TRADEWISER - LIVE DEPOSIT TRACKING FIX

Complete Real-Time Progress Monitoring System

ISSUE IDENTIFIED

The deposit tracking workflow stops after receipt generation and doesn't provide real-time progress updates for client demonstrations. The auto-progression functions aren't working properly.

SOLUTION OVERVIEW

Create a comprehensive real-time tracking system with:

1. **Visual Progress Tracker** - Animated progress bar with live updates
2. **Auto-Progression System** - Realistic timing for each workflow stage
3. **WebSocket Integration** - Real-time updates without page refresh
4. **Client Demo Ready** - Professional presentation interface

COMPLETE TRACKING SYSTEM IMPLEMENTATION

1. REAL-TIME TRACKING PAGE COMPONENT

Create file: `client/src/pages/TrackDepositPage.tsx`

```
import React, { useState, useEffect } from 'react';
import { useParams } from 'wouter';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
import { Badge } from '@/components/ui/badge';
import { Button } from '@/components/ui/button';
import { useToast } from '@/hooks/use-toast';
import {
  Truck,
  Warehouse,
  Scale,
  FlaskConical,
  Calculator,
  FileCheck,
  Clock,
  CheckCircle,
  Loader2,
  MapPin,
  User,
  Phone,
```

```
  AlertCircle
} from 'lucide-react';

interface DepositProgress {
  id: number;
  currentStage: string;
  statusMessage: string;
  estimatedCompletion: string;
  progress: {
    pickup_scheduled: 'pending' | 'current' | 'completed';
    in_transit: 'pending' | 'current' | 'completed';
    arrived_warehouse: 'pending' | 'current' | 'completed';
    quality_check: 'pending' | 'current' | 'completed';
    pricing_calculated: 'pending' | 'current' | 'completed';
    receipt_generated: 'pending' | 'current' | 'completed';
  };
  commodity?: {
    name: string;
    quantity: string;
    unit: string;
    estimatedValue: number;
  };
  warehouse?: {
    name: string;
    address: string;
    contact: string;
  };
  vehicle?: {
    number: string;
    driver: string;
    phone: string;
  };
}

const TrackDepositPage = () => {
  const { depositId } = useParams();
  const [progress, setProgress] = useState<DepositProgress | null>(null);
  const [loading, setLoading] = useState(true);
  const [lastUpdated, setLastUpdated] = useState(new Date());
  const [autoRefresh, setAutoRefresh] = useState(true);
  const { toast } = useToast();

  const stages = [
    {
      key: 'pickup_scheduled',
      title: 'Pickup Scheduled',
      icon: Clock,
      description: 'Vehicle assigned and pickup scheduled',
      color: 'blue'
    },
    {
      key: 'in_transit',
      title: 'In Transit',
      icon: Truck,
      description: 'Commodity being transported to warehouse',
      color: 'orange'
```

```
    },
    {
      key: 'arrived_warehouse',
      title: 'Arrived at Warehouse',
      icon: Warehouse,
      description: 'Commodity received at warehouse facility',
      color: 'purple'
    },
    {
      key: 'quality_check',
      title: 'Quality Assessment',
      icon: FlaskConical,
      description: 'Quality testing and grading in progress',
      color: 'green'
    },
    {
      key: 'pricing_calculated',
      title: 'Pricing Complete',
      icon: Calculator,
      description: 'Market pricing and valuation calculated',
      color: 'indigo'
    },
    {
      key: 'receipt_generated',
      title: 'Receipt Generated',
      icon: FileCheck,
      description: 'Electronic warehouse receipt created',
      color: 'emerald'
    }
  ];

  useEffect(() => {
    if (depositId) {
      fetchProgress();

      // Auto-refresh every 10 seconds during active processing
      const interval = setInterval(() => {
        if (autoRefresh && progress?.currentStage !== 'receipt_generated') {
          fetchProgress();
        }
      }, 10000);

      return () => clearInterval(interval);
    }
  }, [depositId, autoRefresh, progress?.currentStage]);

  const fetchProgress = async () => {
    try {
      const response = await fetch(`/api/deposits/${depositId}/progress`, {
        credentials: 'include'
      });

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

```
      const data = await response.json();
      setProgress(data);
      setLastUpdated(new Date());

      // Show completion notification
      if (data.currentStage === 'receipt_generated' && progress?.currentStage !== 'receip
        toast({
          title: "🎉 Deposit Complete!",
          description: "Your electronic warehouse receipt has been generated",
          duration: 5000
        });
      }
    } catch (error) {
      console.error('Error fetching progress:', error);
      toast({
        title: "Connection Error",
        description: "Unable to fetch live updates",
        variant: "destructive"
      });
    } finally {
      setLoading(false);
    }
  };

  const startTracking = async () => {
    try {
      const response = await fetch(`/api/deposits/${depositId}/start-tracking`, {
        method: 'POST',
        credentials: 'include'
      });

      if (response.ok) {
        toast({
          title: "Tracking Started",
          description: "Real-time progress monitoring activated"
        });
        fetchProgress();
      }
    } catch (error) {
      console.error('Error starting tracking:', error);
    }
  };

  const getStageStatus = (stageKey: string) => {
    if (!progress) return 'pending';
    return progress.progress[stageKey as keyof typeof progress.progress];
  };

  const getProgressPercentage = () => {
    if (!progress) return 0;

    const completedStages = Object.values(progress.progress).filter(status => status ===
    const totalStages = stages.length;
    return Math.round((completedStages / totalStages) * 100);
  };
```

```
  const getCurrentStageIndex = () => {
    if (!progress) return 0;
    return stages.findIndex(stage => stage.key === progress.currentStage);
  };

  if (loading) {
    return (
      <div className="max-w-4xl mx-auto p-4 md:p-6">
        <div className="flex items-center justify-center py-12">
          <Loader2 className="w-8 h-8 animate-spin mr-3" />
          <span>Loading tracking information...</span>
        </div>
      </div>
    );
  }

  if (!progress) {
    return (
      <div className="max-w-4xl mx-auto p-4 md:p-6">
        <Card>
          <CardContent className="text-center py-12">
            <AlertCircle className="w-16 h-16 text-gray-400 mx-auto mb-4" />
            <h2 className="text-xl font-semibold mb-2">Deposit Not Found</h2>
            <p className="text-gray-600 mb-6">Unable to find tracking information for thi
            <Button onClick={startTracking}>Start Tracking</Button>
          </CardContent>
        </Card>
      </div>
    );
  }

  const progressPercentage = getProgressPercentage();
  const currentStageIndex = getCurrentStageIndex();

  return (
    <div className="max-w-6xl mx-auto p-4 md:p-6 space-y-6">
      {/* Header */}
      <div className="text-center">
        <h1 className="text-2xl md:text-3xl font-bold text-gray-900 mb-2">
          Deposit Tracking
        </h1>
        <p className="text-gray-600">
          Real-time progress for Deposit #{depositId}
        </p>
      </div>

      {/* Progress Overview */}
      <Card className="bg-gradient-to-r from-blue-50 to-indigo-50 border-blue-200">
        <CardContent className="p-6">
          <div className="flex items-center justify-between mb-4">
            <div>
              <h2 className="text-xl font-semibold text-blue-900">
                {progress.statusMessage}
              </h2>
              <p className="text-blue-700">
                Stage {currentStageIndex + 1} of {stages.length} • {progressPercentage}%
```

```
            </p>
          </div>
          <div className="text-right">
            <Badge variant={progressPercentage === 100 ? 'default' : 'secondary'} class
              {progressPercentage === 100 ? 'Completed' : 'In Progress'}
            </Badge>
            <p className="text-sm text-blue-600 mt-1">
              Updated: {lastUpdated.toLocaleTimeString()}
            </p>
          </div>
        </div>

        {/* Progress Bar */}
        <div className="w-full bg-blue-200 rounded-full h-3 mb-4">
          <div
            className="bg-gradient-to-r from-blue-500 to-indigo-600 h-3 rounded-full tr
            style={{ width: `${progressPercentage}%` }}
          />
        </div>

        {/* Estimated Completion */}
        {progress.estimatedCompletion && progressPercentage < 100 && (
          <div className="flex items-center text-sm text-blue-700">
            <Clock className="w-4 h-4 mr-2" />
            <span>Estimated completion: {new Date(progress.estimatedCompletion).toLocal
          </div>
        )}
      </CardContent>
    </Card>

    {/* Detailed Progress Stages */}
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
      {stages.map((stage, index) => {
        const status = getStageStatus(stage.key);
        const Icon = stage.icon;

        return (
          <Card
            key={stage.key}
            className={`transition-all duration-500 ${
              status === 'completed' ? 'bg-green-50 border-green-200' :
              status === 'current' ? 'bg-blue-50 border-blue-300 shadow-lg' :
              'bg-gray-50 border-gray-200'
            }`}
          >
            <CardContent className="p-4">
              <div className="flex items-center justify-between mb-3">
                <div className={`w-10 h-10 rounded-full flex items-center justify-cente
                  status === 'completed' ? 'bg-green-500 text-white' :
                  status === 'current' ? 'bg-blue-500 text-white animate-pulse' :
                  'bg-gray-300 text-gray-600'
                }`}>
                  {status === 'completed' ? (
                    <CheckCircle className="w-5 h-5" />
                  ) : status === 'current' ? (
                    <Loader2 className="w-5 h-5 animate-spin" />
```

```
                ) : (
                  <Icon className="w-5 h-5" />
                )}
              </div>
              <Badge
                variant={
                  status === 'completed' ? 'default' :
                  status === 'current' ? 'secondary' :
                  'outline'
                }
                className="text-xs"
              >
                {status === 'completed' ? 'Complete' :
                 status === 'current' ? 'In Progress' :
                 'Pending'}
              </Badge>
            </div>
            <h3 className={`font-semibold text-sm mb-1 ${
              status === 'completed' ? 'text-green-800' :
              status === 'current' ? 'text-blue-800' :
              'text-gray-700'
            }`}>
              {stage.title}
            </h3>
            <p className={`text-xs ${
              status === 'completed' ? 'text-green-600' :
              status === 'current' ? 'text-blue-600' :
              'text-gray-500'
            }`}>
              {stage.description}
            </p>
          </CardContent>
        </Card>
      );
    })}
  </div>

  {/* Commodity Details */}
  {progress.commodity && (
    <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
      <Card>
        <CardHeader>
          <CardTitle className="text-lg">Commodity Details</CardTitle>
        </CardHeader>
        <CardContent className="space-y-2">
          <div>
            <span className="text-sm text-gray-600">Commodity:</span>
            <p className="font-semibold">{progress.commodity.name}</p>
          </div>
          <div>
            <span className="text-sm text-gray-600">Quantity:</span>
            <p className="font-semibold">{progress.commodity.quantity} {progress.comm
          </div>
          <div>
            <span className="text-sm text-gray-600">Estimated Value:</span>
            <p className="font-semibold text-green-600">₹{progress.commodity.estimate
```

```
            </div>
          </CardContent>
        </Card>

        <Card>
          <CardHeader>
            <CardTitle className="text-lg">Warehouse Details</CardTitle>
          </CardHeader>
          <CardContent className="space-y-2">
            <div>
              <span className="text-sm text-gray-600">Warehouse:</span>
              <p className="font-semibold">{progress.warehouse?.name || 'Delhi Central
            </div>
            <div>
              <span className="text-sm text-gray-600">Location:</span>
              <p className="font-semibold flex items-center">
                <MapPin className="w-4 h-4 mr-1" />
                {progress.warehouse?.address || 'Sector 18, Noida, UP'}
              </p>
            </div>
            <div>
              <span className="text-sm text-gray-600">Contact:</span>
              <p className="font-semibold flex items-center">
                <Phone className="w-4 h-4 mr-1" />
                {progress.warehouse?.contact || '+91-120-2345678'}
              </p>
            </div>
          </CardContent>
        </Card>

        <Card>
          <CardHeader>
            <CardTitle className="text-lg">Transport Details</CardTitle>
          </CardHeader>
          <CardContent className="space-y-2">
            {getStageStatus('pickup_scheduled') !== 'pending' ? (
              <>
                <div>
                  <span className="text-sm text-gray-600">Vehicle:</span>
                  <p className="font-semibold">{progress.vehicle?.number || 'DL-01-AB-1
                </div>
                <div>
                  <span className="text-sm text-gray-600">Driver:</span>
                  <p className="font-semibold flex items-center">
                    <User className="w-4 h-4 mr-1" />
                    {progress.vehicle?.driver || 'Ramesh Kumar'}
                  </p>
                </div>
                <div>
                  <span className="text-sm text-gray-600">Contact:</span>
                  <p className="font-semibold flex items-center">
                    <Phone className="w-4 h-4 mr-1" />
                    {progress.vehicle?.phone || '+91-98765-43210'}
                  </p>
                </div>
              </>
```

```
            ) : (
              <p className="text-sm text-gray-500 italic">Vehicle details will be updat
            )}
          </CardContent>
        </Card>
      </div>
    )}

    {/* Live Updates Toggle */}
    <Card>
      <CardContent className="p-4">
        <div className="flex items-center justify-between">
          <div>
            <h3 className="font-semibold">Live Updates</h3>
            <p className="text-sm text-gray-600">
              {autoRefresh ? 'Automatically refreshing every 10 seconds' : 'Manual refr
            </p>
          </div>
          <div className="flex space-x-3">
            <Button
              variant="outline"
              size="sm"
              onClick={() => setAutoRefresh(!autoRefresh)}
            >
              {autoRefresh ? 'Disable Auto-Refresh' : 'Enable Auto-Refresh'}
            </Button>
            <Button
              size="sm"
              onClick={fetchProgress}
            >
              Refresh Now
            </Button>
          </div>
        </div>
      </CardContent>
    </Card>

    {/* Action Buttons */}
    {progressPercentage === 100 && (
      <Card className="bg-green-50 border-green-200">
        <CardContent className="p-6 text-center">
          <CheckCircle className="w-16 h-16 text-green-600 mx-auto mb-4" />
          <h3 className="text-xl font-semibold text-green-800 mb-2">Deposit Complete!</
          <p className="text-green-700 mb-6">
            Your commodity has been processed and your electronic warehouse receipt is
          </p>
          <div className="flex justify-center space-x-4">
            <Button onClick={() => window.location.href = '/receipts'}>
              View Receipt
            </Button>
            <Button variant="outline" onClick={() => window.location.href = '/loans'}>
              Apply for Loan
            </Button>
          </div>
        </CardContent>
      </Card>
```

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

export default TrackDepositPage;
```

2. ENHANCED DEPOSIT API WITH AUTO-PROGRESSION

Update: `server/routes.ts` - Replace existing deposit API with enhanced version:

```
// WORKING DEPOSIT API WITH REAL-TIME TRACKING
apiRouter.post('/deposits', requireAuth, async (req: Request, res: Response) => {
  try {
    const { commodityName, commodityType, quantity, unit, qualityParams, location } = req
    const userId = req.session!.userId as number;

    if (!commodityName || !commodityType || !quantity) {
      return res.status(400).json({
        success: false,
        error: "Missing required fields: commodityName, commodityType, quantity"
      });
    }

    // Calculate market value
    const basePrice = getCommodityBasePrice(commodityName);
    const marketValue = basePrice * parseFloat(quantity);

    // Create commodity entry
    const commodity = await storage.createCommodity({
      name: commodityName,
      type: commodityType,
      quantity: quantity.toString(),
      measurementUnit: unit || 'MT',
      qualityParameters: qualityParams || {},
      gradeAssigned: 'Pending Assessment',
      warehouseId: 1,
      ownerId: userId,
      status: 'active',
      channelType: 'green',
      valuation: marketValue.toString(),
      marketValue: marketValue.toString()
    });

    console.log("Commodity created:", commodity.id);

    // Create process for tracking
    const process = await storage.createProcess({
      processType: "deposit",
      userId: userId,
      commodityId: commodity.id,
      warehouseId: 1,
      status: "in_progress",
      currentStage: "pickup_scheduled",
      progress: 10,
```

```
      estimatedCompletion: new Date(Date.now() + 4 * 60 * 60 * 1000), // 4 hours
      metadata: {
        commodityName,
        commodityType,
        quantity,
        unit: unit || 'MT',
        estimatedValue: marketValue,
        autoProgression: true
      }
    });

    console.log("Process created:", process.id);

    // Start auto-progression immediately
    setTimeout(() => autoProgressDeposit(process.id), 30000); // 30 seconds

    // Create immediate receipt for demo purposes
    const receiptNumber = `TW${Date.now()}-${Math.random().toString(36).substr(2, 6)}`;
    const receipt = await storage.createWarehouseReceipt({
      receiptNumber,
      commodityId: commodity.id,
      ownerId: userId,
      warehouseId: 1,
      quantity: parseFloat(quantity).toString(),
      valuation: marketValue.toString(),
      status: 'active',
      availableForCollateral: true,
      collateralUsed: '0',
      blockchainHash: `BC-${Date.now()}`,
      qualityGrade: 'Grade A',
      commodityName: commodityName,
      measurementUnit: unit || 'MT',
      issuedDate: new Date(),
      expiryDate: new Date(Date.now() + 365 * 24 * 60 * 60 * 1000)
    });

    res.json({
      success: true,
      data: {
        process: process,
        commodity: commodity,
        receipt: receipt,
        trackingUrl: `/track/${process.id}`,
        message: `Deposit created! Track progress at /track/${process.id}`
      }
    });

  } catch (error: any) {
    console.error('Deposit error:', error);
    res.status(500).json({ success: false, error: error.message });
  }
});

// AUTO-PROGRESSION FUNCTION
async function autoProgressDeposit(processId: number) {
  try {
```

```
    console.log(`Starting auto-progression for process ${processId}`);

    // Stage 1: In Transit (after 30 seconds)
    setTimeout(async () => {
      await updateProcessStage(processId, 'in_transit', 'Vehicle is en route to warehouse
      console.log(`Process ${processId}: Stage 1 - In Transit`);
    }, 30000);

    // Stage 2: Arrived at Warehouse (after 2 minutes)
    setTimeout(async () => {
      await updateProcessStage(processId, 'arrived_warehouse', 'Commodity has arrived at
      console.log(`Process ${processId}: Stage 2 - Arrived at Warehouse`);
    }, 120000);

    // Stage 3: Quality Assessment (after 4 minutes)
    setTimeout(async () => {
      await updateProcessStage(processId, 'quality_check', 'Quality assessment and gradin
      console.log(`Process ${processId}: Stage 3 - Quality Check`);
    }, 240000);

    // Stage 4: Pricing Calculated (after 6 minutes)
    setTimeout(async () => {
      await updateProcessStage(processId, 'pricing_calculated', 'Market pricing and valua
      console.log(`Process ${processId}: Stage 4 - Pricing Complete`);
    }, 360000);

    // Stage 5: Receipt Generated (after 8 minutes)
    setTimeout(async () => {
      await updateProcessStage(processId, 'receipt_generated', 'Electronic warehouse rece
      console.log(`Process ${processId}: Stage 5 - Receipt Generated`);
    }, 480000);

  } catch (error) {
    console.error(`Error in auto-progression for process ${processId}:`, error);
  }
}

async function updateProcessStage(processId: number, stage: string, message: string) {
  try {
    const process = await storage.getProcess(processId);
    if (!process) return;

    const stageProgress = {
      pickup_scheduled: 'completed',
      in_transit: stage === 'in_transit' ? 'current' : (stage === 'pickup_scheduled' ? 'p
      arrived_warehouse: stage === 'arrived_warehouse' ? 'current' :
        (['quality_check', 'pricing_calculated', 'receipt_generated'].includes(stage) ? '
      quality_check: stage === 'quality_check' ? 'current' :
        (['pricing_calculated', 'receipt_generated'].includes(stage) ? 'completed' : 'pen
      pricing_calculated: stage === 'pricing_calculated' ? 'current' :
        (stage === 'receipt_generated' ? 'completed' : 'pending'),
      receipt_generated: stage === 'receipt_generated' ? 'completed' : 'pending'
    };

    await storage.updateProcess(processId, {
      currentStage: stage,
```

```
      stageProgress: stageProgress,
      metadata: JSON.stringify({
        ...JSON.parse(process.metadata || '{}'),
        lastUpdate: new Date().toISOString(),
        statusMessage: message
      })
    });

    // Broadcast WebSocket update if available
    if (typeof globalThis.broadcastEntityUpdate === 'function') {
      globalThis.broadcastEntityUpdate(
        process.userId!,
        'process',
        processId,
        {
          type: 'stage_update',
          currentStage: stage,
          statusMessage: message,
          progress: stageProgress,
          timestamp: new Date().toISOString()
        }
      );
    }

  } catch (error) {
    console.error(`Error updating process stage ${processId}:`, error);
  }
}

// ENHANCED PROGRESS TRACKING API
apiRouter.get("/deposits/:id/progress", requireAuth, async (req: Request, res: Response)
  try {
    const { id } = req.params;
    const userId = req.session!.userId as number;

    const process = await storage.getProcess(parseInt(id));
    if (!process || process.userId !== userId) {
      return res.status(404).json({ message: "Process not found or access denied" });
    }

    const metadata = JSON.parse(process.metadata || '{}');

    // Mock data for demo
    const mockData = {
      id: process.id,
      currentStage: process.currentStage || 'pickup_scheduled',
      statusMessage: metadata.statusMessage || 'Pickup has been scheduled and vehicle ass
      estimatedCompletion: new Date(Date.now() + 2 * 60 * 60 * 1000).toISOString(), // 2
      progress: process.stageProgress || {
        pickup_scheduled: 'current',
        in_transit: 'pending',
        arrived_warehouse: 'pending',
        quality_check: 'pending',
        pricing_calculated: 'pending',
        receipt_generated: 'pending'
      },
```

```
      commodity: {
        name: metadata.commodityName || 'Wheat',
        quantity: metadata.quantity || '10',
        unit: metadata.unit || 'MT',
        estimatedValue: metadata.estimatedValue || 25000
      },
      warehouse: {
        name: 'Delhi Central Warehouse',
        address: 'Sector 18, Noida, UP',
        contact: '+91-120-2345678'
      },
      vehicle: {
        number: 'DL-01-AB-1234',
        driver: 'Ramesh Kumar',
        phone: '+91-98765-43210'
      }
    };

    res.json(mockData);
  } catch (error) {
    console.error('Error fetching deposit progress:', error);
    res.status(500).json({ message: "Failed to fetch progress" });
  }
});

// START TRACKING API
apiRouter.post("/deposits/:id/start-tracking", requireAuth, async (req: Request, res: Res
  try {
    const { id } = req.params;
    const userId = req.session!.userId as number;

    const process = await storage.getProcess(parseInt(id));
    if (!process || process.userId !== userId) {
      return res.status(404).json({ message: "Process not found or access denied" });
    }

    // Start auto-progression if not already started
    autoProgressDeposit(parseInt(id));

    const updatedProcess = await storage.updateProcess(parseInt(id), {
      currentStage: "pickup_scheduled",
      stageProgress: {
        pickup_scheduled: 'current',
        in_transit: 'pending',
        arrived_warehouse: 'pending',
        quality_check: 'pending',
        pricing_calculated: 'pending',
        receipt_generated: 'pending'
      }
    });

    res.json({ success: true, data: updatedProcess });
  } catch (error) {
    console.error('Error starting tracking:', error);
    res.status(500).json({ message: "Failed to start tracking" });
```

```
  }
});
```

3. ENHANCED DEPOSIT FORM WITH TRACKING REDIRECT

Update: `client/src/components/deposit/SimplifiedDepositFlow.tsx` - Add redirect to tracking:

```
// Replace the handleConfirmDeposit function with this enhanced version:

const handleConfirmDeposit = async () => {
  setLoading(true);
  try {
    const response = await fetch('/api/deposits', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      credentials: 'include',
      body: JSON.stringify({
        commodityName: formData.commodityName,
        commodityType: commodityCategories.find(c => c.name === formData.commodityName)?.
        quantity: formData.quantity,
        unit: formData.unit,
        qualityParams: {},
        location: userLocation
      })
    });

    const result = await response.json();

    if (result.success) {
      toast({
        title: "✅ Deposit Created Successfully!",
        description: "Your commodity is being processed. Track real-time progress now.",
        duration: 5000
      });

      // Redirect to tracking page
      setTimeout(() => {
        setLocation(`/track/${result.data.process.id}`);
      }, 2000);
    } else {
      toast({
        title: "Error",
        description: result.error || 'Failed to create deposit',
        variant: "destructive"
      });
    }
  } catch (error) {
    toast({
      title: "Network Error",
      description: "Please check your connection and try again",
      variant: "destructive"
    });
  } finally {
    setLoading(false);
```

```
  }
};
```

4. ADD TRACKING ROUTE TO APP

Update: `client/src/App.tsx` - Add tracking route:

```
// Add this route to your existing routes:
<Route path="/track/:depositId" component={() => <TrackDepositPage />} />
```

5. ADD TRACKING BUTTON TO DASHBOARD

Update: `client/src/components/portfolio/ZerodhaPortfolioDashboard.tsx`:

```
// Add to the holdings table Actions column:
<td className="p-3">
  <Button
    size="sm"
    variant="outline"
    onClick={() => window.location.href = `/track/${receipt.id}`}
  >
    Track
  </Button>
</td>
```

TESTING THE ENHANCED TRACKING SYSTEM

Test Steps:

1. **Create New Deposit**:
   ```
   - Go to /deposits/new
   - Fill: Wheat, 15 MT
   - Submit deposit
   - Note the process ID in response
   ```

2. **Track Real-Time Progress**:
   ```
   - Automatically redirected to /track/{processId}
   - Watch stages progress every 30 seconds:
     * Pickup Scheduled → In Transit → Arrived → Quality Check → Pricing → Receipt
   ```

3. **Visual Features**:
   ```
   - Animated progress bar (0% → 100%)
   - Color-coded stage cards
   - Real-time status updates
   - Completion notifications
   ```

4. **Client Demo Ready**:

```
- Professional progress visualization
- Realistic timing (8 minutes total)
- Live updates without page refresh
- Complete commodity details
```

Expected Timeline:

- **0:30** - In Transit
- **2:00** - Arrived at Warehouse
- **4:00** - Quality Assessment
- **6:00** - Pricing Complete
- **8:00** - Receipt Generated

This creates a **compelling 8-minute client demo** showing real-time commodity processing with professional visual updates!