Masroofy Wallet - Feature Implementation Documentation

Author: AI Assistant
Date: December 20, 2025
Project: Masroofy - Family Wallet Management System

Table of Contents

Overview

This document outlines the features implemented during this development session for the Masroofy wallet application. The Masroofy project is a MERN stack (MongoDB, Express, React, Node.js) family wallet management system that allows parents to manage allowances for their children.

Features Implemented:

1. **Expense Recording** - Allow users to record and track their expenses
2. **Child Transaction History** - Allow parents to view their children's transaction history
3. **CSS Enhancements** - Premium styling for modals and transaction tables

---

Feature 1: Expense Recording & Tracking

Purpose

Enable both parents and children to record their expenses directly from their dashboards, helping them track where their money goes.

Files Modified

1. Frontend API (`frontend/src/api.js`)

```
// Added createExpense method to the transactions API
export const transactionsAPI = {
    getAll: (type) => api.get('/transactions', { params: { type } }),
    create: (data) => api.post('/transactions', data),
    createExpense: (data) => api.post('/transactions', data),
    getChildTransactions: (childId) => api.get(`/transactions/child/${childId}`)
};

// Alias for ExpenseModal component compatibility
export const transactionAPI = transactionsAPI;
```

Explanation: The `createExpense` method calls the POST `/api/transactions` endpoint with expense data. The `transactionAPI` alias provides backward compatibility with the existing `ExpenseModal` component.

---

2. Parent Dashboard (`frontend/src/components/ParentDashboard.jsx`)

Added State:

```
const [showExpenseModal, setShowExpenseModal] = useState(false);
```

Added Action Card:

```
<div className="card card-clickable action-card" onClick={() => setShowExpenseModal(true)}>
    <div className="action-card-icon">🧾</div>
    <h3>Record Expense</h3>
    <p className="text-secondary">Track where your money goes</p>
    <button className="btn btn-primary btn-sm" style={{ marginTop: '1rem' }}>
        <span>Add Expense</span>
    </button>
</div>
```

Added Modal Render:

```
{showExpenseModal && (
    <ExpenseModal
        isOpen={showExpenseModal}
        onClose={() => setShowExpenseModal(false)}
        onSuccess={refreshData}
    />
)}
```

Explanation: A new "Record Expense" card was added to the action cards grid. When clicked, it opens the ExpenseModal for recording expenses. The `onSuccess` callback refreshes the dashboard data after a successful expense recording.

---

3. Child Dashboard (`frontend/src/components/ChildDashboard.jsx`)

Similar implementation to Parent Dashboard - added a "Record Expense" card and integrated the ExpenseModal component.

Feature 2: Child Transaction History

Purpose

Allow parents to view the complete transaction history of any of their children by clicking on the child's name in the Child Accounts section.

Files Modified

1. Backend API (`04_omar_samer_transaction_history.js`)

New Endpoint: GET /api/transactions/child/:childId

```
// Get child's transactions (parent only)
router.get('/child/:childId', authMiddleware, async (req, res) => {
    try {
        // Only parents can view child transactions
        if (req.user.role !== 'parent') {
            return res.status(403).json({
                success: false,
                message: 'Only parents can view child transactions'
            });
        }

        const { childId } = req.params;

        // Verify the child belongs to this parent
        const parent = await User.findById(req.user._id).populate('children');
        const isParentOfChild = parent.children.some(
            child => child._id.toString() === childId
        );

        if (!isParentOfChild) {
            return res.status(403).json({
                success: false,
                message: 'This child does not belong to your account'
            });
        }

        // Get all transactions where child is sender or receiver
        const transactions = await Transaction.find({
            $or: [
                { senderId: childId },
                { receiverId: childId }
            ]
        })
            .populate('senderId', 'username')
            .populate('receiverId', 'username')
            .sort({ timestamp: -1 })
            .limit(100);

        // Format transactions for response
        const formattedTransactions = transactions.map(tx => {
            const isIncoming = tx.receiverId &&
                               tx.receiverId._id.toString() === childId;
            return {
                id: tx._id,
                type: tx.type,
                amount: tx.amount,
                direction: isIncoming ? 'incoming' : 'outgoing',
                sender: tx.senderId ? tx.senderId.username : 'External',
                receiver: tx.receiverId ? tx.receiverId.username : 'Unknown',
                description: tx.description,
                timestamp: tx.timestamp,
                date: tx.timestamp.toLocaleDateString(),
                time: tx.timestamp.toLocaleTimeString()
            };
        });

        res.json({
            success: true,
            transactions: formattedTransactions,
            count: formattedTransactions.length
        });
    } catch (error) {
        console.error('Get child transactions error:', error);
        res.status(500).json({
            success: false,
            message: 'Server error fetching child transactions'
        });
    }
});
```

Explanation:

- **Authorization:** Only users with role `parent` can access this endpoint
- **Ownership Check:** Verifies the requested child belongs to the authenticated parent
- **Query:** Finds all transactions where the child is either sender or receiver
- **Formatting:** Determines if each transaction is incoming or outgoing from the child's perspective

---

2. Frontend API Addition

```
getChildTransactions: (childId) => api.get(`/transactions/child/${childId}`)
```

---

3. New Component: ChildTransactionModal (`frontend/src/components/ChildTransactionModal.jsx`)

```
import React, { useState, useEffect, useRef } from 'react';
import ReactDOM from 'react-dom';
import { transactionsAPI } from '../api';

function ChildTransactionModal({ isOpen, onClose, child }) {
    const [transactions, setTransactions] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState('');
    const lastChildId = useRef(null);

    useEffect(() => {
        const childId = child?.id || child?._id;

        // Only fetch if modal is open and child ID changed
        if (isOpen && childId && childId !== lastChildId.current) {
            lastChildId.current = childId;
            fetchChildTransactions(childId);
        }
        // Reset when modal closes
        if (!isOpen) {
            lastChildId.current = null;
        }
    }, [isOpen, child?.id, child?._id]);

    const fetchChildTransactions = async (childId) => {
        setLoading(true);
        setError('');
        try {
            const response = await transactionsAPI.getChildTransactions(childId);
            if (response.data.success) {
                setTransactions(response.data.transactions);
            }
        } catch (err) {
            setError(err.response?.data?.message || 'Failed to load');
        } finally {
            setLoading(false);
        }
    };

    if (!isOpen) return null;

    // Use React Portal to render at document.body level
    return ReactDOM.createPortal(
        <div className="modal-overlay" onClick={onClose}>
            {/* Modal content */}
        </div>,
        document.body
    );
}
```

Key Technical Points:

1. **useRef for preventing loops:** The `lastChildId` ref prevents infinite re-fetching by tracking the last fetched child ID.
2. **React Portal:** Uses `ReactDOM.createPortal` to render the modal directly to `document.body`. This solves z-index issues when the modal is rendered inside a nested component (like ChildAccounts inside a card).

---

4. Modified ChildAccounts Component

Made child items clickable:

```
<li
    key={child.id || child._id}
    className="child-item"
    onClick={() => setSelectedChild(child)}
    style={{ cursor: 'pointer' }}
    title="Click to view transactions"
>
    <div className="child-username">
        {child.username}
        <span style={{ fontSize: '0.75rem', color: 'var(--text-secondary)' }}>
            📊 View History
        </span>
    </div>
    <div className="child-balance">
        EGP {(child.balance || 0).toFixed(2)}
    </div>
</li>
```

Feature 3: CSS Enhancements

Modal Overlay Enhancement (`frontend/src/index.css`)

Before:

```css
.modal-overlay {
    background: rgba(0, 0, 0, 0.75);
    backdrop-filter: blur(4px);
}
```

After:

```css
.modal-overlay {
    background: linear-gradient(135deg,
        rgba(15, 23, 42, 0.95),
        rgba(30, 41, 59, 0.9));
    backdrop-filter: blur(8px) saturate(150%);
}

.modal-overlay::before {
    content: '';
    position: absolute;
    top: 0; left: 0; right: 0; bottom: 0;
    background: radial-gradient(
        circle at 50% 30%,
        rgba(59, 130, 246, 0.15),
        transparent 60%
    );
}
```

Explanation: The overlay now uses a gradient instead of plain dark overlay, with enhanced blur and a subtle blue radial glow for a premium feel.

Transaction Table Enhancements

New Features:

- Rounded corners on table headers
- Row striping with alternating backgrounds
- Hover effects with scale transform
- Glow shadows on amounts (green for income, red for expenses)
- Monospace font for amounts

```css
.transaction-table th {
    background: rgba(59, 130, 246, 0.1);
    border-bottom: 2px solid var(--primary);
    color: var(--primary-light);
    font-weight: 700;
}

.transaction-table tbody tr:hover {
    background: rgba(59, 130, 246, 0.08);
    transform: scale(1.01);
    box-shadow: 0 4px 20px rgba(0, 0, 0, 0.3);
}

.transaction-amount.positive {
    color: var(--success);
    text-shadow: 0 0 10px rgba(16, 185, 129, 0.3);
}
```

Badge Styling

New badge classes with gradient backgrounds:

```css
.badge-success {
    background: linear-gradient(135deg,
        rgba(16, 185, 129, 0.2),
        rgba(16, 185, 129, 0.1));
    color: var(--success-light);
    border: 1px solid rgba(16, 185, 129, 0.3);
    box-shadow: 0 2px 8px rgba(16, 185, 129, 0.2);
}
```

Technical Implementation Details

Architecture

```
┌─────────────────┐     ┌─────────────────┐     ┌─────────────┐
│   React App     │────▶│  Express API    │────▶│   MongoDB   │
│   (Frontend)    │     │   (Backend)     │     │  (Database) │
└─────────────────┘     └─────────────────┘     └─────────────┘
```

Key Technologies Used

- **React 18** - Frontend framework

- **React Router** - Client-side routing
- **Axios** - HTTP client
- **Express.js** - Backend framework
- **MongoDB/Mongoose** - Database
- **JWT** - Authentication

File Structure of Modified Files

```
Masroofy/
├── 04_omar_samer_transaction_history.js  (Backend - new endpoint)
├── frontend/
│   └── src/
│       ├── api.js                         (API methods)
│       ├── index.css                      (Enhanced styles)
│       └── components/
│           ├── ParentDashboard.jsx        (Expense modal integration)
│           ├── ChildDashboard.jsx         (Expense modal integration)
│           ├── ChildTransactionModal.jsx  (NEW - Child history modal)
│           └── 06_bahaa_ahmed_ChildAccounts.jsx (Clickable children)
```

Summary

Feature	Files Modified	Lines Changed
Expense Recording	3 files	~50 lines
Child Transaction History	4 files	~200 lines
CSS Enhancements	1 file	~180 lines

All features have been tested and pushed to GitHub.