TRADEWISER EMERGENCY FIX - CLEAN IMPLEMENTATION

CONTEXT

TradeWiser dashboard is showing blank screen due to broken React Query setup. Need to fix core functionality and implement proper credit line model for agricultural commodity financing.

OBJECTIVE

Fix broken components, implement working deposit-to-credit-line workflow, and create clean professional interface.

PHASE 1: FIX CRITICAL DASHBOARD ERRORS

1.1 Fix Broken ZerodhaPortfolioDashboard Component

REPLACE the entire content of `client/src/components/portfolio/ZerodhaPortfolioDashboard.tsx`:

```
import React from 'react';
import { useQuery, useQueryClient } from '@tanstack/react-query';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
import { Button } from '@/components/ui/button';
import { Badge } from '@/components/ui/badge';
import { Progress } from '@/components/ui/progress';
import { useToast } from '@/hooks/use-toast';
import {
  TrendingUp,
  Wallet,
  FileText,
  CreditCard,
  Plus,
  Activity,
  Loader2
} from 'lucide-react';
import { Link } from 'wouter';

const PortfolioDashboard = () => {
  const { toast } = useToast();
  const queryClient = useQueryClient();

  // Fix broken query - add proper queryFn
  const { data: portfolioResponse, isLoading, error } = useQuery({
    queryKey: ['portfolio'],
    queryFn: async () => {
      const response = await fetch('/api/portfolio', {
        credentials: 'include',
        headers: {
```

```
          'Content-Type': 'application/json'
        }
      });

      if (!response.ok) {
        throw new Error('Failed to fetch portfolio data');
      }

      return response.json();
    },
    refetchInterval: 30000,
    retry: 3,
    staleTime: 10000
  });

  const { data: creditLineResponse } = useQuery({
    queryKey: ['credit-line'],
    queryFn: async () => {
      const response = await fetch('/api/credit-line/details', {
        credentials: 'include'
      });
      if (!response.ok) return { success: true, data: { totalLimit: 0, availableBalance:
      return response.json();
    },
    refetchInterval: 30000
  });

  if (isLoading) {
    return (
      <div className="flex items-center justify-center min-h-[400px]">
        <Loader2 className="w-8 h-8 animate-spin" />
      </div>
    );
  }

  if (error) {
    return (
      <div className="text-center py-12">
        <p className="text-red-600 mb-4">Failed to load portfolio data</p>
        <Button onClick={() => queryClient.refetchQueries(['portfolio'])}>
          Try Again
        </Button>
      </div>
    );
  }

  const portfolio = portfolioResponse?.data || {
    totalValue: 0,
    receiptsCount: 0,
    availableCredit: 0,
    receipts: [],
    commodities: []
  };

  const creditLine = creditLineResponse?.data || {
    totalLimit: 0,
```

```
    availableBalance: 0,
    outstandingAmount: 0
  };

  return (
    <div className="max-w-7xl mx-auto p-6 space-y-8">
      {/* Header */}
      <div>
        <h1 className="text-3xl font-bold text-gray-900">Portfolio Dashboard</h1>
        <p className="text-gray-600 mt-2">Manage your commodity holdings and financing</p
      </div>

      {/* Portfolio Overview */}
      <div className="grid grid-cols-1 md:grid-cols-4 gap-6">
        <Card>
          <CardContent className="p-6">
            <div className="flex items-center justify-between">
              <div>
                <p className="text-sm font-medium text-gray-600">Portfolio Value</p>
                <p className="text-2xl font-bold">₹{portfolio.totalValue.toLocaleString()
              </div>
              <TrendingUp className="w-8 h-8 text-blue-600" />
            </div>
          </CardContent>
        </Card>

        <Card>
          <CardContent className="p-6">
            <div className="flex items-center justify-between">
              <div>
                <p className="text-sm font-medium text-gray-600">Credit Available</p>
                <p className="text-2xl font-bold text-green-600">₹{creditLine.availableBa
              </div>
              <Wallet className="w-8 h-8 text-green-600" />
            </div>
          </CardContent>
        </Card>

        <Card>
          <CardContent className="p-6">
            <div className="flex items-center justify-between">
              <div>
                <p className="text-sm font-medium text-gray-600">Outstanding</p>
                <p className="text-2xl font-bold text-red-600">₹{creditLine.outstandingAm
              </div>
              <CreditCard className="w-8 h-8 text-red-600" />
            </div>
          </CardContent>
        </Card>

        <Card>
          <CardContent className="p-6">
            <div className="flex items-center justify-between">
              <div>
                <p className="text-sm font-medium text-gray-600">Active Receipts</p>
                <p className="text-2xl font-bold">{portfolio.receiptsCount}</p>
```

```
            </div>
            <FileText className="w-8 h-8 text-purple-600" />
          </div>
        </CardContent>
      </Card>
    </div>

    {/* Quick Actions */}
    <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
      <Card>
        <CardContent className="p-6">
          <h3 className="text-lg font-semibold mb-2">New Deposit</h3>
          <p className="text-gray-600 mb-4">Add commodities to your portfolio</p>
          <Link href="/deposits/new">
            <Button className="w-full">
              <Plus className="w-4 h-4 mr-2" />
              Deposit Commodity
            </Button>
          </Link>
        </CardContent>
      </Card>

      <Card>
        <CardContent className="p-6">
          <h3 className="text-lg font-semibold mb-2">Credit Line</h3>
          <p className="text-gray-600 mb-4">Withdraw funds against your holdings</p>
          <Link href="/credit-line">
            <Button className="w-full" variant="outline">
              <Wallet className="w-4 h-4 mr-2" />
              Manage Credit
            </Button>
          </Link>
        </CardContent>
      </Card>

      <Card>
        <CardContent className="p-6">
          <h3 className="text-lg font-semibold mb-2">View Receipts</h3>
          <p className="text-gray-600 mb-4">Manage warehouse receipts</p>
          <Link href="/receipts">
            <Button className="w-full" variant="outline">
              <FileText className="w-4 h-4 mr-2" />
              View All
            </Button>
          </Link>
        </CardContent>
      </Card>
    </div>

    {/* Holdings Table */}
    <Card>
      <CardHeader>
        <CardTitle>Holdings ({portfolio.receipts.length} positions)</CardTitle>
      </CardHeader>
      <CardContent>
        {portfolio.receipts.length === 0 ? (
```

```
            <div className="text-center py-8">
              <FileText className="w-12 h-12 text-gray-400 mx-auto mb-4" />
              <h3 className="text-lg font-semibold text-gray-900 mb-2">No holdings yet</h
              <p className="text-gray-600 mb-4">Start by depositing your first commodity<
              <Link href="/deposits/new">
                <Button>Create First Deposit</Button>
              </Link>
            </div>
          ) : (
            <div className="overflow-x-auto">
              <table className="w-full">
                <thead>
                  <tr className="border-b">
                    <th className="text-left p-2">Commodity</th>
                    <th className="text-left p-2">Quantity</th>
                    <th className="text-left p-2">Value</th>
                    <th className="text-left p-2">Status</th>
                  </tr>
                </thead>
                <tbody>
                  {portfolio.receipts.map((receipt) => (
                    <tr key={receipt.id} className="border-b">
                      <td className="p-2">
                        <div>
                          <p className="font-medium">{receipt.commodityName || 'Unknown'}
                          <p className="text-sm text-gray-500">{receipt.receiptNumber}</p
                        </div>
                      </td>
                      <td className="p-2">
                        {parseFloat(receipt.quantity).toFixed(2)} {receipt.measurementUni
                      </td>
                      <td className="p-2">₹{parseFloat(receipt.valuation || 0).toLocaleSt
                      <td className="p-2">
                        <Badge variant={receipt.status === 'active' ? 'default' : 'second
                          {receipt.status}
                        </Badge>
                      </td>
                    </tr>
                  ))}
                </tbody>
              </table>
            </div>
          )}
        </CardContent>
      </Card>
    </div>
  );
};

export default PortfolioDashboard;
```

1.2 Fix Portfolio API Endpoint

UPDATE `server/routes.ts` to add working portfolio endpoint:

```
// Add this to server/routes.ts
app.get('/api/portfolio', async (req, res) => {
  try {
    if (!req.user) {
      return res.status(401).json({ success: false, error: 'Not authenticated' });
    }

    // Get receipts with commodity data
    const receipts = await db.select({
      id: warehouseReceipts.id,
      receiptNumber: warehouseReceipts.receiptNumber,
      commodityName: commodities.name,
      quantity: warehouseReceipts.quantity,
      valuation: warehouseReceipts.valuation,
      qualityGrade: warehouseReceipts.qualityGrade,
      status: warehouseReceipts.status,
      collateralUsed: warehouseReceipts.collateralUsed,
      measurementUnit: commodities.unit,
      createdAt: warehouseReceipts.createdAt
    })
    .from(warehouseReceipts)
    .leftJoin(commodities, eq(warehouseReceipts.commodityId, commodities.id))
    .where(eq(warehouseReceipts.ownerId, req.user.id));

    // Calculate portfolio metrics
    const totalValue = receipts.reduce((sum, receipt) =>
      sum + (parseFloat(receipt.valuation) || 0), 0);

    const totalCollateralUsed = receipts.reduce((sum, receipt) =>
      sum + (parseFloat(receipt.collateralUsed) || 0), 0);

    const availableCredit = Math.max(0, (totalValue * 0.8) - totalCollateralUsed);

    res.json({
      success: true,
      data: {
        totalValue,
        receiptsCount: receipts.length,
        availableCredit,
        receipts,
        commodities: receipts
      }
    });
  } catch (error) {
    console.error('Portfolio API error:', error);
    res.status(500).json({
      success: false,
      error: error.message,
      data: {
        totalValue: 0,
        receiptsCount: 0,
        availableCredit: 0,
```

```
        receipts: [],
        commodities: []
      }
    });
  }
});
```

PHASE 2: IMPLEMENT CREDIT LINE SYSTEM

2.1 Credit Line API Endpoints

ADD these endpoints to `server/routes.ts`:

```
// Credit Line Details
app.get('/api/credit-line/details', async (req, res) => {
  try {
    if (!req.user) {
      return res.status(401).json({ success: false, error: 'Not authenticated' });
    }

    // Mock data for now - replace with actual NBFC API call
    const mockCreditLine = {
      totalLimit: 500000,
      availableBalance: 350000,
      outstandingAmount: 150000,
      interestRate: 12.0,
      dailyInterest: Math.round((150000 * 12 / 100 / 365) * 100) / 100,
      monthlyInterest: Math.round((150000 * 12 / 100 / 12) * 100) / 100,
      lastPaymentDate: '2025-09-01'
    };

    // TODO: Replace with actual NBFC API call
    // const response = await fetch(`${process.env.NBFC_API_URL}/credit-line/${req.user.i
    //   headers: {
    //     'Authorization': `Bearer ${process.env.NBFC_API_KEY}`,
    //     'Content-Type': 'application/json'
    //   }
    // });

    res.json({
      success: true,
      data: mockCreditLine
    });
  } catch (error) {
    console.error('Credit line details error:', error);
    res.status(500).json({ success: false, error: error.message });
  }
});

// Withdraw Money
app.post('/api/credit-line/withdraw', async (req, res) => {
  try {
    const { amount, purpose } = req.body;
```

```
    if (!req.user) {
      return res.status(401).json({ success: false, error: 'Not authenticated' });
    }

    if (!amount || amount <= 0) {
      return res.status(400).json({ success: false, error: 'Invalid amount' });
    }

    // Mock response - replace with actual NBFC API call
    const mockResponse = {
      transactionId: `TXN${Date.now()}`,
      amount: parseFloat(amount),
      purpose: purpose || 'Working Capital',
      status: 'success',
      timestamp: new Date().toISOString()
    };

    // TODO: Replace with actual NBFC API call
    // const response = await fetch(`${process.env.NBFC_API_URL}/withdraw`, {
    //   method: 'POST',
    //   headers: {
    //     'Authorization': `Bearer ${process.env.NBFC_API_KEY}`,
    //     'Content-Type': 'application/json'
    //   },
    //   body: JSON.stringify({
    //     userId: req.user.id,
    //     amount: parseFloat(amount),
    //     purpose,
    //     collateralType: 'warehouse_receipts'
    //   })
    // });

    res.json({
      success: true,
      data: mockResponse,
      message: `₹${amount} withdrawn successfully`
    });
  } catch (error) {
    console.error('Withdrawal error:', error);
    res.status(500).json({ success: false, error: error.message });
  }
});

// Repay Money
app.post('/api/credit-line/repay', async (req, res) => {
  try {
    const { amount } = req.body;

    if (!req.user) {
      return res.status(401).json({ success: false, error: 'Not authenticated' });
    }

    if (!amount || amount <= 0) {
      return res.status(400).json({ success: false, error: 'Invalid amount' });
    }
```

```
    // Mock response - replace with actual NBFC API call
    const mockResponse = {
      transactionId: `REP${Date.now()}`,
      amount: parseFloat(amount),
      status: 'success',
      timestamp: new Date().toISOString()
    };

    res.json({
      success: true,
      data: mockResponse,
      message: `₹${amount} repaid successfully`
    });
  } catch (error) {
    console.error('Repayment error:', error);
    res.status(500).json({ success: false, error: error.message });
  }
});
```

2.2 Credit Line Management Page

CREATE new file `client/src/pages/CreditLinePage.tsx`:

```
import React, { useState } from 'react';
import { useQuery, useQueryClient } from '@tanstack/react-query';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { useToast } from '@/hooks/use-toast';
import MainLayout from '@/components/layout/MainLayout';
import { Wallet, CreditCard, TrendingUp, Activity } from 'lucide-react';

const CreditLinePage = () => {
  const { toast } = useToast();
  const queryClient = useQueryClient();

  const [withdrawAmount, setWithdrawAmount] = useState('');
  const [repayAmount, setRepayAmount] = useState('');
  const [isWithdrawing, setIsWithdrawing] = useState(false);
  const [isRepaying, setIsRepaying] = useState(false);

  const { data: creditLineResponse, isLoading } = useQuery({
    queryKey: ['credit-line'],
    queryFn: async () => {
      const response = await fetch('/api/credit-line/details', {
        credentials: 'include'
      });
      if (!response.ok) throw new Error('Failed to fetch credit line data');
      return response.json();
    },
    refetchInterval: 30000
  });

  const creditLine = creditLineResponse?.data || {
```

```
    totalLimit: 0,
    availableBalance: 0,
    outstandingAmount: 0,
    interestRate: 12,
    dailyInterest: 0,
    monthlyInterest: 0
  };

  const handleWithdraw = async (e) => {
    e.preventDefault();
    if (!withdrawAmount || parseFloat(withdrawAmount) <= 0) {
      toast.error('Please enter a valid amount');
      return;
    }

    if (parseFloat(withdrawAmount) > creditLine.availableBalance) {
      toast.error('Amount exceeds available balance');
      return;
    }

    setIsWithdrawing(true);
    try {
      const response = await fetch('/api/credit-line/withdraw', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        credentials: 'include',
        body: JSON.stringify({
          amount: withdrawAmount,
          purpose: 'Working Capital'
        })
      });

      const result = await response.json();
      if (result.success) {
        toast.success(result.message);
        setWithdrawAmount('');
        queryClient.invalidateQueries(['credit-line']);
      } else {
        toast.error(result.error);
      }
    } catch (error) {
      toast.error('Withdrawal failed. Please try again.');
    } finally {
      setIsWithdrawing(false);
    }
  };

  const handleRepay = async (e) => {
    e.preventDefault();
    if (!repayAmount || parseFloat(repayAmount) <= 0) {
      toast.error('Please enter a valid amount');
      return;
    }

    if (parseFloat(repayAmount) > creditLine.outstandingAmount) {
      toast.error('Amount exceeds outstanding balance');
```

```
      return;
    }

    setIsRepaying(true);
    try {
      const response = await fetch('/api/credit-line/repay', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        credentials: 'include',
        body: JSON.stringify({ amount: repayAmount })
      });

      const result = await response.json();
      if (result.success) {
        toast.success(result.message);
        setRepayAmount('');
        queryClient.invalidateQueries(['credit-line']);
      } else {
        toast.error(result.error);
      }
    } catch (error) {
      toast.error('Repayment failed. Please try again.');
    } finally {
      setIsRepaying(false);
    }
  };

  if (isLoading) {
    return (
      <MainLayout>
        <div className="flex justify-center py-12">
          <div>Loading credit line data...</div>
        </div>
      </MainLayout>
    );
  }

  return (
    <MainLayout>
      <div className="max-w-6xl mx-auto p-6 space-y-8">
        <div>
          <h1 className="text-3xl font-bold">Credit Line Management</h1>
          <p className="text-gray-600 mt-2">Manage your credit line and payments</p>
        </div>

        {/* Credit Line Overview */}
        <div className="grid grid-cols-1 md:grid-cols-4 gap-6">
          <Card>
            <CardContent className="p-6">
              <div className="flex items-center justify-between">
                <div>
                  <p className="text-sm font-medium text-gray-600">Total Limit</p>
                  <p className="text-2xl font-bold">₹{creditLine.totalLimit.toLocaleStrin
                </div>
                <CreditCard className="w-8 h-8 text-blue-600" />
              </div>
```

```
            </CardContent>
          </Card>

          <Card>
            <CardContent className="p-6">
              <div className="flex items-center justify-between">
                <div>
                  <p className="text-sm font-medium text-gray-600">Available Balance</p>
                  <p className="text-2xl font-bold text-green-600">₹{creditLine.available
                </div>
                <Wallet className="w-8 h-8 text-green-600" />
              </div>
            </CardContent>
          </Card>

          <Card>
            <CardContent className="p-6">
              <div className="flex items-center justify-between">
                <div>
                  <p className="text-sm font-medium text-gray-600">Outstanding</p>
                  <p className="text-2xl font-bold text-red-600">₹{creditLine.outstanding
                </div>
                <TrendingUp className="w-8 h-8 text-red-600" />
              </div>
            </CardContent>
          </Card>

          <Card>
            <CardContent className="p-6">
              <div className="flex items-center justify-between">
                <div>
                  <p className="text-sm font-medium text-gray-600">Interest Rate</p>
                  <p className="text-2xl font-bold">{creditLine.interestRate}%</p>
                </div>
                <Activity className="w-8 h-8 text-purple-600" />
              </div>
            </CardContent>
          </Card>
        </div>

        {/* Interest Details */}
        <Card>
          <CardHeader>
            <CardTitle>Interest Information</CardTitle>
          </CardHeader>
          <CardContent>
            <div className="grid grid-cols-2 gap-6">
              <div className="flex justify-between py-2">
                <span className="text-gray-600">Daily Interest:</span>
                <span className="font-medium">₹{creditLine.dailyInterest.toLocaleString()
              </div>
              <div className="flex justify-between py-2">
                <span className="text-gray-600">Monthly Interest:</span>
                <span className="font-medium">₹{creditLine.monthlyInterest.toLocaleStrin
              </div>
            </div>
```

```
          </CardContent>
        </Card>

        {/* Actions */}
        <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
          {/* Withdraw */}
          <Card>
            <CardHeader>
              <CardTitle>Withdraw Funds</CardTitle>
            </CardHeader>
            <CardContent>
              <form onSubmit={handleWithdraw} className="space-y-4">
                <div>
                  <label className="block text-sm font-medium mb-2">Amount to Withdraw</l
                  <Input
                    type="number"
                    step="0.01"
                    placeholder="Enter amount"
                    value={withdrawAmount}
                    onChange={(e) => setWithdrawAmount(e.target.value)}
                    max={creditLine.availableBalance}
                    required
                  />
                  <p className="text-sm text-gray-500 mt-1">
                    Available: ₹{creditLine.availableBalance.toLocaleString()}
                  </p>
                </div>
                <Button
                  type="submit"
                  className="w-full"
                  disabled={isWithdrawing}
                >
                  {isWithdrawing ? 'Processing...' : 'Withdraw Funds'}
                </Button>
              </form>
            </CardContent>
          </Card>

          {/* Repay */}
          <Card>
            <CardHeader>
              <CardTitle>Make Payment</CardTitle>
            </CardHeader>
            <CardContent>
              <form onSubmit={handleRepay} className="space-y-4">
                <div>
                  <label className="block text-sm font-medium mb-2">Amount to Repay</labe
                  <Input
                    type="number"
                    step="0.01"
                    placeholder="Enter amount"
                    value={repayAmount}
                    onChange={(e) => setRepayAmount(e.target.value)}
                    max={creditLine.outstandingAmount}
                    required
                  />
```

```
                  <p className="text-sm text-gray-500 mt-1">
                    Outstanding: ₹{creditLine.outstandingAmount.toLocaleString()}
                  </p>
                </div>
                <Button
                  type="submit"
                  className="w-full bg-green-600 hover:bg-green-700"
                  disabled={isRepaying}
                >
                  {isRepaying ? 'Processing...' : 'Make Payment'}
                </Button>
              </form>
            </CardContent>
          </Card>
        </div>
      </div>
    </MainLayout>
  );
};

export default CreditLinePage;
```

2.3 Add Route for Credit Line

UPDATE `client/src/App.tsx` to add credit line route:

```
// Add this import at the top
import CreditLinePage from "@/pages/CreditLinePage";

// Add this route in the Router function
<Route path="/credit-line" component={CreditLinePage} />
```

PHASE 3: FIX DEPOSIT WORKFLOW

3.1 Fix Deposit API

UPDATE deposit endpoint in `server/routes.ts`:

```
app.post('/api/deposits', async (req, res) => {
  try {
    const { commodityName, commodityType, quantity, unit } = req.body;

    if (!req.user) {
      return res.status(401).json({ success: false, error: 'Not authenticated' });
    }

    // Validate input
    if (!commodityName || !quantity || quantity <= 0) {
      return res.status(400).json({ success: false, error: 'Invalid commodity data' });
    }
```

```
    // Get base price for commodity
    const commodityPrices = {
      'Wheat': 2500, 'Rice': 3000, 'Maize': 2000, 'Soybean': 4500,
      'Cotton': 6000, 'Sugarcane': 300, 'Chickpea': 5000, 'Turmeric': 8000
    };

    const basePrice = commodityPrices[commodityName] || 2500;
    const marketValue = basePrice * parseFloat(quantity);

    // Create commodity entry
    const commodity = await db.insert(commodities).values({
      ownerId: req.user.id,
      name: commodityName,
      category: commodityType,
      quantity: parseFloat(quantity),
      unit: unit || 'MT',
      marketValue: marketValue,
      qualityGrade: 'Grade A',
      status: 'deposited',
      createdAt: new Date()
    }).returning();

    // Immediately create warehouse receipt
    const receiptNumber = `TW${Date.now()}-${Math.random().toString(36).substr(2, 6).toUp

    const receipt = await db.insert(warehouseReceipts).values({
      receiptNumber,
      commodityId: commodity[0].id,
      ownerId: req.user.id,
      quantity: parseFloat(quantity),
      valuation: marketValue,
      qualityGrade: 'Grade A',
      status: 'active',
      availableForCollateral: true,
      collateralUsed: 0,
      blockchainHash: `BC-${Date.now()}`,
      createdAt: new Date()
    }).returning();

    res.json({
      success: true,
      data: {
        commodity: commodity[0],
        receipt: receipt[0],
        message: `Commodity deposited successfully! Receipt ${receiptNumber} generated.`
      }
    });
  } catch (error) {
    console.error('Deposit error:', error);
    res.status(500).json({ success: false, error: error.message });
  }
});
```

SUCCESS CRITERIA

✓ Dashboard Fixed

- Portfolio dashboard loads without blank screen
- Real data displayed properly
- Query functions work correctly

✓ Credit Line System Working

- Credit line details API functional
- Withdraw and repay functionality working
- Interest calculations displayed
- Integration ready for NBFC middleware

✓ Clean Interface

- No unnecessary external references
- Professional agricultural fintech design
- Clear navigation and actions
- Proper error handling

✓ Core Workflow Complete

- Deposit creates commodity and receipt immediately
- Portfolio updates in real-time
- Credit line management functional
- API-based calculations ready

IMPLEMENT THIS CLEAN FIX TO RESTORE TRADEWISER FUNCTIONALITY.