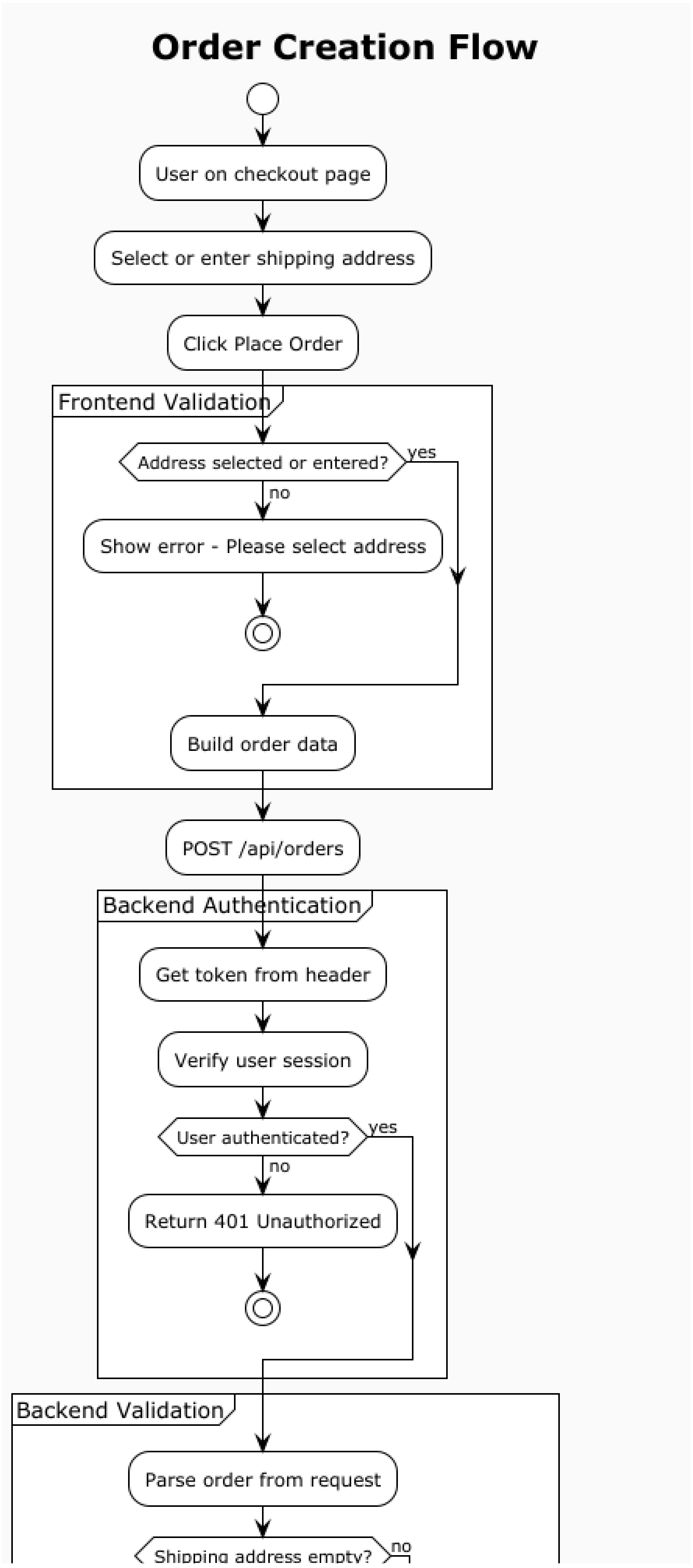
```
// cart.js - Selection management
let selectedItems = new Set();  // Stores selected productIds

// Checkbox handler
$(document).on('change', '.item-checkbox', function() {
    const productId = $(this).data('product-id');
    if ($(this).is(':checked')) {
        selectedItems.add(productId);
    } else {
        selectedItems.delete(productId);
    }
    updateSummary();
});

// Proceed to checkout
$('#checkoutBtn').on('click', async function() {
    if (selectedItems.size === 0) {
        Utils.showToast('Please select items to checkout', 'warning');
        return;
    }

    // Filter selected items from cart data
    const itemsToCheck = cartData.items.filter(
        item => selectedItems.has(item.productId)
    );

    // Stock validation
    const stockResult = await API.checkStock(itemsToCheck);
    if (!stockResult.success) {
        Utils.showToast(stockResult.message, 'error');
        return;
    }

    // Pass selected items via sessionStorage
    sessionStorage.setItem('checkoutItems', JSON.stringify(itemsToCheck));
    window.location.href = '/checkout.html';
});
```

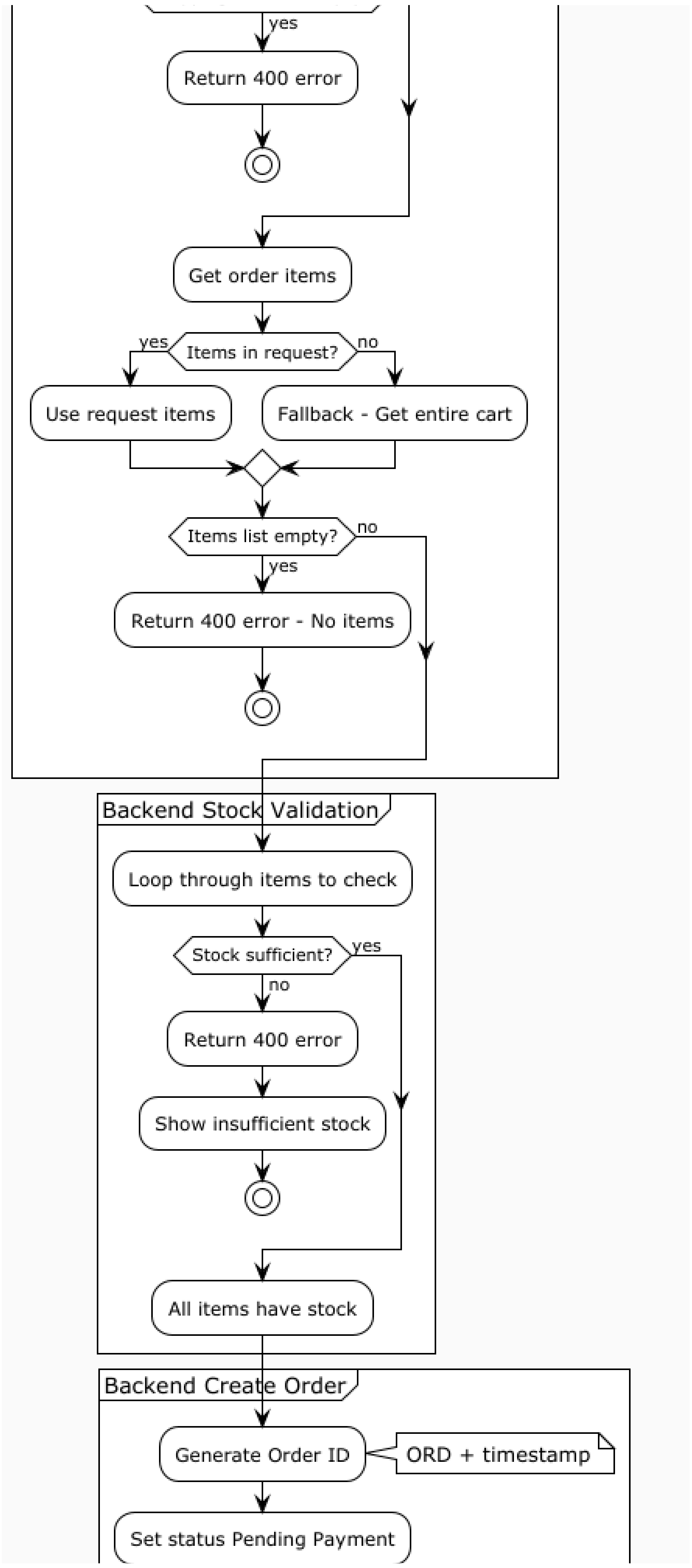
Checkout Page Retrieval:

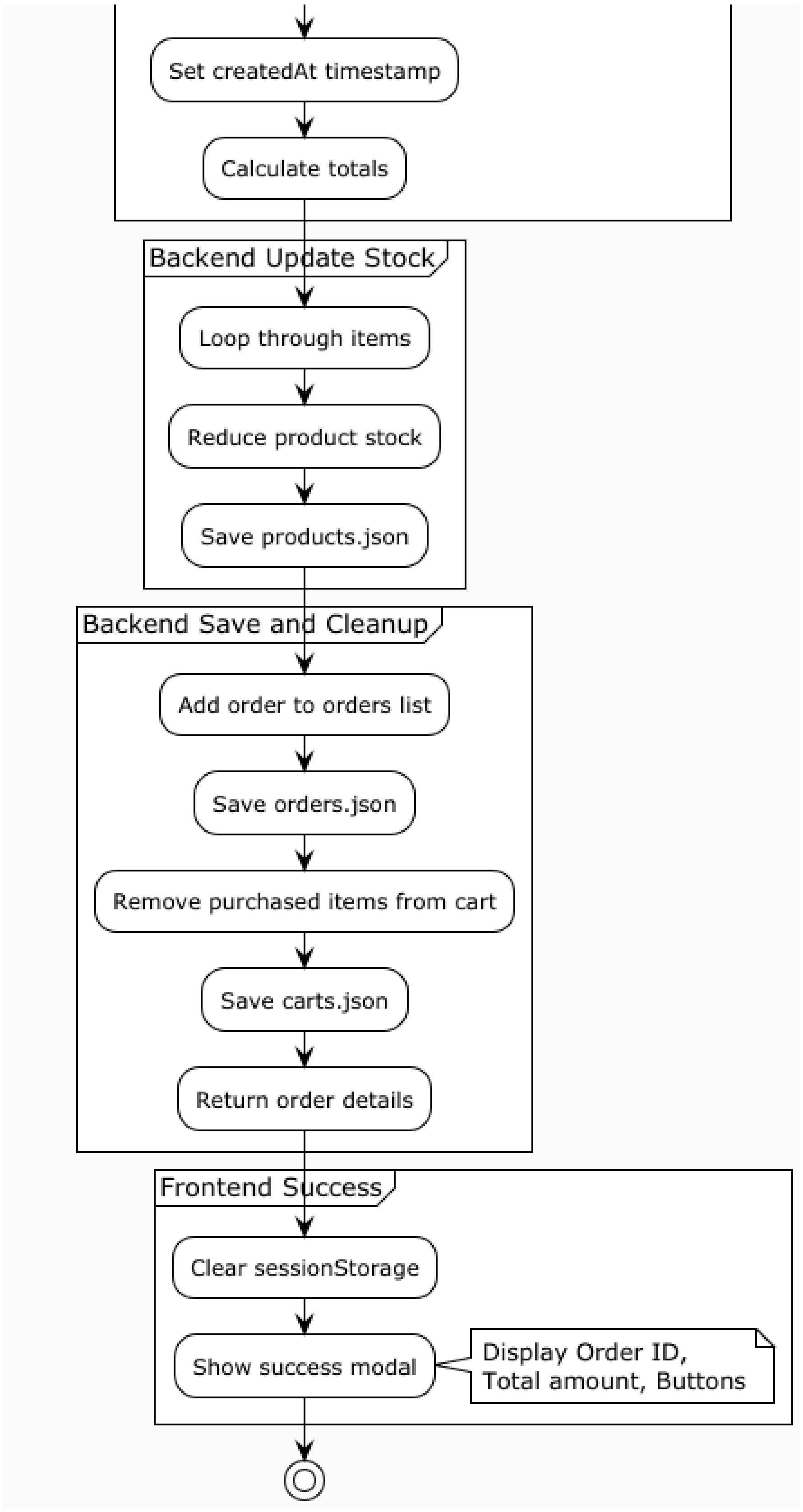
```
// checkout.js - loadCheckoutItems()
function loadCheckoutItems() {
    const itemsStr = sessionStorage.getItem('checkoutItems');

    if (itemsStr) {
        checkoutItems = JSON.parse(itemsStr);
        if (!checkoutItems || checkoutItems.length === 0) {
            window.location.href = '/cart.html';  // Redirect if empty
            return;
        }
        displayOrderItems();
    } else {
        // Fallback: load entire cart (backward compatibility)
        loadFullCart();
    }
}
```

Order Module

Order Generation Flow


Order Creation Flow
User on checkout page
Select or enter shipping address
Click Place Order
Frontend Validation
Address selected or entered?
yes
no
Show error - Please select address
Build order data
POST /api/orders
Backend Authentication
Get token from header
Verify user session
User authenticated?
yes
no
Return 401 Unauthorized
Backend Validation
Parse order from request
Shipping address empty?
no


yes
Return 400 error
Get order items
Items in request?
yes
no
Use request items
Fallback - Get entire cart
Items list empty?
no
yes
Return 400 error - No items
Backend Stock Validation
Loop through items to check
Stock sufficient?
yes
no
Return 400 error
Show insufficient stock
All items have stock
Backend Create Order
Generate Order ID
ORD + timestamp
Set status Pending Payment



Complete Order Creation Process

```java
// OrderManager.java - createOrder()
public Order createOrder(Order order) {
    // 1. Generate unique order ID with timestamp
    String orderId = "ORD" + System.currentTimeMillis();
    order.setId(orderId);

    // 2. Set initial status
    order.setStatus("Pending Payment");

    // 3. Set creation timestamp
    order.setCreatedAt(LocalDateTime.now()
        .toString().replace("T", " ").substring(0, 19));

    // 4. Calculate totals (with discount)
    double total = order.getItems().stream()
        .mapToDouble(item ->
            item.getPrice() * (1 - item.getDiscount()) * item.getQuantity())
        .sum();
    order.setTotalAmount(total);

    // 5. Calculate original total and discount amount
    double originalTotal = order.getItems().stream()
        .mapToDouble(item -> item.getPrice() * item.getQuantity())
        .sum();
    order.setOriginalTotal(originalTotal);
    order.setDiscountTotal(originalTotal - total);

    // 6. Deduct stock for each item
    for (CartItem item : order.getItems()) {
        productManager.updateStock(item.getProductId(), item.getQuantity());
    }

    // 7. Save order
    orders.add(order);
    saveOrders();

    // 8. Remove purchased items from cart
    cartManager.removeItems(order.getUserId(), order.getItems());

    return order;
}
```

Controller Layer Order Creation

```java
// OrderController.java - POST /api/orders
post("/api/orders", (req, res) -> {
    res.type("application/json");
    String userId =
UserController.getCurrentUserId(req.headers("Authorization"));

    if (userId == null) {
        res.status(401);
        return gson.toJson(ApiResponse.error("Please login first"));
    }

    // Parse order from request
    Order orderInfo = gson.fromJson(req.body(), Order.class);

    // Validate shipping address
    if (orderInfo.getShippingAddress() == null ||
        orderInfo.getShippingAddress().isEmpty()) {
        res.status(400);
        return gson.toJson(ApiResponse.error("Please provide shipping address"));
    }

    // Get items (from request or cart fallback)
    List<CartItem> selectedItems = orderInfo.getItems();
    if (selectedItems == null || selectedItems.isEmpty()) {
        selectedItems = dataStore.getCart(userId);
    }

    if (selectedItems.isEmpty()) {
        res.status(400);
        return gson.toJson(ApiResponse.error("Select items to checkout"));
    }

    // Stock validation for each item
    for (CartItem item : selectedItems) {
        if (!dataStore.checkStock(item.getProductId(), item.getQuantity())) {
            Product p = dataStore.getProductById(item.getProductId());
            res.status(400);
            return gson.toJson(ApiResponse.error(
                "Product " + item.getProductName() +
                " insufficient stock (Stock: " + p.getStock() + ")"));
        }
    }

    // Build and create order
    Order order = new Order();
    order.setUserId(userId);
    order.setItems(selectedItems);
    order.setShippingAddress(orderInfo.getShippingAddress());
    order.setPaymentMethod(orderInfo.getPaymentMethod() != null ?
        orderInfo.getPaymentMethod() : "Online Payment");
    order.setContactName(orderInfo.getContactName());
    order.setContactPhone(orderInfo.getContactPhone());

    Order createdOrder = dataStore.createOrder(order);

    return gson.toJson(ApiResponse.success("Order created successfully",
createdOrder));
});
```

Frontend Order Submission

```
// checkout.js - submitOrder()
async function submitOrder() {
    // Validate address selection
    if (!selectedAddressId && !$('#useNewAddress').is(':checked')) {
        Utils.showToast('Please select a shipping address', 'warning');
        return;
    }

    // Build shipping address string
    const address = getSelectedAddressString();
    const contactInfo = getContactInfo();

    const orderData = {
        items: checkoutItems,  // From sessionStorage
        shippingAddress: address,
        contactName: contactInfo.name,
        contactPhone: contactInfo.phone,
        paymentMethod: 'Online Payment'
    };

    const result = await API.createOrder(orderData);

    if (result.success) {
        // Clear checkout items from session
        sessionStorage.removeItem('checkoutItems');
```

```

            // Show success modal with order details
            showSuccessModal(result.data);
        } else {
            Utils.showToast(result.message, 'error');
        }
    }
```

📡 API Reference

Product Endpoints

Method	Path	Description
GET	`/api/products`	Get products (supports filtering and sorting)
GET	`/api/products/:id`	Get single product details
GET	`/api/categories`	Get all categories

User Endpoints

Method	Path	Description
POST	`/api/auth/login`	User login
POST	`/api/auth/register`	User registration
POST	`/api/auth/logout`	Logout
GET	`/api/auth/me`	Get current user info
PUT	`/api/auth/me`	Update user profile
GET	`/api/addresses`	Get address list
POST	`/api/addresses`	Add new address

Method	Path	Description
PUT	`/api/addresses/:id`	Update address
DELETE	`/api/addresses/:id`	Delete address

Cart Endpoints

Method	Path	Description
GET	`/api/cart`	Get cart contents
POST	`/api/cart`	Add item to cart
PUT	`/api/cart/:productId`	Update cart item quantity
DELETE	`/api/cart/:productId`	Remove item from cart
DELETE	`/api/cart`	Clear entire cart
POST	`/api/cart/check-stock`	Validate stock availability

Order Endpoints

Method	Path	Description
POST	`/api/orders`	Create new order
GET	`/api/orders`	Get user’s orders
GET	`/api/orders/:id`	Get order details
