| **Palm ERP** | $999-2,999/mo | 1-2 weeks | 80% | 100 clients | | **Business Consulting** | $500-2,000/session | 1-2 hours | 90% | 40 sessions/mo | | **Custom Development** | $10K-100K | 1-6 months | 55% | 3 projects | | **Security Services** | $2K-5K/mo | Ongoing | 65% | 30 clients |

### **Revenue Projections (Conservative)**

**Month 1-3: Foundation**

Virtual Assistants: 5 clients × $1,000 = $5,000

Websites: 3 projects × $3,000 = $9,000

Consulting: 10 sessions × $750 = $7,500

Total: $21,500/month

Net Profit (60% margin): $12,900

**Month 4-6: Growth**

Virtual Assistants: 15 clients × $1,000 = $15,000

ERP Subscriptions: 10 clients × $1,500 = $15,000

Transformation: 1 project × $25,000 = $25,000

Websites: 5 projects × $3,000 = $15,000

Total: $70,000/month

Net Profit: $42,000

**Month 7-12: Scale**

Virtual Assistants: 30 clients × $1,200 = $36,000

ERP Subscriptions: 30 clients × $1,500 = $45,000

Transformation: 2 projects × $30,000 = $60,000

Custom Dev: 1 project × $50,000 = $50,000

Security Services: 10 clients × $3,000 = $30,000

Total: $221,000/month

Net Profit: $132,600

**Year 1 Total Revenue: $1.5M - $2M**

## **📊 PAYMENT ANALYTICS DASHBOARD**

### **Real-Time Financial Tracking**

// FRONTEND/src/pages/FinanceDashboard.jsx

export function FinanceDashboard() {

const [metrics, setMetrics] = useState({

todayRevenue: 0,

monthRevenue: 0,

pendingPayments: 0,

cryptoBalance: 0,

stripeBalance: 0

});

useEffect(() => {

loadMetrics();

// Refresh every 30 seconds

const interval = setInterval(loadMetrics, 30000);

return () => clearInterval(interval);

}, []);

return (

<div className="min-h-screen bg-slate-900 p-8">

<h1 className="text-4xl font-bold text-white mb-8">Financial Dashboard</h1>

{/\* Revenue Cards \*/}

<div className="grid md:grid-cols-4 gap-6 mb-8">

<MetricCard

title="Today's Revenue"

value={`${metrics.todayRevenue.toLocaleString()}`}

icon="💰"

trend="+12%"

/>

<MetricCard

title="This Month"

value={`${metrics.monthRevenue.toLocaleString()}`}

icon="📈"

trend="+45%"

/>

<MetricCard

title="Pending Payments"

value={`${metrics.pendingPayments.toLocaleString()}`}

icon="⏳"

count="12 orders"

/>

<MetricCard

title="Total Balance"

value={`${(metrics.cryptoBalance + metrics.stripeBalance).toLocaleString()}`}

icon="🏦"

detail="Ready to withdraw"

/>

</div>

{/\* Payment Method Breakdown \*/}

<div className="grid md:grid-cols-2 gap-6 mb-8">

<div className="bg-slate-800 p-6 rounded-lg">

<h3 className="text-white text-xl font-bold mb-4">Revenue by Method</h3>

<div className="space-y-4">

<PaymentMethodBar method="Stripe" amount={45000} percentage={60} />

<PaymentMethodBar method="USDT" amount={25000} percentage={33} />

<PaymentMethodBar method="Wire" amount={5000} percentage={7} />

</div>

</div>

<div className="bg-slate-800 p-6 rounded-lg">

<h3 className="text-white text-xl font-bold mb-4">Recent Transactions</h3>

<TransactionList />

</div>

</div>

{/\* Withdrawal Options \*/}

<div className="bg-slate-800 p-6 rounded-lg">

<h3 className="text-white text-xl font-bold mb-4">Withdraw Funds</h3>

<div className="grid md:grid-cols-3 gap-4">

<WithdrawOption

method="Stripe Payout"

balance="$45,230"

time="1-2 business days"

available={true}

/>

<WithdrawOption

method="Crypto Exchange"

balance="$25,000 USDT"

time="Instant"

available={true}

/>

<WithdrawOption

method="P2P Transfer"

balance="$25,000 USDT"

time="Minutes"

available={true}

/>

</div>

</div>

</div>

);

}

## **🔄 CURRENCY CONVERSION AUTOMATION**

### **Auto-Convert Everything to Base Currency**

// BACKEND/currency-converter/index.ts

import { createClient } from '@supabase/supabase-js';

const supabase = createClient(

Deno.env.get('SUPABASE\_URL'),

Deno.env.get('SUPABASE\_SERVICE\_KEY')

);

// Using multiple exchange rate APIs for reliability

const EXCHANGE\_RATE\_APIS = [

# Dig|lit Payment Infrastructure & Revenue Model

## Universal Multi-Currency Payment System

\*\*Mission\*\*: Accept payments globally in any currency, deposit to single bank account automatically.

---

## 🌐 PAYMENT ARCHITECTURE OVERVIEW

### The Universal Payment Flow

CLIENT INITIATES PAYMENT

↓

┌─────────────────┐

│ Payment Router │ (AI selects optimal method)

└─────────────────┘

↓

┌─────────────────────┼─────────────────────┐

↓ ↓ ↓

[CRYPTO PATH] [CARD PATH] [WIRE PATH] ↓ ↓ ↓ USDT TRC20 Stripe Connect Bank Transfer (Tron Network) (135+ currencies) (International) ↓ ↓ ↓ TronLink API Auto-conversion Manual verify ↓ ↓ ↓ Verify TX Process payment Confirm receipt ↓ ↓ ↓ Smart Escrow Escrow (optional) Invoice matching ↓ ↓ ↓ Milestone Release Instant/delayed Project completion ↓ ↓ ↓ └─────────────────────┼─────────────────────┘ ↓ YOUR BANK ACCOUNT (Single destination) ↓ Auto-converted to USD/PKR ↓ Available for withdrawal

---

## 💰 METHOD 1: USDT (TRC20) - CRYPTO NATIVE

### Why USDT on Tron?

- \*\*Near-zero fees\*\*: ~$1 per transaction (vs $20+ on Ethereum)

- \*\*Fast settlement\*\*: 3-second block time

- \*\*Stable value\*\*: Pegged to USD

- \*\*Global reach\*\*: No borders, no banks required

- \*\*24/7 operation\*\*: Never closes

### Technical Implementation

#### 1. Wallet Setup (One-Time)

```bash

# Install TronLink Chrome Extension

# Download from: tronlink.org

# Create wallet

1. Generate new wallet

2. BACKUP seed phrase (CRITICAL!)

3. Get your TRC20 address (starts with 'T')

4. Fund with small TRX (~5 TRX for fees)

# Your wallet address example:

TYourWalletAddressHere123456789ABC

#### **2. Frontend Payment Component**

// FRONTEND/src/components/USDTPayment.jsx

import TronWeb from 'tronweb';

import { useState } from 'react';

const USDT\_CONTRACT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'; // Official USDT TRC20

const DIGLIT\_WALLET = 'YOUR\_TRON\_WALLET\_ADDRESS';

const FEE\_WALLET = 'BACKUP\_WALLET\_ADDRESS'; // For multi-sig security

export function USDTPayment({ amount, orderId, onSuccess }) {

const [status, setStatus] = useState('idle');

const [txHash, setTxHash] = useState('');

const payWithUSDT = async () => {

try {

setStatus('connecting');

// Check if TronLink is installed

if (!window.tronWeb || !window.tronWeb.ready) {

alert('Please install TronLink wallet extension');

window.open('https://www.tronlink.org/', '\_blank');

return;

}

setStatus('processing');

// Get contract instance

const tronWeb = window.tronWeb;

const contract = await tronWeb.contract().at(USDT\_CONTRACT);

// Convert amount to Sun (6 decimals for USDT)

const amountSun = amount \* 1000000;

// Send transaction

const tx = await contract.transfer(

DIGLIT\_WALLET,

amountSun

).send({

feeLimit: 100000000,

callValue: 0,

shouldPollResponse: true

});

setTxHash(tx);

setStatus('confirming');

// Verify transaction on backend

const verified = await verifyTransaction(tx, orderId, amount);

if (verified) {

setStatus('success');

onSuccess(tx);

} else {

setStatus('failed');

alert('Transaction verification failed. Please contact support.');

}

} catch (error) {

console.error('USDT Payment Error:', error);

setStatus('failed');

alert('Payment failed: ' + error.message);

}

};

const verifyTransaction = async (txHash, orderId, amount) => {

const response = await fetch('/api/verify-crypto-payment', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ txHash, orderId, amount, currency: 'USDT' })

});

const data = await response.json();

return data.verified;

};

return (

<div className="bg-slate-800 p-6 rounded-lg">

<h3 className="text-white text-xl font-bold mb-4">Pay with USDT</h3>

<div className="mb-4">

<div className="text-white/60 text-sm mb-2">Amount to Pay</div>

<div className="text-3xl font-bold text-green-400">{amount} USDT</div>

<div className="text-white/40 text-xs mt-1">

Network: Tron (TRC20) • Fee: ~$1

</div>

</div>

{status === 'idle' && (

<button

onClick={payWithUSDT}

className="w-full bg-green-600 hover:bg-green-500 py-4 rounded-lg text-white font-bold transition-all">

Pay with TronLink Wallet

</button>

)}

{status === 'connecting' && (

<div className="text-center py-4">

<div className="text-white">Connecting to TronLink...</div>

</div>

)}

{status === 'processing' && (

<div className="text-center py-4">

<div className="animate-spin text-4xl mb-2">⚡</div>

<div className="text-white">Processing payment...</div>

</div>

)}

{status === 'confirming' && (

<div className="text-center py-4">

<div className="animate-pulse text-4xl mb-2">🔍</div>

<div className="text-white">Verifying transaction...</div>

<div className="text-white/60 text-xs mt-2">TX: {txHash.substring(0, 16)}...</div>

</div>

)}

{status === 'success' && (

<div className="text-center py-4">

<div className="text-6xl mb-2">✅</div>

<div className="text-green-400 font-bold">Payment Confirmed!</div>

<a

href={`https://tronscan.org/#/transaction/${txHash}`}

target="\_blank"

className="text-purple-400 text-sm mt-2 inline-block">

View on TronScan →

</a>

</div>

)}

{status === 'failed' && (

<div className="text-center py-4">

<div className="text-6xl mb-2">❌</div>

<div className="text-red-400 font-bold">Payment Failed</div>

<button

onClick={() => setStatus('idle')}

className="mt-4 text-purple-400 hover:text-purple-300">

Try Again

</button>

</div>

)}

<div className="mt-6 p-4 bg-slate-900 rounded text-xs text-white/60">

<div className="font-bold mb-2">Don't have USDT?</div>

<div>Buy USDT on: Binance, Coinbase, or Kraken</div>

<div className="mt-2">Send to your TronLink wallet (TRC20 network)</div>

</div>

</div>

);

}

#### **3. Backend Verification (Critical!)**

// BACKEND/verify-crypto-payment/index.ts

import TronWeb from 'tronweb';

import { createClient } from '@supabase/supabase-js';

const tronWeb = new TronWeb({

fullHost: 'https://api.trongrid.io',

headers: { 'TRON-PRO-API-KEY': Deno.env.get('TRONGRID\_API\_KEY') }

});

const supabase = createClient(

Deno.env.get('SUPABASE\_URL'),

Deno.env.get('SUPABASE\_SERVICE\_KEY')

);

const USDT\_CONTRACT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t';

const DIGLIT\_WALLET = Deno.env.get('DIGLIT\_TRON\_WALLET');

export default async (req) => {

const { txHash, orderId, amount, currency } = await req.json();

try {

// Get transaction details

const tx = await tronWeb.trx.getTransaction(txHash);

// Verify transaction exists

if (!tx || !tx.ret || tx.ret[0].contractRet !== 'SUCCESS') {

return new Response(JSON.stringify({

verified: false,

error: 'Transaction not found or failed'

}));

}

// Decode contract parameters

const contract = tx.raw\_data.contract[0];

const parameter = contract.parameter.value;

// Verify recipient address

const toAddress = tronWeb.address.fromHex(parameter.to);

if (toAddress !== DIGLIT\_WALLET) {

return new Response(JSON.stringify({

verified: false,

error: 'Wrong recipient address'

}));

}

// Verify amount (USDT has 6 decimals)

const transferAmount = parseInt(parameter.amount) / 1000000;

const expectedAmount = parseFloat(amount);

if (Math.abs(transferAmount - expectedAmount) > 0.01) {

return new Response(JSON.stringify({

verified: false,

error: 'Amount mismatch'

}));

}

// Check if transaction already processed

const { data: existing } = await supabase

.from('transactions')

.select('id')

.eq('tx\_hash', txHash)

.single();

if (existing) {

return new Response(JSON.stringify({

verified: false,

error: 'Transaction already processed'

}));

}

// Record transaction

await supabase.from('transactions').insert({

tx\_hash: txHash,

order\_id: orderId,

amount: transferAmount,

currency: 'USDT',

network: 'TRC20',

status: 'confirmed',

from\_address: tronWeb.address.fromHex(parameter.owner\_address),

to\_address: toAddress,

block\_number: tx.blockNumber,

confirmed\_at: new Date(tx.block\_timestamp)

});

// Update order status

await supabase

.from('orders')

.update({

payment\_status: 'paid',

payment\_tx: txHash,

paid\_at: new Date()

})

.eq('id', orderId);

// Trigger fulfillment (send to queue)

await triggerOrderFulfillment(orderId);

return new Response(JSON.stringify({

verified: true,

txHash,

amount: transferAmount

}));

} catch (error) {

console.error('Verification error:', error);

return new Response(JSON.stringify({

verified: false,

error: error.message

}), { status: 500 });

}

};

async function triggerOrderFulfillment(orderId) {

// Add to fulfillment queue

await supabase.from('fulfillment\_queue').insert({

order\_id: orderId,

status: 'pending',

priority: 'high'

});

// Send notification email

// Assign to team member

// Update client dashboard

}

#### **4. Transaction Database Schema**

-- Store all crypto transactions

CREATE TABLE transactions (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

tx\_hash TEXT UNIQUE NOT NULL,

order\_id UUID REFERENCES orders(id),

amount DECIMAL(20,6) NOT NULL,

currency TEXT NOT NULL, -- 'USDT', 'BTC', 'ETH'

network TEXT NOT NULL, -- 'TRC20', 'ERC20', 'BEP20'

status TEXT DEFAULT 'pending', -- 'pending', 'confirmed', 'failed'

from\_address TEXT NOT NULL,

to\_address TEXT NOT NULL,

block\_number BIGINT,

confirmations INTEGER DEFAULT 0,

confirmed\_at TIMESTAMP,

created\_at TIMESTAMP DEFAULT NOW()

);

-- Index for fast lookups

CREATE INDEX idx\_tx\_hash ON transactions(tx\_hash);

CREATE INDEX idx\_order\_id ON transactions(order\_id);

CREATE INDEX idx\_status ON transactions(status);

### **Converting USDT to Your Bank Account**

**Option 1: Exchange Withdrawal (Recommended)**

1. Send USDT from Tron wallet to Binance/Coinbase

2. Convert USDT → USD/PKR on exchange

3. Withdraw to bank account

4. Timeline: 1-3 business days

5. Fees: 0.1% trading + withdrawal fee

**Option 2: P2P Trading**

1. Use Binance P2P or LocalBitcoins

2. Sell USDT directly to buyer

3. Receive bank transfer

4. Timeline: Minutes to hours

5. Fees: 0% (built into exchange rate)

**Option 3: OTC Desk (High Volume)**

1. Contact OTC desk (Kraken, Circle)

2. Negotiate rate for large amounts ($50K+)

3. Wire transfer to account

4. Timeline: Same day

5. Fees: Negotiable (0.1-0.5%)

## **💳 METHOD 2: STRIPE CONNECT - UNIVERSAL FIAT**

### **Why Stripe Connect?**

* **135+ currencies** supported
* **Deposits to YOUR bank** automatically
* **No intermediary accounts** needed
* **Professional checkout** experience
* **Fraud protection** built-in
* **Compliance handled** by Stripe

### **Architecture**

CLIENT PAYMENT → STRIPE → YOUR CONNECTED ACCOUNT → YOUR BANK

↓

(Auto-conversion happens here)

↓

All currencies → USD/PKR

### **Implementation**

#### **1. Stripe Account Setup**

# Create Stripe Account

1. Go to stripe.com/connect

2. Sign up for business account

3. Complete business verification (1-2 days)

4. Enable Stripe Connect

5. Add bank account for payouts

# Get API Keys

Dashboard → Developers → API Keys

- Publishable key (pk\_live\_...)

- Secret key (sk\_live\_...)

- Webhook signing secret (whsec\_...)

#### **2. Stripe Connect Integration**

// BACKEND/create-payment-intent/index.ts

import Stripe from 'stripe';

import { createClient } from '@supabase/supabase-js';

const stripe = new Stripe(Deno.env.get('STRIPE\_SECRET\_KEY'));

const supabase = createClient(

Deno.env.get('SUPABASE\_URL'),

Deno.env.get('SUPABASE\_SERVICE\_KEY')

);

export default async (req) => {

const {

amount,

currency = 'usd',

orderId,

customerEmail,

description

} = await req.json();

try {

// Create or retrieve customer

let customer;

const { data: existingCustomer } = await supabase

.from('customers')

.select('stripe\_customer\_id')

.eq('email', customerEmail)

.single();

if (existingCustomer?.stripe\_customer\_id) {

customer = await stripe.customers.retrieve(existingCustomer.stripe\_customer\_id);

} else {

customer = await stripe.customers.create({

email: customerEmail,

metadata: { order\_id: orderId }

});

// Save to database

await supabase.from('customers').insert({

email: customerEmail,

stripe\_customer\_id: customer.id

});

}

// Create payment intent

const paymentIntent = await stripe.paymentIntents.create({

amount: Math.round(amount \* 100), // Convert to cents

currency: currency.toLowerCase(),

customer: customer.id,

description: description || `Dig|lit Order ${orderId}`,

metadata: {

order\_id: orderId,

platform: 'diglit'

},

automatic\_payment\_methods: {

enabled: true,

},

receipt\_email: customerEmail

});

// Store intent in database

await supabase.from('payment\_intents').insert({

intent\_id: paymentIntent.id,

order\_id: orderId,

amount: amount,

currency: currency,

status: 'pending',

customer\_email: customerEmail

});

return new Response(JSON.stringify({

clientSecret: paymentIntent.client\_secret,

intentId: paymentIntent.id

}));

} catch (error) {

console.error('Payment Intent Error:', error);

return new Response(JSON.stringify({

error: error.message

}), { status: 500 });

}

};

#### **3. Frontend Stripe Component**

// FRONTEND/src/components/StripePayment.jsx

import { useState } from 'react';

import { loadStripe } from '@stripe/stripe-js';

import {

Elements,

PaymentElement,

useStripe,

useElements

} from '@stripe/react-stripe-js';

const stripePromise = loadStripe(import.meta.env.VITE\_STRIPE\_PUBLIC\_KEY);

function CheckoutForm({ amount, orderId, onSuccess }) {

const stripe = useStripe();

const elements = useElements();

const [processing, setProcessing] = useState(false);

const [message, setMessage] = useState('');

const handleSubmit = async (e) => {

e.preventDefault();

if (!stripe || !elements) return;

setProcessing(true);

setMessage('');

const { error, paymentIntent } = await stripe.confirmPayment({

elements,

confirmParams: {

return\_url: `${window.location.origin}/payment-success?order=${orderId}`,

},

redirect: 'if\_required'

});

if (error) {

setMessage(error.message);

setProcessing(false);

} else if (paymentIntent.status === 'succeeded') {

onSuccess(paymentIntent.id);

setMessage('Payment successful!');

}

setProcessing(false);

};

return (

<form onSubmit={handleSubmit} className="space-y-6">

<PaymentElement />

<button

type="submit"

disabled={!stripe || processing}

className="w-full bg-purple-600 hover:bg-purple-500 disabled:bg-gray-600 py-4 rounded-lg text-white font-bold transition-all">

{processing ? 'Processing...' : `Pay ${amount}`}

</button>

{message && (

<div className={`p-4 rounded-lg text-center ${

message.includes('success')

? 'bg-green-900/30 text-green-400'

: 'bg-red-900/30 text-red-400'

}`}>

{message}

</div>

)}

<div className="text-center text-white/40 text-xs">

<div>🔒 Secured by Stripe</div>

<div className="mt-1">Accepts all major cards + Google Pay + Apple Pay</div>

</div>

</form>

);

}

export function StripePayment({ amount, orderId, customerEmail, onSuccess }) {

const [clientSecret, setClientSecret] = useState('');

useState(() => {

// Create payment intent

fetch('/api/create-payment-intent', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({

amount,

orderId,

customerEmail,

currency: 'usd'

})

})

.then(res => res.json())

.then(data => setClientSecret(data.clientSecret));

}, []);

const options = {

clientSecret,

appearance: {

theme: 'night',

variables: {

colorPrimary: '#a855f7',

colorBackground: '#1e293b',

colorText: '#ffffff',

colorDanger: '#ef4444',

fontFamily: 'system-ui, sans-serif',

borderRadius: '8px',

}

}

};

return (

<div className="bg-slate-800 p-6 rounded-lg">

<h3 className="text-white text-xl font-bold mb-4">Pay with Card</h3>

<div className="mb-6">

<div className="text-white/60 text-sm mb-2">Total Amount</div>

<div className="text-3xl font-bold text-purple-400">${amount}</div>

<div className="text-white/40 text-xs mt-1">

Supports 135+ currencies • Instant processing

</div>

</div>

{clientSecret && (

<Elements stripe={stripePromise} options={options}>

<CheckoutForm

amount={amount}

orderId={orderId}

onSuccess={onSuccess}

/>

</Elements>

)}

</div>

);

}

#### **4. Webhook Handler (Critical for Automation)**

// BACKEND/stripe-webhook/index.ts

import Stripe from 'stripe';

import { createClient } from '@supabase/supabase-js';

const stripe = new Stripe(Deno.env.get('STRIPE\_SECRET\_KEY'));

const supabase = createClient(

Deno.env.get('SUPABASE\_URL'),

Deno.env.get('SUPABASE\_SERVICE\_KEY')

);

export default async (req) => {

const signature = req.headers.get('stripe-signature');

const body = await req.text();

let event;

try {

event = stripe.webhooks.constructEvent(

body,

signature,

Deno.env.get('STRIPE\_WEBHOOK\_SECRET')

);

} catch (err) {

console.error('Webhook signature verification failed:', err);

return new Response('Invalid signature', { status: 400 });

}

// Handle different event types

switch (event.type) {

case 'payment\_intent.succeeded':

await handlePaymentSuccess(event.data.object);

break;

case 'payment\_intent.payment\_failed':

await handlePaymentFailure(event.data.object);

break;

case 'charge.refunded':

await handleRefund(event.data.object);

break;

case 'customer.created':

console.log('New customer:', event.data.object.id);

break;

default:

console.log('Unhandled event type:', event.type);

}

return new Response(JSON.stringify({ received: true }));

};

async function handlePaymentSuccess(paymentIntent) {

const orderId = paymentIntent.metadata.order\_id;

// Update order status

await supabase

.from('orders')

.update({

payment\_status: 'paid',

payment\_method: 'stripe',

payment\_intent\_id: paymentIntent.id,

paid\_at: new Date(),

amount\_received: paymentIntent.amount / 100,

currency: paymentIntent.currency

})

.eq('id', orderId);

// Update payment intent record

await supabase

.from('payment\_intents')

.update({ status: 'succeeded' })

.eq('intent\_id', paymentIntent.id);

// Trigger fulfillment

await supabase.from('fulfillment\_queue').insert({

order\_id: orderId,

status: 'pending',

priority: 'high',

payment\_method: 'stripe'

});

// Send confirmation email

await sendOrderConfirmationEmail(orderId);

console.log(`Order ${orderId} payment confirmed via Stripe`);

}

async function handlePaymentFailure(paymentIntent) {

const orderId = paymentIntent.metadata.order\_id;

await supabase

.from('orders')

.update({

payment\_status: 'failed',

payment\_error: paymentIntent.last\_payment\_error?.message

})

.eq('id', orderId);

// Notify customer

await sendPaymentFailureEmail(orderId);

}

async function handleRefund(charge) {

// Find order by charge ID

const { data: order } = await supabase

.from('orders')

.select('\*')

.eq('payment\_intent\_id', charge.payment\_intent)

.single();

if (order) {

await supabase

.from('orders')

.update({

payment\_status: 'refunded',

refunded\_at: new Date(),

refund\_amount: charge.amount\_refunded / 100

})

.eq('id', order.id);

// Notify customer

await sendRefundConfirmationEmail(order.id);

}

}

#### **5. Automatic Payouts Configuration**

// Stripe Dashboard Settings

Payouts:

✅ Automatic payouts: Enabled

✅ Payout schedule: Daily

✅ Payout speed: Standard (2-3 days)

✅ Bank account: [Your bank details]

Currency Conversion:

✅ Automatic conversion: Enabled

✅ Convert to: USD (or PKR if available)

✅ Conversion rate: Mid-market rate

✅ Fee: 1% (included in Stripe fees)

// All incoming payments automatically convert and deposit to your bank

## **🏦 METHOD 3: WIRE TRANSFER - ENTERPRISE**

### **For High-Value Transactions ($10K+)**

#### **Implementation**

// FRONTEND/src/components/WireTransferPayment.jsx

export function WireTransferPayment({ amount, orderId, currency = 'USD' }) {

const [copied, setCopied] = useState(false);

const bankDetails = {

USD: {

bankName: 'Your Bank Name',

accountName: 'Dig|lit LLC',

accountNumber: 'XXXXXXXXXX',

routingNumber: 'XXXXXXXXX',

swiftCode: 'XXXXXXXX',

address: 'Bank Address'

},

EUR: {

bankName: 'European Bank',

iban: 'EUXXXXXXXXXXXXXXXXXX',

swiftCode: 'XXXXXXXX'

},

PKR: {

bankName: 'Pakistani Bank',

accountName: 'Dig|lit',

accountNumber: 'XXXXXXXXXX',

branchCode: 'XXXX'

}

};

const details = bankDetails[currency] || bankDetails.USD;

const copyDetails = () => {

const text = Object.entries(details)

.map(([key, value]) => `${key}: ${value}`)

.join('\n');

navigator.clipboard.writeText(text);

setCopied(true);

setTimeout(() => setCopied(false), 2000);

};

return (

<div className="bg-slate-800 p-6 rounded-lg">

<h3 className="text-white text-xl font-bold mb-4">Wire Transfer</h3>

<div className="mb-6">

<div className="text-white/60 text-sm mb-2">Amount to Transfer</div>

<div className="text-3xl font-bold text-blue-400">

{currency} {amount.toLocaleString()}

</div>

</div>

<div className="bg-slate-900 p-4 rounded-lg space-y-3 mb-4">

{Object.entries(details).map(([key, value]) => (

<div key={key}>

<div className="text-white/60 text-xs uppercase">{key}</div>

<div className="text-white font-mono">{value}</div>

</div>

))}

</div>

<div className="bg-yellow-900/30 border border-yellow-600/50 p-4 rounded-lg mb-4">

<div className="text-yellow-400 font-bold mb-2">⚠️ Important</div>

<ul className="text-yellow-200 text-sm space-y-1">

<li>• Include Order ID: {orderId} in reference</li>

<li>• Transfer exact amount: {currency} {amount}</li>

<li>• Processing time: 1-3 business days</li>

<li>• Upload proof of transfer below</li>

</ul>

</div>

<button

onClick={copyDetails}

className="w-full bg-blue-600 hover:bg-blue-500 py-3 rounded-lg text-white font-bold mb-4">

{copied ? '✓ Copied!' : '📋 Copy Bank Details'}

</button>

<div>

<label className="block text-white mb-2">Upload Transfer Proof</label>

<input

type="file"

accept="image/\*,application/pdf"

className="w-full bg-slate-700 text-white p-3 rounded-lg"

onChange={(e) => uploadProof(e.target.files[0], orderId)}

/>

</div>

</div>

);

}

async function uploadProof(file, orderId) {

const formData = new FormData();

formData.append('file', file);

formData.append('orderId', orderId);

await fetch('/api/upload-payment-proof', {

method: 'POST',

body: formData

});

alert('Proof uploaded! We will verify and confirm within 24 hours.');

}

## **🔐 SMART CONTRACT ESCROW (OPTIONAL)**

### **For Trust & Milestone-Based Payments**

// BACKEND/contracts/DigilitEscrow.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract DigilitEscrow is ReentrancyGuard {

address public admin;

uint256 public escrowCounter;

struct Escrow {

address client;

address provider;

uint256 amount;

address token; // USDT contract address

uint256 milestones;

uint256 releasedMilestones;

bool completed;

bool disputed;

}

mapping(uint256 => Escrow) public escrows;

event EscrowCreated(uint256 indexed escrowId, address client, uint256 amount);

event MilestoneReleased(uint256 indexed escrowId, uint256 milestone, uint256 amount);

event EscrowCompleted(uint256 indexed escrowId);

constructor() {

admin = msg.sender;

}

function createEscrow(

address \_provider,

uint256 \_amount,

address \_token,

uint256 \_milestones

) external returns (uint256) {

require(\_milestones > 0 && \_milestones <= 10, "Invalid milestones");

escrowCounter++;

escrows[escrowCounter] = Escrow({

client: msg.sender,

provider: \_provider,

amount: \_amount,

token: \_token,

milestones: \_milestones,

releasedMilestones: 0,

completed: false,

disputed: false

});

// Transfer tokens to escrow

IERC20(\_token).transferFrom(msg.sender, address(this), \_amount);

emit EscrowCreated(escrowCounter, msg.sender, \_amount);

return escrowCounter;

}

function releaseMilestone(uint256 \_escrowId) external nonReentrant {

Escrow storage escrow = escrows[\_escrowId];

require(msg.sender == escrow.client || msg.sender == admin, "Not authorized");

require(!escrow.completed, "Escrow completed");

require(!escrow.disputed, "Escrow disputed");

require(escrow.releasedMilestones < escrow.milestones, "All milestones released");

escrow.releasedMilestones++;

uint256 releaseAmount = escrow.amount / escrow.milestones;

IERC20(escrow.token).transfer(escrow.provider, releaseAmount);

emit MilestoneReleased(\_escrowId, escrow.releasedMilestones, releaseAmount);

if (escrow.releasedMilestones == escrow.milestones) {

escrow.completed = true;

emit EscrowCompleted(\_escrowId);

}

}

function disputeEscrow(uint256 \_escrowId) external {

Escrow storage escrow = escrows[\_escrowId];

require(msg.sender == escrow.client || msg.sender == escrow.provider, "Not authorized");

require(!escrow.completed, "Escrow completed");

escrow.disputed = true;

// Admin will resolve manually

}

}

**Deploy to Tron Network:**

# Use TronBox or TronIDE

npm install -g tronbox

tronbox compile

tronbox migrate --network mainnet

# Contract address: Save for frontend integration

## **🌍 KYC/AML COMPLIANCE**

### **Automated Identity Verification**

// BACKEND/kyc-verification/index.ts

import { Sumsub } from '@sumsub/api';

const sumsub = new Sumsub({

appToken: Deno.env.get('SUMSUB\_APP\_TOKEN'),

secretKey: Deno.env.get('SUMSUB\_SECRET\_KEY'),

baseURL: 'https://api.sumsub.com'

});

export default async (req) => {

const { userId, email } = await req.json();

// Create applicant

const applicant = await sumsub.createApplicant({

externalUserId: userId,

email: email,

fixedInfo: {

firstName: '',

lastName: ''

}

});

// Generate access token for verification flow

const accessToken = await sumsub.getAccessToken(applicant.id);

// Store in database

await supabase.from('kyc\_verifications').insert({

user\_id: userId,

applicant\_id: applicant.id,

status: 'pending',

created\_at: new Date()

});

return new Response(JSON.stringify({

applicantId: applicant.id,

accessToken: accessToken

}));

};

### **Frontend KYC Component**

// FRONTEND/src/components/KYCVerification.jsx

import SumsubWebSdk from '@sumsub/websdk-react';

export function KYCVerification({ userId, onComplete }) {

const [accessToken, setAccessToken] = useState('');

useEffect(() => {

// Get access token from backend

fetch('/api/kyc-verification', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ userId, email: user.email })

})

.then(res => res.json())

.then(data => setAccessToken(data.accessToken));

}, []);

const config = {

lang: 'en',

theme: 'dark',

i18n: {

document: {

subTitles: {

IDENTITY: 'Upload ID for verification'

}

}

}

};

const options = {

addViewportTag: false,

adaptIframeHeight: true

};

const handleMessage = (type, payload) => {

if (type === 'idCheck.onStepCompleted') {

console.log('Verification step completed');

}

if (type === 'idCheck.applicantStatus') {

if (payload.reviewStatus === 'completed') {

onComplete();

}

}

};

return (

<div className="bg-slate-800 p-6 rounded-lg">

<h3 className="text-white text-xl font-bold mb-4">Identity Verification</h3>

<p className="text-white/60 mb-6">

Complete KYC verification to access all features

</p>

{accessToken && (

<SumsubWebSdk

accessToken={accessToken}

expirationHandler={() => refreshToken()}

config={config}

options={options}

onMessage={handleMessage}

/>

)}

</div>

);

}

### **Compliance Thresholds**

// BACKEND/compliance-check/index.ts

const THRESHOLDS = {

NO\_KYC: 1000, // Under $1K: No KYC required

BASIC\_KYC: 10000, // $1K-$10K: Basic KYC (ID only)

FULL\_KYC: 50000, // $10K-$50K: Full KYC (ID + address)

ENHANCED: Infinity // $50K+: Enhanced due diligence

};

export function getKYCLevel(amount) {

if (amount < THRESHOLDS.NO\_KYC) return 'none';

if (amount < THRESHOLDS.BASIC\_KYC) return 'basic';

if (amount < THRESHOLDS.FULL\_KYC) return 'full';

return 'enhanced';

}

// Chainalysis for crypto transactions

import { Chainalysis } from 'chainalysis-sdk';

const chainalysis = new Chainalysis(Deno.env.get('CHAINALYSIS\_API\_KEY'));

async function screenCryptoAddress(address, network) {

const result = await chainalysis.screenAddress({

address,

asset: network === 'TRC20' ? 'USDT-TRC20' : 'ETH'

});

if (result.riskLevel === 'high' || result.sanctions) {

// Block transaction

return { allowed: false, reason: 'High-risk address' };

}

return { allowed: true };

}

## **💰 REVENUE MODEL & PROJECTIONS**

### **Service Pricing Matrix**

| **Service** | **Price Range** | **Delivery Time** | **Margin** | **Monthly Capacity** |
| --- | --- | --- | --- | --- |
| **Virtual Assistant** | $500-2,000/mo | Immediate | 70% | 50 clients |
| **Website Development** | $1,500-5,000 | 3-5 days | 60% | 20 projects |
| **Digital Transformation** | $10K-50K | 3-6 months | 50% | 5 projects |
| \*\*Palm ERP |  |  |  |  |

**Note** : create a calculator that will jot needs and share the dynamic range or ask for budget discounts different packages