TRADEWISER MASTER IMPLEMENTATION PROMPT - COMPLETE PLATFORM FIX

CONTEXT

TradeWiser is a sophisticated agricultural commodity warehousing platform with existing TypeScript/React frontend, Express.js backend, PostgreSQL database with Drizzle ORM, and Docker deployment. The core infrastructure works but has critical business logic gaps, UX issues, and needs production-grade features.

OBJECTIVE

Implement comprehensive fixes for business logic gaps, streamline user experience, add enterprise monitoring, and create a complete production-ready platform in one implementation.

PART 1: CRITICAL BUSINESS LOGIC FIXES

1.1 Receipt-to-Loan Connection (HIGH PRIORITY)

PROBLEM: Generated warehouse receipts don't appear in loans section for collateralization.

IMPLEMENTATION:

```
// Loans page enhancement
const LoansPage = () => {
  const { data: eligibleReceipts } = useQuery(['eligible-receipts'], () =>
    fetch('/api/receipts?available_for_collateral=true').then(r => r.json())
  );

  const calculateLoanAmount = (receiptValue: number) => {
    return receiptValue * 0.8; // 80% collateral ratio
  };

  return (
    <div className="space-y-6">
      <div className="bg-blue-50 p-6 rounded-lg">
        <h2 className="text-xl font-semibold mb-4">Available Collateral</h2>
        {eligibleReceipts?.map(receipt => (
          <div key={receipt.id} className="border p-4 rounded flex justify-between items-
            <div>
              <h3>{receipt.commodity} - {receipt.quantity}</h3>
              <p>Value: ₹{receipt.valuation?.toLocaleString()}</p>
              <p>Available Loan: ₹{calculateLoanAmount(receipt.valuation)?.toLocaleString
            </div>
            <Button onClick={() => applyForLoan(receipt.id)}>Apply for Loan</Button>
          </div>
        ))}
```

```
      </div>
    </div>
  );
};
```

DATABASE CHANGES:

```
-- Add collateral availability to warehouse receipts
ALTER TABLE warehouse_receipts ADD COLUMN available_for_collateral BOOLEAN DEFAULT true;
ALTER TABLE warehouse_receipts ADD COLUMN collateral_used DECIMAL DEFAULT 0;

-- Update receipts when loans are created
CREATE OR REPLACE FUNCTION update_collateral_usage()
RETURNS trigger AS $$
BEGIN
  UPDATE warehouse_receipts
  SET collateral_used = collateral_used + NEW.amount
  WHERE id = NEW.collateral_receipt_id;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER loan_created_trigger
  AFTER INSERT ON loans
  FOR EACH ROW EXECUTE FUNCTION update_collateral_usage();
```

BACKEND API:

```
// Enhanced receipts endpoint
app.get('/api/receipts', async (req, res) => {
  try {
    const { available_for_collateral } = req.query;

    let query = db.select().from(warehouseReceipts)
      .where(eq(warehouseReceipts.ownerId, req.user.id));

    if (available_for_collateral === 'true') {
      query = query.where(
        and(
          eq(warehouseReceipts.status, 'active'),
          eq(warehouseReceipts.availableForCollateral, true),
          sql`collateral_used < valuation * 0.8`
        )
      );
    }

    const receipts = await query;
    res.json({ success: true, data: receipts });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});

// Loan application endpoint
```

```
app.post('/api/loans', async (req, res) => {
  try {
    const { collateralReceiptId, requestedAmount, duration } = req.body;

    // Validate collateral availability
    const receipt = await db.select().from(warehouseReceipts)
      .where(eq(warehouseReceipts.id, collateralReceiptId))
      .limit(1);

    if (!receipt[0]) {
      return res.status(400).json({ success: false, error: 'Receipt not found' });
    }

    const maxLoanAmount = receipt[0].valuation * 0.8 - (receipt[0].collateralUsed || 0);

    if (requestedAmount > maxLoanAmount) {
      return res.status(400).json({
        success: false,
        error: `Maximum loan amount available: ₹${maxLoanAmount}`
      });
    }

    const newLoan = await db.insert(loans).values({
      borrowerId: req.user.id,
      collateralReceiptId,
      amount: requestedAmount,
      interestRate: 12.0, // 12% per annum
      durationDays: duration,
      status: 'pending',
      blockchainHash: `LOAN-${Date.now()}-${Math.random().toString(36).substr(2, 9)}`
    }).returning();

    res.json({ success: true, data: newLoan[0] });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});
```

1.2 Deposit Workflow Automation (HIGH PRIORITY)

PROBLEM: Manual "Track Deposit" clicking, convoluted multi-step processes.

IMPLEMENTATION:

```
// Auto-start tracking on deposit creation
const createDeposit = async (depositData) => {
  const response = await fetch('/api/deposits', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    credentials: 'include',
    body: JSON.stringify(depositData)
  });

  const result = await response.json();
```

```
  if (result.success) {
    // AUTO-START tracking immediately
    await fetch(`/api/deposits/${result.data.id}/start-tracking`, {
      method: 'POST',
      credentials: 'include'
    });

    // Redirect to single tracking page
    window.location.href = `/track/${result.data.id}`;
  }

  return result;
};

// Single streamlined progress component
const DepositTracker = ({ depositId }) => {
  const { data: deposit, isLoading } = useQuery(['deposit-progress', depositId],
    () => fetch(`/api/deposits/${depositId}/progress`).then(r => r.json()),
    { refetchInterval: 10000 } // Auto-refresh every 10 seconds
  );

  const stages = [
    { key: 'created', name: 'Submitted', icon: '' },
    { key: 'pickup_scheduled', name: 'Pickup Scheduled', icon: '' },
    { key: 'in_transit', name: 'In Transit', icon: '' },
    { key: 'quality_check', name: 'Quality Assessment', icon: '' },
    { key: 'receipt_generated', name: 'Receipt Ready', icon: '' }
  ];

  const currentStageIndex = stages.findIndex(s => s.key === deposit?.currentStage);

  return (
    <div className="max-w-4xl mx-auto p-6">
      <div className="bg-white rounded-lg shadow p-6">
        <h2 className="text-2xl font-bold mb-6">Deposit Progress</h2>

        {/* Linear Progress Indicator */}
        <div className="mb-8">
          <div className="flex justify-between items-center">
            {stages.map((stage, index) => (
              <div key={stage.key} className="flex flex-col items-center">
                <div className={`w-12 h-12 rounded-full flex items-center justify-center
                  ${index <= currentStageIndex ? 'bg-green-100 text-green-600' : 'bg-gray
                  {stage.icon}
                </div>
                <span className={`text-sm text-center
                  ${index <= currentStageIndex ? 'text-green-600 font-medium' : 'text-gra
                  {stage.name}
                </span>
              </div>
            ))}
          </div>
          <div className="mt-4 h-2 bg-gray-200 rounded">
            <div
              className="h-2 bg-green-500 rounded transition-all duration-500"
              style={{ width: `${((currentStageIndex + 1) / stages.length) * 100}%` }}
```

```
            />
          </div>
        </div>

        {/* Current Status */}
        <div className="bg-blue-50 p-4 rounded-lg mb-6">
          <h3 className="font-semibold text-blue-900">Current Status</h3>
          <p className="text-blue-800">{deposit?.statusMessage}</p>
          {deposit?.estimatedCompletion && (
            <p className="text-blue-600 text-sm mt-1">
              Estimated completion: {new Date(deposit.estimatedCompletion).toLocaleString
            </p>
          )}
        </div>

        {/* Auto-refresh indicator */}
        <div className="flex items-center text-sm text-gray-500">
          <div className="animate-spin w-4 h-4 border-2 border-blue-500 border-t-transpar
          Updates automatically every 10 seconds
        </div>
      </div>
    </div>
  );
};
```

BACKEND AUTO-PROGRESSION:

```
// Auto-start tracking endpoint
app.post('/api/deposits/:id/start-tracking', async (req, res) => {
  try {
    const { id } = req.params;

    // Create process entry
    const process = await db.insert(processes).values({
      userId: req.user.id,
      commodityId: id,
      processType: 'deposit',
      status: 'processing',
      currentStage: 'pickup_scheduled',
      stageProgress: {
        startedAt: new Date(),
        estimatedCompletion: new Date(Date.now() + 4 * 60 * 60 * 1000) // 4 hours
      }
    }).returning();

    // Trigger automatic progression
    setTimeout(() => progressToNextStage(process[0].id), 15 * 60 * 1000); // 15 minutes

    res.json({ success: true, data: process[0] });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});

// Auto-progression function
```

```
const progressToNextStage = async (processId) => {
  const process = await db.select().from(processes).where(eq(processes.id, processId));

  if (!process[0]) return;

  const stageProgression = {
    'pickup_scheduled': 'in_transit',
    'in_transit': 'arrived_warehouse',
    'arrived_warehouse': 'quality_check',
    'quality_check': 'receipt_generated'
  };

  const nextStage = stageProgression[process[0].currentStage];

  if (nextStage) {
    await db.update(processes)
      .set({
        currentStage: nextStage,
        stageProgress: {
          ...process[0].stageProgress,
          [`${nextStage}_at`]: new Date()
        }
      })
      .where(eq(processes.id, processId));

    // Continue progression if not final stage
    if (nextStage !== 'receipt_generated') {
      const delay = getStageDelay(nextStage);
      setTimeout(() => progressToNextStage(processId), delay);
    } else {
      // Generate receipt when process complete
      await generateWarehouseReceipt(process[0].commodityId);
    }
  }
};

const getStageDelay = (stage) => {
  const delays = {
    'in_transit': 30 * 60 * 1000,        // 30 minutes
    'arrived_warehouse': 45 * 60 * 1000, // 45 minutes
    'quality_check': 90 * 60 * 1000,     // 90 minutes
    'receipt_generated': 15 * 60 * 1000  // 15 minutes
  };
  return delays[stage] || 30 * 60 * 1000;
};
```

PART 2: USER EXPERIENCE IMPROVEMENTS

2.1 Improved Location Selection

```
const LocationSelector = ({ onLocationSelect }) => {
  const [address, setAddress] = useState('');
  const [suggestions, setSuggestions] = useState([]);
  const [selectedLocation, setSelectedLocation] = useState(null);
  const [nearbyWarehouses, setNearbyWarehouses] = useState([]);

  const getCurrentLocation = () => {
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition((position) => {
        const { latitude, longitude } = position.coords;
        setSelectedLocation({ lat: latitude, lng: longitude });
        reverseGeocode(latitude, longitude);
        findNearbyWarehouses(latitude, longitude);
      });
    }
  };

  const handleAddressChange = async (value) => {
    setAddress(value);
    if (value.length > 3) {
      // Mock address suggestions - replace with real geocoding API
      const mockSuggestions = [
        `${value}, New Delhi, Delhi`,
        `${value}, Gurgaon, Haryana`,
        `${value}, Noida, Uttar Pradesh`
      ];
      setSuggestions(mockSuggestions.slice(0, 3));
    } else {
      setSuggestions([]);
    }
  };

  const selectAddress = async (selectedAddress) => {
    setAddress(selectedAddress);
    setSuggestions([]);

    // Mock geocoding - replace with real API
    const mockCoords = { lat: 28.6139, lng: 77.2090 };
    setSelectedLocation(mockCoords);
    findNearbyWarehouses(mockCoords.lat, mockCoords.lng);
    onLocationSelect(selectedAddress, mockCoords);
  };

  const findNearbyWarehouses = async (lat, lng) => {
    try {
      const response = await fetch(`/api/warehouses/nearby?lat=${lat}&lng=${lng}&radius=5
      const data = await response.json();
      setNearbyWarehouses(data.warehouses || []);
    } catch (error) {
      console.error('Failed to find nearby warehouses:', error);
    }
  };

  return (
```

```
    <div className="space-y-4">
      <div className="flex gap-2">
        <Button
          type="button"
          variant="outline"
          onClick={getCurrentLocation}
          className="flex items-center gap-2"
        >
          Use Current Location
        </Button>
      </div>

      <div className="relative">
        <Input
          placeholder="Enter your pickup address"
          value={address}
          onChange={(e) => handleAddressChange(e.target.value)}
          className="w-full"
        />

        {suggestions.length > 0 && (
          <div className="absolute z-10 w-full mt-1 bg-white border rounded-md shadow-lg"
            {suggestions.map((suggestion, index) => (
              <div
                key={index}
                className="p-3 hover:bg-gray-100 cursor-pointer"
                onClick={() => selectAddress(suggestion)}
              >
                {suggestion}
              </div>
            ))}
          </div>
        )}
      </div>

      {selectedLocation && nearbyWarehouses.length > 0 && (
        <div className="bg-green-50 p-4 rounded-lg">
          <h3 className="font-medium text-green-900 mb-2">Nearby Warehouses</h3>
          <div className="space-y-2">
            {nearbyWarehouses.map(warehouse => (
              <div key={warehouse.id} className="flex justify-between items-center p-2 bg
                <div>
                  <span className="font-medium">{warehouse.name}</span>
                  <p className="text-sm text-gray-600">{warehouse.distance}km away</p>
                </div>
                <span className="text-sm text-green-600">Available</span>
              </div>
            ))}
          </div>
        </div>
      )}
    </div>
  );
};
```

2.2 Streamlined Deposit Form

```
const StreamlinedDepositForm = () => {
  const [formData, setFormData] = useState({
    commodityName: '',
    commodityType: '',
    quantity: '',
    unit: 'MT',
    location: '',
    qualityParams: {}
  });
  const [estimatedCosts, setEstimatedCosts] = useState(null);
  const [isSubmitting, setIsSubmitting] = useState(false);

  // Auto-calculate costs as user types
  useEffect(() => {
    if (formData.commodityName && formData.quantity) {
      calculateEstimatedCosts(formData).then(setEstimatedCosts);
    }
  }, [formData]);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSubmitting(true);

    try {
      const result = await createDeposit(formData);
      if (result.success) {
        toast.success('Deposit submitted! Tracking started automatically.');
        // Auto-redirect to tracking page
        setTimeout(() => {
          window.location.href = `/track/${result.data.id}`;
        }, 1000);
      }
    } catch (error) {
      toast.error('Failed to submit deposit');
    } finally {
      setIsSubmitting(false);
    }
  };

  return (
    <div className="max-w-2xl mx-auto p-6">
      <div className="bg-white rounded-lg shadow p-6">
        <h2 className="text-2xl font-bold mb-6">Deposit Commodity</h2>

        <form onSubmit={handleSubmit} className="space-y-6">
          {/* Commodity Selection */}
          <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
            <div>
              <label className="block text-sm font-medium mb-2">Commodity Name</label>
              <CommoditySelector
                value={formData.commodityName}
                onChange={(value) => setFormData(prev => ({ ...prev, commodityName: value
              />
            </div>
```

```
            <div>
              <label className="block text-sm font-medium mb-2">Category</label>
              <Select
                value={formData.commodityType}
                onValueChange={(value) => setFormData(prev => ({ ...prev, commodityType:
              >
                <SelectTrigger>
                  <SelectValue placeholder="Select category" />
                </SelectTrigger>
                <SelectContent>
                  <SelectItem value="grains">Grains</SelectItem>
                  <SelectItem value="pulses">Pulses</SelectItem>
                  <SelectItem value="oilseeds">Oil Seeds</SelectItem>
                  <SelectItem value="spices">Spices</SelectItem>
                </SelectContent>
              </Select>
            </div>
          </div>

          {/* Quantity */}
          <div className="grid grid-cols-2 gap-4">
            <div>
              <label className="block text-sm font-medium mb-2">Quantity</label>
              <Input
                type="number"
                placeholder="Enter quantity"
                value={formData.quantity}
                onChange={(e) => setFormData(prev => ({ ...prev, quantity: e.target.value
                required
              />
            </div>
            <div>
              <label className="block text-sm font-medium mb-2">Unit</label>
              <Select
                value={formData.unit}
                onValueChange={(value) => setFormData(prev => ({ ...prev, unit: value }))
              >
                <SelectTrigger>
                  <SelectValue />
                </SelectTrigger>
                <SelectContent>
                  <SelectItem value="MT">Metric Tons (MT)</SelectItem>
                  <SelectItem value="quintal">Quintals</SelectItem>
                  <SelectItem value="kg">Kilograms</SelectItem>
                </SelectContent>
              </Select>
            </div>
          </div>

          {/* Location Selection */}
          <div>
            <label className="block text-sm font-medium mb-2">Pickup Location</label>
            <LocationSelector
              onLocationSelect={(address, coords) => {
                setFormData(prev => ({ ...prev, location: address, coordinates: coords })
              }}
```

```
            />
          </div>

          {/* Optional Quality Parameters */}
          <div className="bg-gray-50 p-4 rounded-lg">
            <h3 className="font-medium mb-3">Quality Parameters (Optional)</h3>
            <div className="grid grid-cols-3 gap-4">
              <div>
                <label className="block text-sm text-gray-600 mb-1">Moisture (%)</label>
                <Input
                  type="number"
                  placeholder="12"
                  onChange={(e) => setFormData(prev => ({
                    ...prev,
                    qualityParams: { ...prev.qualityParams, moisture: e.target.value }
                  }))}
                />
              </div>
              <div>
                <label className="block text-sm text-gray-600 mb-1">Foreign Matter (%)</l
                <Input
                  type="number"
                  placeholder="2"
                  onChange={(e) => setFormData(prev => ({
                    ...prev,
                    qualityParams: { ...prev.qualityParams, foreignMatter: e.target.value
                  }))}
                />
              </div>
              <div>
                <label className="block text-sm text-gray-600 mb-1">Broken Grains (%)</la
                <Input
                  type="number"
                  placeholder="5"
                  onChange={(e) => setFormData(prev => ({
                    ...prev,
                    qualityParams: { ...prev.qualityParams, brokenGrains: e.target.value
                  }))}
                />
              </div>
            </div>
            <p className="text-sm text-gray-500 mt-2">
              Quality assessment will be done at warehouse. These values help with initia
            </p>
          </div>

          {/* Estimated Costs */}
          {estimatedCosts && (
            <div className="bg-blue-50 p-4 rounded-lg">
              <h3 className="font-medium text-blue-900 mb-3">Estimated Costs & Returns</h
              <div className="grid grid-cols-2 gap-4 text-sm">
                <div>
                  <span className="text-blue-700">Storage Cost:</span>
                  <span className="float-right font-medium">₹{estimatedCosts.storageCost?
                </div>
                <div>
```

```
                  <span className="text-blue-700">Quality Testing:</span>
                  <span className="float-right font-medium">₹{estimatedCosts.testingCost?
                </div>
                <div>
                  <span className="text-blue-700">Est. Market Value:</span>
                  <span className="float-right font-medium">₹{estimatedCosts.marketValue?
                </div>
                <div className="border-t pt-2">
                  <span className="text-blue-700 font-medium">Available for Loan:</span>
                  <span className="float-right font-medium">₹{estimatedCosts.loanAmount?.
                </div>
              </div>
            </div>
          )}

          {/* Submit Button */}
          <Button
            type="submit"
            size="lg"
            className="w-full"
            disabled={isSubmitting}
          >
            {isSubmitting ? (
              <div className="flex items-center gap-2">
                <div className="animate-spin w-4 h-4 border-2 border-white border-t-trans
                Processing Deposit...
              </div>
            ) : (
              'Submit Deposit & Start Tracking'
            )}
          </Button>
        </form>
      </div>
    </div>
  );
};
```

2.3 Enhanced Dashboard with Quick Actions

```
const EnhancedDashboard = () => {
  const { data: portfolioData } = useQuery(['portfolio'], fetchPortfolioData);
  const { data: eligibleReceipts } = useQuery(['eligible-receipts'], fetchEligibleReceipt
  const { data: activeProcesses } = useQuery(['active-processes'], fetchActiveProcesses);
  const { data: recentActivity } = useQuery(['recent-activity'], fetchRecentActivity);

  return (
    <div className="max-w-7xl mx-auto p-6 space-y-6">
      {/* Portfolio Overview */}
      <div className="grid grid-cols-1 md:grid-cols-4 gap-6">
        <div className="bg-gradient-to-br from-blue-500 to-blue-600 text-white p-6 rounde
          <h3 className="text-sm font-medium opacity-90">Total Portfolio Value</h3>
          <p className="text-3xl font-bold mt-2">₹{portfolioData?.totalValue?.toLocaleStr
          <p className="text-sm opacity-75 mt-1">Across {portfolioData?.receiptsCount} re
        </div>
```

```
        <div className="bg-gradient-to-br from-green-500 to-green-600 text-white p-6 roun
          <h3 className="text-sm font-medium opacity-90">Available Credit</h3>
          <p className="text-3xl font-bold mt-2">₹{portfolioData?.availableCredit?.toLoca
          <p className="text-sm opacity-75 mt-1">80% of portfolio value</p>
        </div>

        <div className="bg-gradient-to-br from-orange-500 to-orange-600 text-white p-6 ro
          <h3 className="text-sm font-medium opacity-90">Active Processes</h3>
          <p className="text-3xl font-bold mt-2">{activeProcesses?.length || 0}</p>
          <p className="text-sm opacity-75 mt-1">Currently in progress</p>
        </div>

        <div className="bg-gradient-to-br from-purple-500 to-purple-600 text-white p-6 ro
          <h3 className="text-sm font-medium opacity-90">Ready for Loans</h3>
          <p className="text-3xl font-bold mt-2">{eligibleReceipts?.length || 0}</p>
          <p className="text-sm opacity-75 mt-1">Warehouse receipts</p>
        </div>
      </div>

      {/* Quick Actions */}
      <div className="bg-white rounded-xl shadow p-6">
        <h2 className="text-xl font-semibold mb-4">Quick Actions</h2>
        <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
          <QuickActionCard
            title="Apply for Loans"
            description={`${eligibleReceipts?.length || 0} receipts ready for collateral
            action="Apply Now"
            actionColor="bg-green-600 hover:bg-green-700"
            onClick={() => window.location.href = '/loans'}
            disabled={!eligibleReceipts?.length}
          />

          <QuickActionCard
            title="Track Deposits"
            description={`${activeProcesses?.length || 0} deposits in progress`}
            action="View Progress"
            actionColor="bg-blue-600 hover:bg-blue-700"
            onClick={() => window.location.href = '/processes'}
            disabled={!activeProcesses?.length}
          />

          <QuickActionCard
            title="New Deposit"
            description="Deposit commodities in verified warehouses"
            action="Start Deposit"
            actionColor="bg-purple-600 hover:bg-purple-700"
            onClick={() => window.location.href = '/deposit'}
          />
        </div>
      </div>

      {/* Recent Activity */}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <div className="bg-white rounded-xl shadow p-6">
          <h2 className="text-xl font-semibold mb-4">Recent Warehouse Receipts</h2>
          {portfolioData?.recentReceipts?.length > 0 ? (
```

```
            <div className="space-y-3">
              {portfolioData.recentReceipts.map(receipt => (
                <div key={receipt.id} className="flex justify-between items-center p-3 bg
                  <div>
                    <h3 className="font-medium">{receipt.receiptNumber}</h3>
                    <p className="text-sm text-gray-600">{receipt.commodity} - {receipt.c
                  </div>
                  <div className="text-right">
                    <p className="font-medium">₹{receipt.valuation?.toLocaleString()}</p>
                    <span className="text-xs px-2 py-1 bg-green-100 text-green-700 rounde
                      {receipt.status}
                    </span>
                  </div>
                </div>
              ))}
            </div>
          ) : (
            <div className="text-center py-8 text-gray-500">
              <p>No warehouse receipts yet</p>
              <Button className="mt-4" onClick={() => window.location.href = '/deposit'}>
                Create First Deposit
              </Button>
            </div>
          )}
        </div>

        <div className="bg-white rounded-xl shadow p-6">
          <h2 className="text-xl font-semibold mb-4">Active Processes</h2>
          {activeProcesses?.length > 0 ? (
            <div className="space-y-3">
              {activeProcesses.map(process => (
                <div key={process.id} className="p-3 border border-gray-200 rounded-lg">
                  <div className="flex justify-between items-start mb-2">
                    <h3 className="font-medium">{process.commodity}</h3>
                    <span className="text-xs px-2 py-1 bg-blue-100 text-blue-700 rounded-
                      {process.currentStage?.replace('_', ' ')}
                    </span>
                  </div>
                  <div className="w-full bg-gray-200 rounded-full h-2">
                    <div
                      className="bg-blue-500 h-2 rounded-full transition-all duration-300
                      style={{ width: `${process.progress || 0}%` }}
                    />
                  </div>
                  <div className="flex justify-between text-xs text-gray-500 mt-2">
                    <span>Started {new Date(process.createdAt).toLocaleDateString()}</spa
                    <Button
                      size="sm"
                      variant="outline"
                      onClick={() => window.location.href = `/track/${process.id}`}
                    >
                      Track
                    </Button>
                  </div>
                </div>
              ))}
```

```
            </div>
          ) : (
            <div className="text-center py-8 text-gray-500">
              <p>No active processes</p>
            </div>
          )}
        </div>
      </div>
    </div>
  );
};

const QuickActionCard = ({ title, description, action, actionColor, onClick, disabled }) =>
  <div className="border border-gray-200 rounded-lg p-4">
    <h3 className="font-medium mb-2">{title}</h3>
    <p className="text-sm text-gray-600 mb-4">{description}</p>
    <Button
      onClick={onClick}
      disabled={disabled}
      className={`w-full ${actionColor}`}
      variant={disabled ? "secondary" : "default"}
    >
      {action}
    </Button>
  </div>
);
```

PART 3: INTEGRATION & WEBHOOK SYSTEM

3.1 Complete Webhook Implementation

```
// Enhanced webhook system with comprehensive error handling
const webhookRoutes = (app) => {
  // Webhook authentication middleware
  const authenticateWebhook = (req, res, next) => {
    const apiKey = req.headers['x-api-key'];
    const validKeys = {
      'warehouse-key-123': 'warehouse',
      'quality-key-456': 'quality',
      'pricing-key-789': 'pricing'
    };

    if (!validKeys[apiKey]) {
      return res.status(401).json({ success: false, error: 'Invalid API key' });
    }

    req.moduleType = validKeys[apiKey];
    next();
  };

  // Warehouse status updates
  app.post('/api/webhooks/warehouse/status-update', authenticateWebhook, async (req, res)
    try {
```

```
    const { depositId, currentStep, status, estimatedCompletion, metadata } = req.body;

    // Update process status
    await db.update(processes)
      .set({
        currentStage: currentStep,
        status: status,
        stageProgress: {
          ...metadata,
          estimatedCompletion: estimatedCompletion,
          updatedAt: new Date()
        }
      })
      .where(eq(processes.commodityId, depositId));

    // Trigger real-time updates (SSE or WebSocket)
    broadcastUpdate({
      type: 'process-update',
      depositId,
      currentStep,
      status,
      estimatedCompletion
    });

    res.json({ success: true, message: 'Status updated successfully' });
  } catch (error) {
    console.error('Webhook error:', error);
    res.status(500).json({ success: false, error: error.message });
  }
});

// IoT weight and quality data
app.post('/api/webhooks/warehouse/weight-update', authenticateWebhook, async (req, res)
  try {
    const { depositId, weightData, qualityData } = req.body;

    // Update commodity with actual measurements
    await db.update(commodities)
      .set({
        quantity: weightData.netWeight,
        qualityGrade: qualityData.overallGrade,
        metadata: {
          ...qualityData,
          weightData,
          lastUpdated: new Date()
        }
      })
      .where(eq(commodities.id, depositId));

    // Auto-generate receipt if quality meets standards
    if (qualityData.overallGrade && ['Premium', 'A', 'Good'].includes(qualityData.overa
      await generateWarehouseReceipt(depositId, weightData, qualityData);
    }

    res.json({ success: true, message: 'Weight and quality data updated' });
  } catch (error) {
```

```
      console.error('Weight update webhook error:', error);
      res.status(500).json({ success: false, error: error.message });
    }
  });

  // Quality testing results
  app.post('/api/webhooks/quality/results', authenticateWebhook, async (req, res) => {
    try {
      const { depositId, testResults, images } = req.body;
      const { moistureContent, foreignMatter, brokenGrains, overallGrade, aiConfidence }

      // Save quality assessment
      await db.insert(sackQualityAssessments).values({
        sackId: depositId,
        moistureContent: parseFloat(moistureContent),
        foreignMatter: parseFloat(foreignMatter),
        brokenGrains: parseFloat(brokenGrains),
        overallGrade,
        assessmentMethod: 'AI_ANALYSIS',
        assessmentData: {
          aiConfidence: parseFloat(aiConfidence),
          testImages: images,
          testDate: new Date()
        }
      });

      // Update commodity quality and recalculate valuation
      const qualityMultiplier = getQualityMultiplier(overallGrade);
      const basePrice = await getCurrentMarketPrice(depositId);
      const newValuation = basePrice * qualityMultiplier;

      await db.update(commodities)
        .set({
          qualityGrade: overallGrade,
          marketValue: newValuation,
          lastUpdated: new Date()
        })
        .where(eq(commodities.id, depositId));

      // Auto-generate receipt for high-quality commodities
      if (aiConfidence >= 80 && ['Premium', 'A'].includes(overallGrade)) {
        await generateQualityCertifiedReceipt(depositId, testResults);
      }

      res.json({ success: true, message: 'Quality results processed' });
    } catch (error) {
      console.error('Quality results webhook error:', error);
      res.status(500).json({ success: false, error: error.message });
    }
  });
};

// Auto-receipt generation
const generateWarehouseReceipt = async (commodityId, weightData, qualityData) => {
  try {
    const commodity = await db.select().from(commodities)
```

```
      .where(eq(commodities.id, commodityId))
      .limit(1);

    if (!commodity[0]) throw new Error('Commodity not found');

    const qualityMultiplier = getQualityMultiplier(qualityData.overallGrade);
    const basePrice = await getCurrentMarketPrice(commodity[0].name);
    const finalValuation = basePrice * qualityMultiplier * weightData.netWeight;

    const receipt = await db.insert(warehouseReceipts).values({
      receiptNumber: `WR${Date.now()}-${Math.random().toString(36).substr(2, 6)}`,
      commodityId,
      ownerId: commodity[0].ownerId,
      warehouseId: commodity[0].warehouseId,
      quantity: weightData.netWeight,
      valuation: finalValuation,
      status: 'active',
      blockchainHash: `BC-${Date.now()}-${Math.random().toString(36).substr(2, 9)}`,
      availableForCollateral: true,
      qualityGrade: qualityData.overallGrade,
      metadata: {
        weightData,
        qualityData,
        generatedAt: new Date(),
        autoGenerated: true
      }
    }).returning();

    // Update process to completed
    await db.update(processes)
      .set({
        status: 'completed',
        currentStage: 'receipt_generated',
        completedAt: new Date()
      })
      .where(eq(processes.commodityId, commodityId));

    // Broadcast receipt generation
    broadcastUpdate({
      type: 'receipt-generated',
      receiptId: receipt[0].id,
      commodityId,
      valuation: finalValuation
    });

    return receipt[0];
  } catch (error) {
    console.error('Receipt generation error:', error);
    throw error;
  }
};

const getQualityMultiplier = (grade) => {
  const multipliers = {
    'Premium': 1.3,
    'A': 1.1,
```

```
    'Good': 1.0,
    'Fair': 0.9,
    'Poor': 0.8
  };
  return multipliers[grade] || 1.0;
};
```

3.2 Real-Time Updates with Server-Sent Events

```
// SSE implementation for real-time updates
app.get('/api/events/stream', (req, res) => {
  const userId = req.user?.id;
  if (!userId) {
    return res.status(401).json({ error: 'Unauthorized' });
  }

  res.writeHead(200, {
    'Content-Type': 'text/event-stream',
    'Cache-Control': 'no-cache',
    'Connection': 'keep-alive',
    'Access-Control-Allow-Origin': '*',
    'Access-Control-Allow-Headers': 'Cache-Control'
  });

  // Send initial connection confirmation
  res.write(`data: ${JSON.stringify({ type: 'connected', userId })}\n\n`);

  // Store connection for user-specific updates
  const connectionId = `${userId}-${Date.now()}`;
  userConnections.set(connectionId, { userId, res, lastPing: Date.now() });

  // Send periodic heartbeat
  const heartbeat = setInterval(() => {
    res.write(`data: ${JSON.stringify({ type: 'heartbeat', timestamp: Date.now() })}\n\n`
  }, 30000);

  // Cleanup on disconnect
  req.on('close', () => {
    clearInterval(heartbeat);
    userConnections.delete(connectionId);
  });
});

// Global broadcast function
const broadcastUpdate = (updateData) => {
  for (const [connectionId, connection] of userConnections.entries()) {
    try {
      if (shouldSendToUser(connection.userId, updateData)) {
        connection.res.write(`data: ${JSON.stringify(updateData)}\n\n`);
      }
    } catch (error) {
      console.error('SSE broadcast error:', error);
      userConnections.delete(connectionId);
    }
  }
```

```
};

// Frontend SSE hook
const useRealTimeUpdates = () => {
  const queryClient = useQueryClient();

  useEffect(() => {
    const eventSource = new EventSource('/api/events/stream', {
      withCredentials: true
    });

    eventSource.onmessage = (event) => {
      const data = JSON.parse(event.data);

      switch (data.type) {
        case 'process-update':
          queryClient.invalidateQueries(['deposit-progress', data.depositId]);
          queryClient.invalidateQueries(['active-processes']);
          break;
        case 'receipt-generated':
          queryClient.invalidateQueries(['receipts']);
          queryClient.invalidateQueries(['portfolio']);
          queryClient.invalidateQueries(['eligible-receipts']);
          toast.success('New warehouse receipt generated!');
          break;
        case 'portfolio-update':
          queryClient.invalidateQueries(['portfolio']);
          break;
      }
    };

    eventSource.onerror = (error) => {
      console.error('SSE connection error:', error);
    };

    return () => {
      eventSource.close();
    };
  }, [queryClient]);
};
```

PART 4: ENTERPRISE MONITORING & FEATURES

4.1 Integration Health Dashboard

```
const IntegrationHealthDashboard = () => {
  const [healthData, setHealthData] = useState(null);
  const [logs, setLogs] = useState([]);

  useEffect(() => {
    const fetchHealthData = async () => {
      const response = await fetch('/api/admin/integration-health');
      const data = await response.json();
```

```
      setHealthData(data);
    };

    const fetchLogs = async () => {
      const response = await fetch('/api/admin/integration-logs?limit=50');
      const data = await response.json();
      setLogs(data.logs);
    };

    fetchHealthData();
    fetchLogs();

    const interval = setInterval(() => {
      fetchHealthData();
      fetchLogs();
    }, 10000);

    return () => clearInterval(interval);
  }, []);

  return (
    <div className="max-w-7xl mx-auto p-6 space-y-6">
      <h1 className="text-3xl font-bold">Integration Health Dashboard</h1>

      {/* Health Overview */}
      <div className="grid grid-cols-1 md:grid-cols-4 gap-6">
        <div className="bg-white rounded-lg shadow p-6">
          <h3 className="text-sm font-medium text-gray-500">Webhook Success Rate</h3>
          <p className="text-2xl font-bold text-green-600">
            {healthData?.webhookSuccessRate || 0}%
          </p>
          <p className="text-sm text-gray-600">Last 24 hours</p>
        </div>

        <div className="bg-white rounded-lg shadow p-6">
          <h3 className="text-sm font-medium text-gray-500">Avg Response Time</h3>
          <p className="text-2xl font-bold text-blue-600">
            {healthData?.avgResponseTime || 0}ms
          </p>
          <p className="text-sm text-gray-600">API calls</p>
        </div>

        <div className="bg-white rounded-lg shadow p-6">
          <h3 className="text-sm font-medium text-gray-500">Active Connections</h3>
          <p className="text-2xl font-bold text-purple-600">
            {healthData?.activeConnections || 0}
          </p>
          <p className="text-sm text-gray-600">Real-time updates</p>
        </div>

        <div className="bg-white rounded-lg shadow p-6">
          <h3 className="text-sm font-medium text-gray-500">Failed Requests</h3>
          <p className="text-2xl font-bold text-red-600">
            {healthData?.failedRequests || 0}
          </p>
          <p className="text-sm text-gray-600">Last hour</p>
```

```
        </div>
      </div>

      {/* Module Status */}
      <div className="bg-white rounded-lg shadow p-6">
        <h2 className="text-xl font-semibold mb-4">Module Connectivity</h2>
        <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
          {healthData?.moduleStatus?.map(module => (
            <div key={module.name} className="border rounded-lg p-4">
              <div className="flex justify-between items-center mb-2">
                <h3 className="font-medium">{module.name}</h3>
                <span className={`px-2 py-1 rounded-full text-xs ${
                  module.status === 'online'
                    ? 'bg-green-100 text-green-700'
                    : 'bg-red-100 text-red-700'
                }`}>
                  {module.status}
                </span>
              </div>
              <p className="text-sm text-gray-600">
                Last check: {new Date(module.lastCheck).toLocaleString()}
              </p>
              <p className="text-sm text-gray-600">
                Response time: {module.responseTime}ms
              </p>
            </div>
          ))}
        </div>
      </div>

      {/* Recent Logs */}
      <div className="bg-white rounded-lg shadow p-6">
        <h2 className="text-xl font-semibold mb-4">Recent Integration Logs</h2>
        <div className="overflow-x-auto">
          <table className="w-full text-sm">
            <thead>
              <tr className="border-b">
                <th className="text-left p-2">Timestamp</th>
                <th className="text-left p-2">Type</th>
                <th className="text-left p-2">Module</th>
                <th className="text-left p-2">Status</th>
                <th className="text-left p-2">Message</th>
              </tr>
            </thead>
            <tbody>
              {logs.map((log, index) => (
                <tr key={index} className="border-b">
                  <td className="p-2 text-gray-600">
                    {new Date(log.timestamp).toLocaleString()}
                  </td>
                  <td className="p-2">
                    <span className={`px-2 py-1 rounded text-xs ${
                      log.type === 'webhook' ? 'bg-blue-100 text-blue-700' :
                      log.type === 'outbound' ? 'bg-purple-100 text-purple-700' :
                      'bg-gray-100 text-gray-700'
                    }`}>
```

```
                    {log.type}
                  </span>
                </td>
                <td className="p-2">{log.module}</td>
                <td className="p-2">
                  <span className={`px-2 py-1 rounded text-xs ${
                    log.status === 'success' ? 'bg-green-100 text-green-700' :
                    'bg-red-100 text-red-700'
                  }`}>
                    {log.status}
                  </span>
                </td>
                <td className="p-2 text-gray-600">{log.message}</td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  );
};
```

4.2 API Documentation Generator

```
// Auto-generated API documentation
app.get('/api/docs', (req, res) => {
  const apiSpec = {
    openapi: '3.0.0',
    info: {
      title: 'TradeWiser Integration API',
      version: '1.0.0',
      description: 'API specification for TradeWiser warehouse and quality module integra
    },
    servers: [
      { url: 'https://tradewiser.com/api', description: 'Production server' },
      { url: 'http://localhost:5000/api', description: 'Development server' }
    ],
    components: {
      securitySchemes: {
        ApiKeyAuth: {
          type: 'apiKey',
          in: 'header',
          name: 'X-API-Key'
        }
      }
    },
    paths: {
      '/webhooks/warehouse/status-update': {
        post: {
          summary: 'Warehouse Status Update Webhook',
          description: 'Receive real-time status updates from warehouse management module
          security: [{ ApiKeyAuth: [] }],
          requestBody: {
            required: true,
```

```
            content: {
              'application/json': {
                schema: {
                  type: 'object',
                  required: ['depositId', 'currentStep', 'status'],
                  properties: {
                    depositId: { type: 'string', description: 'Deposit ID' },
                    currentStep: {
                      type: 'string',
                      enum: ['pickup_scheduled', 'in_transit', 'arrived_warehouse', 'qual
                      description: 'Current processing stage'
                    },
                    status: {
                      type: 'string',
                      enum: ['processing', 'completed', 'failed'],
                      description: 'Process status'
                    },
                    estimatedCompletion: { type: 'string', format: 'date-time' },
                    metadata: { type: 'object', description: 'Additional stage-specific d
                  }
                }
              }
            }
          },
          responses: {
            '200': {
              description: 'Status updated successfully',
              content: {
                'application/json': {
                  schema: {
                    type: 'object',
                    properties: {
                      success: { type: 'boolean' },
                      message: { type: 'string' }
                    }
                  }
                }
              }
            },
            '401': { description: 'Invalid API key' },
            '500': { description: 'Internal server error' }
          }
        }
      }
      // ... more endpoints
    }
  };

  res.json(apiSpec);
});

// Swagger UI for interactive testing
app.use('/api/docs/ui', swaggerUi.serve, swaggerUi.setup(null, {
  swaggerUrl: '/api/docs'
}));
```

PART 5: PRODUCTION DEPLOYMENT & CONFIGURATION

5.1 Environment Configuration

```
// Enhanced environment configuration
const config = {
  // Database
  DATABASE_URL: process.env.DATABASE_URL || 'postgresql://user:pass@localhost:5432/tradew

  // Session
  SESSION_SECRET: process.env.SESSION_SECRET || 'fallback-secret-key',

  // Integration URLs
  WAREHOUSE_MODULE_URL: process.env.WAREHOUSE_MODULE_URL || 'http://localhost:3001',
  QUALITY_MODULE_URL: process.env.QUALITY_MODULE_URL || 'http://localhost:3002',
  PRICING_MODULE_URL: process.env.PRICING_MODULE_URL || 'http://localhost:3003',

  // API Keys
  WEBHOOK_API_KEYS: (process.env.WEBHOOK_API_KEYS || 'warehouse-key-123,quality-key-456,p
  WEBHOOK_SECRET: process.env.WEBHOOK_SECRET || 'tradewiser-webhook-secret-2024',

  // Feature Flags
  USE_MOCK_INTEGRATIONS: process.env.USE_MOCK_INTEGRATIONS === 'true',
  ENABLE_SSE: process.env.ENABLE_SSE !== 'false',
  ENABLE_WEBHOOKS: process.env.ENABLE_WEBHOOKS !== 'false',

  // Performance
  WEBHOOK_TIMEOUT: parseInt(process.env.WEBHOOK_TIMEOUT) || 10000,
  MAX_RETRY_ATTEMPTS: parseInt(process.env.MAX_RETRY_ATTEMPTS) || 3,
  RATE_LIMIT: parseInt(process.env.RATE_LIMIT) || 100,

  // Monitoring
  LOG_LEVEL: process.env.LOG_LEVEL || 'info',
  ENABLE_METRICS: process.env.ENABLE_METRICS !== 'false'
};

// Validation
const requiredEnvVars = [
  'DATABASE_URL',
  'SESSION_SECRET'
];

for (const envVar of requiredEnvVars) {
  if (!process.env[envVar]) {
    console.error(`Missing required environment variable: ${envVar}`);
    process.exit(1);
  }
}

module.exports = config;
```

5.2 Testing Framework

```
// Comprehensive testing utilities
const TestingUtils = {
  // Trigger mock warehouse process
  async simulateWarehouseProcess(depositId) {
    const stages = ['pickup_scheduled', 'in_transit', 'arrived_warehouse', 'quality_check

    for (let i = 0; i < stages.length; i++) {
      await new Promise(resolve => setTimeout(resolve, 5000)); // 5 second delay

      await fetch('/api/webhooks/warehouse/status-update', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'X-API-Key': 'warehouse-key-123'
        },
        body: JSON.stringify({
          depositId,
          currentStep: stages[i],
          status: 'processing',
          estimatedCompletion: new Date(Date.now() + (stages.length - i - 1) * 300000), /
          metadata: {
            stage: i + 1,
            totalStages: stages.length,
            message: `Processing stage: ${stages[i]}`
          }
        })
      });
    }
  },

  // Simulate quality testing
  async simulateQualityTest(depositId) {
    await new Promise(resolve => setTimeout(resolve, 10000)); // 10 second delay

    const mockResults = {
      moistureContent: Math.random() * 5 + 10, // 10-15%
      foreignMatter: Math.random() * 2, // 0-2%
      brokenGrains: Math.random() * 5, // 0-5%
      overallGrade: ['Premium', 'A', 'Good'][Math.floor(Math.random() * 3)],
      aiConfidence: Math.random() * 20 + 80 // 80-100%
    };

    await fetch('/api/webhooks/quality/results', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'X-API-Key': 'quality-key-456'
      },
      body: JSON.stringify({
        depositId,
        testResults: mockResults,
        images: ['mock-image-1.jpg', 'mock-image-2.jpg']
      })
    });
```

```
  }
};

// Testing dashboard component
const TestingDashboard = () => {
  const [testLogs, setTestLogs] = useState([]);

  const runFullWorkflowTest = async () => {
    const log = (message) => {
      setTestLogs(prev => [...prev, { message, timestamp: new Date().toISOString() }]);
    };

    try {
      log('Starting full workflow test...');

      // Create test deposit
      const depositResponse = await fetch('/api/deposits', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        credentials: 'include',
        body: JSON.stringify({
          commodityName: 'Test Wheat',
          commodityType: 'grains',
          quantity: 50,
          unit: 'MT',
          location: 'Test Location'
        })
      });

      const deposit = await depositResponse.json();
      log(`Created test deposit: ${deposit.data.id}`);

      // Simulate warehouse processing
      log('Starting warehouse simulation...');
      await TestingUtils.simulateWarehouseProcess(deposit.data.id);
      log('Warehouse simulation completed');

      // Simulate quality testing
      log('Starting quality testing simulation...');
      await TestingUtils.simulateQualityTest(deposit.data.id);
      log('Quality testing simulation completed');

      log('Full workflow test completed successfully!');

    } catch (error) {
      log(`Test failed: ${error.message}`);
    }
  };

  return (
    <div className="max-w-4xl mx-auto p-6">
      <h1 className="text-3xl font-bold mb-6">Integration Testing Dashboard</h1>

      <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-6">
        <Button onClick={runFullWorkflowTest} className="bg-blue-600 hover:bg-blue-700">
          Run Full Workflow Test
```

```
        </Button>
        <Button onClick={() => TestingUtils.simulateQualityTest('test-123')} variant="out
          Test Quality Assessment
        </Button>
        <Button onClick={() => window.location.reload()} variant="outline">
          Clear Logs
        </Button>
      </div>

      <div className="bg-black text-green-400 p-4 rounded-lg h-96 overflow-y-auto font-mo
        <div className="mb-2">TradeWiser Integration Testing Console</div>
        <div className="mb-4">{'='* 50}</div>
        {testLogs.map((log, index) => (
          <div key={index} className="mb-1">
            [{new Date(log.timestamp).toLocaleTimeString()}] {log.message}
          </div>
        ))}
        {testLogs.length === 0 && (
          <div className="text-gray-500">Ready for testing... Click a button above to sta
        )}
      </div>
    </div>
  );
};
```

IMPLEMENTATION SUCCESS CRITERIA

✅ Business Logic Fixes

- Warehouse receipts automatically appear in loans section with calculated loan amounts
- Receipt-to-loan connection works seamlessly
- Automatic deposit tracking without manual intervention
- Streamlined single-page forms with smart defaults

✅ User Experience Improvements

- Simplified location selection with auto-detect
- Single progress view instead of 9 separate steps
- Enhanced dashboard with actionable quick actions
- Real-time updates via Server-Sent Events
- Professional, consistent UI throughout

✅ Integration System

- Complete webhook system for external module communication
- Comprehensive API documentation with Swagger UI
- Mock integration system for testing

- Enterprise-grade monitoring and health dashboard

✅ Production Features

- Advanced error handling and retry mechanisms
- Security enhancements with API key authentication
- Performance optimizations and caching
- Comprehensive testing framework and utilities

IMPLEMENT ALL FIXES AND FEATURES IN THIS MASTER PROMPT TO CREATE A PRODUCTION-READY TRADEWISER PLATFORM.