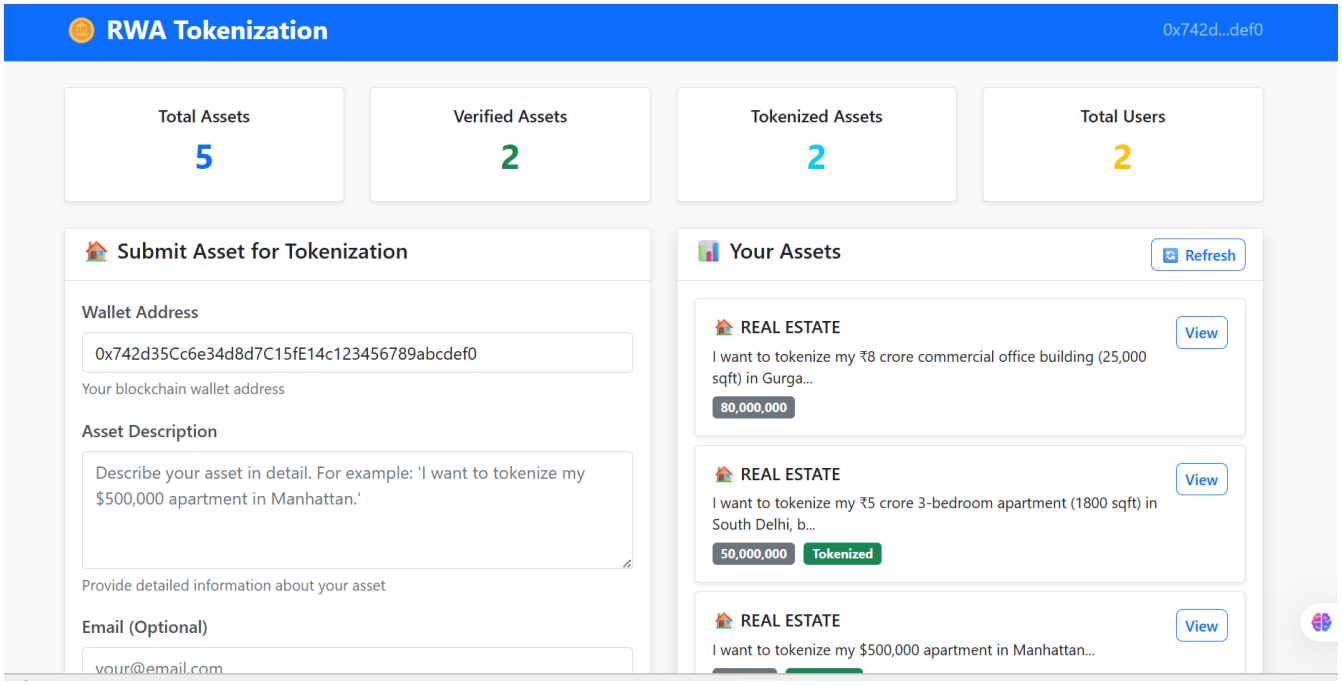
# RWA Tokenization App

# Developer & User Manual

Version 1.0

# Screenshot of Application



# File: app/main.py

```
import sys
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from flask import Flask, request, jsonify, render_template
from flask_sqlalchemy import SQLAlchemy
from flask_cors import CORS
import os
import json
import logging
from datetime import datetime

from app.models.database import db, User, Asset, Transaction

from app.agents.nlp_agent import NLPAgent

from app.agents.verification_agent import VerificationAgent
from app.agents.tokenization_agent import TokenizationAgent

from flask import Flask


app = Flask(__name__, template_folder='../templates', static_folder='../static')
app.config['SECRET_KEY'] = 'your-secret-key-change-this'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///rwa_tokenization.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False



# Initialize extensions
db.init_app(app)
CORS(app)

# Initialize agents
nlp_agent = NLPAgent()
verification_agent = VerificationAgent()
tokenization_agent = TokenizationAgent()

# Configure logging
os.makedirs('logs', exist_ok=True)
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('logs/app.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

# Create tables
with app.app_context():
    db.create_all()

# Simple root route
@app.route('/')
```

```
def home():
    return render_template('index.html')

# ■ Basic Health Check for system_test.sh compatibility
@app.route('/health')
def health_check_basic():
    return jsonify(status="ok"), 200

# Detailed Health Check
@app.route('/api/health')
def health_check():
    return jsonify({
        'status': 'healthy',
        'timestamp': datetime.utcnow().isoformat(),
        'version': '1.0.0'
    })

# Asset Intake Route
@app.route('/api/intake', methods=['POST'])
def asset_intake():
    try:
        data = request.get_json()

        if not data or 'user_input' not in data or 'wallet_address' not in data:
            return jsonify({'error': 'Missing required fields: user_input, wallet_address'}), 400

        user_input = data['user_input']
        wallet_address = data['wallet_address']
        logger.info(f"Processing intake for wallet: {wallet_address}")

        parsed_data = nlp_agent.parse_user_input(user_input)

        user = User.query.filter_by(wallet_address=wallet_address).first()
        if not user:
            user = User(
                wallet_address=wallet_address,
                email=data.get('email'),
                jurisdiction=parsed_data.get('location', '').split(',')[-1].strip()[:2]
            )
            db.session.add(user)
            db.session.commit()

        asset = Asset(
            user_id=user.id,
            asset_type=parsed_data.get('asset_type', 'unknown'),
            description=parsed_data.get('description', user_input),
            estimated_value=parsed_data.get('estimated_value', 0),
            location=parsed_data.get('location', 'Unknown'),
            requirements=json.dumps({
                'confidence_score': parsed_data.get('confidence_score', 0),
                'sentiment': parsed_data.get('sentiment', {}),
                'entities': parsed_data.get('entities', [])
            })
        )
        db.session.add(asset)
        db.session.commit()

        follow_up_questions = nlp_agent.generate_follow_up_questions(parsed_data)

        return jsonify({
            'success': True,
```

```
            'asset': asset.to_dict(),
            'parsed_data': parsed_data,
            'follow_up_questions': follow_up_questions,
            'next_steps': [
                'Review asset information',
                'Proceed with verification',
                'Submit for tokenization'
            ]
        })

    except Exception as e:
        logger.error(f"Asset intake failed: {str(e)}")
        return jsonify({'error': 'Internal server error', 'details': str(e)}), 500

@app.route('/api/verify/<int:asset_id>', methods=['POST'])
def verify_asset(asset_id):
    try:
        asset = Asset.query.get_or_404(asset_id)
        logger.info(f"Verifying asset: {asset_id}")

        asset_data = asset.to_dict()
        verification_result = verification_agent.verify_asset(asset_data)

        asset.verification_status = verification_result['status']
        asset.updated_at = datetime.utcnow()
        db.session.commit()

        transaction = Transaction(
            asset_id=asset.id,
            transaction_type='verification',
            status=verification_result['status'],
            details=json.dumps(verification_result)
        )
        db.session.add(transaction)
        db.session.commit()

        return jsonify({
            'success': True,
            'verification_result': verification_result,
            'asset': asset.to_dict()
        })

    except Exception as e:
        logger.error(f"Verification failed: {str(e)}")
        return jsonify({'error': 'Verification failed', 'details': str(e)}), 500

@app.route('/api/tokenize/<int:asset_id>', methods=['POST'])
def tokenize_asset(asset_id):
    try:
        asset = Asset.query.get_or_404(asset_id)

        if asset.verification_status != 'verified':
            return jsonify({'error': 'Asset must be verified before tokenization'}), 400

        asset_data = asset.to_dict()

        last_verification = Transaction.query.filter_by(
            asset_id=asset_id,
            transaction_type='verification'
        ).order_by(Transaction.created_at.desc()).first()
        verification_result = json.loads(last_verification.details) if last_verification else {'statu
```

```
        tokenization_result = tokenization_agent.tokenize_asset(asset_data, verification_result)

        if tokenization_result.get('success'):
            asset.token_id = tokenization_result['token_id']
            asset.updated_at = datetime.utcnow()
            db.session.commit()

            transaction = Transaction(
                asset_id=asset.id,
                transaction_type='tokenization',
                transaction_hash=tokenization_result['transaction_hash'],
                status='completed',
                details=json.dumps(tokenization_result)
            )
            db.session.add(transaction)
            db.session.commit()

            return jsonify({
                'success': True,
                'tokenization_result': tokenization_result,
                'asset': asset.to_dict()
            })
        else:
            return jsonify(tokenization_result), 400

    except Exception as e:
        logger.error(f"Tokenization failed: {str(e)}")
        return jsonify({'error': 'Tokenization failed', 'details': str(e)}), 500

@app.route('/api/asset/<int:asset_id>')
def get_asset(asset_id):
    try:
        asset = Asset.query.get_or_404(asset_id)
        transactions = Transaction.query.filter_by(asset_id=asset_id).order_by(Transaction.created_at
        return jsonify({
            'asset': asset.to_dict(),
            'transactions': [tx.to_dict() for tx in transactions]
        })

    except Exception as e:
        logger.error(f"Get asset failed: {str(e)}")
        return jsonify({'error': 'Asset not found', 'details': str(e)}), 404

@app.route('/api/assets/<wallet_address>')
def get_user_assets(wallet_address):
    try:
        user = User.query.filter_by(wallet_address=wallet_address).first()
        if not user:
            return jsonify({'assets': []})

        assets = Asset.query.filter_by(user_id=user.id).order_by(Asset.created_at.desc()).all()

        return jsonify({
            'user': user.to_dict(),
            'assets': [asset.to_dict() for asset in assets]
        })

    except Exception as e:
        logger.error(f"Get user assets failed: {str(e)}")
        return jsonify({'error': 'Failed to retrieve assets', 'details': str(e)}), 500
@app.route('/api/stats')
```

```
def get_stats():
    try:
        total_assets = Asset.query.count()
        total_users = User.query.count()
        verified_assets = Asset.query.filter_by(verification_status='verified').count()
        tokenized_assets = Asset.query.filter(Asset.token_id.isnot(None)).count()

        return jsonify({
            'total_assets': total_assets,
            'total_users': total_users,
            'verified_assets': verified_assets,
            'tokenized_assets': tokenized_assets,
            'verification_rate': (verified_assets / total_assets * 100) if total_assets > 0 else 0,
            'tokenization_rate': (tokenized_assets / verified_assets * 100) if verified_assets > 0 el
        })

    except Exception as e:
        logger.error(f"Stats failed: {str(e)}")
        return jsonify({'error': 'Failed to retrieve stats', 'details': str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

# File: app/agents/nlp_agent.py

```
import spacy
import re
from nltk.sentiment import SentimentIntensityAnalyzer
from typing import Dict, List, Optional

class NLPAgent:
    def __init__(self):
        self.nlp = spacy.load("en_core_web_sm")
        self.sentiment_analyzer = SentimentIntensityAnalyzer()

        # Asset type patterns
        self.asset_patterns = {
            'real_estate': ['house', 'apartment', 'property', 'building', 'land', 'condo', 'flat', 'v
            'vehicle': ['car', 'truck', 'motorcycle', 'boat', 'plane', 'vehicle', 'mileage', 'sedan',
            'artwork': ['painting', 'sculpture', 'art', 'artwork', 'canvas', 'oil painting', 'artist'
            'equipment': ['machinery', 'equipment', 'tool', 'device', 'machine', 'serial number', 'op
            'commodity': ['gold', 'silver', 'oil', 'wheat', 'commodity', 'metal', 'oz', 'purity']
        }

        # Value patterns
        self.value_patterns = [
            r'\$([0-9,]+(?:\.[0-9]{2})?)',
            r'([0-9,]+(?:\.[0-9]{2})?) dollars?',
            r'worth ([0-9,]+)',
            r'valued at ([0-9,]+)',
            r'inr[\s■]*([\d,]+)',
            r'(\d+(?:\.\d+)?)\s*(crore|crores|cr)',
            r'(\d+(?:\.\d+)?)\s*(lakh|lac|lacs)'
        ]

        # Location patterns
        self.location_patterns = [
            r'in ([A-Z][a-z]+(?: [A-Z][a-z]+)*)',
            r'located in ([A-Z][a-z]+(?: [A-Z][a-z]+)*)',
            r'at ([A-Z][a-z]+(?: [A-Z][a-z]+)*)'
        ]

    def parse_user_input(self, text: str) -> Dict:
        doc = self.nlp(text.lower())

        result = {
            'asset_type': self._extract_asset_type(text),
            'description': self._clean_description(text),
            'estimated_value': self._extract_value(text),
            'location': self._extract_location(text),
            'sentiment': self._analyze_sentiment(text),
            'entities': self._extract_entities(doc),
            'confidence_score': 0.0
        }

        result['confidence_score'] = self._calculate_confidence(result)

        return result

    def _extract_asset_type(self, text: str) -> Optional[str]:
        text_lower = text.lower()
        for asset_type, keywords in self.asset_patterns.items():
```

```
        for keyword in keywords:
            if keyword in text_lower:
                return asset_type
    return 'unknown'

def _extract_value(self, text: str) -> Optional[float]:
    text = text.lower()

    for pattern in self.value_patterns:
        match = re.search(pattern, text, re.IGNORECASE)
        if match:
            value_str = match.group(1).replace(',', '')
            try:
                value = float(value_str)
                # Apply crore / lakh scaling
                if len(match.groups()) >= 2:
                    unit = match.group(2)
                    if unit in ['crore', 'crores', 'cr']:
                        value *= 1e7
                    elif unit in ['lakh', 'lac', 'lacs']:
                        value *= 1e5
                return value
            except ValueError:
                continue

    return None

def _extract_location(self, text: str) -> Optional[str]:
    for pattern in self.location_patterns:
        match = re.search(pattern, text)
        if match:
            return match.group(1)
    return None

def _clean_description(self, text: str) -> str:
    return ' '.join(text.split())[:500]

def _analyze_sentiment(self, text: str) -> Dict:
    scores = self.sentiment_analyzer.polarity_scores(text)
    return {
        'compound': scores['compound'],
        'positive': scores['pos'],
        'negative': scores['neg'],
        'neutral': scores['neu']
    }

def _extract_entities(self, doc) -> List[Dict]:
    return [
        {
            'text': ent.text,
            'label': ent.label_,
            'description': spacy.explain(ent.label_)
        }
        for ent in doc.ents
    ]

def _calculate_confidence(self, result: Dict) -> float:
    score = 0.0

    if result['asset_type'] != 'unknown':
        score += 0.25
```

```
    if result['estimated_value'] is not None:
        score += 0.25

    if result['location'] is not None:
        score += 0.25

    if result['sentiment']['compound'] >= 0:
        score += 0.25

    return round(min(score, 1.0), 2)

def generate_follow_up_questions(self, parsed_data: Dict) -> List[str]:
    questions = []

    if parsed_data['asset_type'] == 'unknown':
        questions.append("What type of asset are you looking to tokenize?")

    if parsed_data['estimated_value'] is None:
        questions.append("What is the estimated value of your asset?")

    if parsed_data['location'] is None:
        questions.append("Where is the asset located?")

    if parsed_data['confidence_score'] < 0.7:
        questions.append("Could you provide more details about your asset?")

    if parsed_data['asset_type'] == 'real_estate':
        questions.append("Please provide size (sqft), bedrooms, year built.")

    if parsed_data['asset_type'] == 'vehicle':
        questions.append("Please provide year, model, mileage of the vehicle.")

    if parsed_data['asset_type'] == 'artwork':
        questions.append("Please provide artist name, dimensions, and medium.")

    return questions[:3]
```

# File: app/agents/verification_agent.py

```
import re
from typing import Dict, List
from datetime import datetime
import json

class VerificationAgent:
    def __init__(self):
        self.verification_threshold = 0.7

        # Jurisdictional support
        self.supported_jurisdictions = {
            'US': {'compliance_level': 'high', 'required_docs': ['title', 'appraisal']},
            'EU': {'compliance_level': 'high', 'required_docs': ['ownership', 'certificate']},
            'UK': {'compliance_level': 'medium', 'required_docs': ['deed', 'valuation']},
            'CA': {'compliance_level': 'medium', 'required_docs': ['title', 'assessment']},
            'SG': {'compliance_level': 'high', 'required_docs': ['certificate', 'valuation']}
        }

        # Asset value ranges for validation
        self.value_ranges = {
            'real_estate': {'min': 10000, 'max': 50000000},
            'vehicle': {'min': 1000, 'max': 2000000},
            'artwork': {'min': 500, 'max': 100000000},
            'equipment': {'min': 100, 'max': 5000000},
            'commodity': {'min': 50, 'max': 10000000}
        }

    def verify_asset(self, asset_data: Dict) -> Dict:
        """Comprehensive asset verification"""
        verification_result = {
            'overall_score': 0.0,
            'status': 'pending',
            'breakdown': {},
            'issues': [],
            'recommendations': [],
            'next_steps': []
        }

        try:
            # Basic validation
            basic_score = self._verify_basic_information(asset_data)
            verification_result['breakdown']['basic_info'] = basic_score

            # Value assessment
            value_score = self._verify_value(asset_data)
            verification_result['breakdown']['value_assessment'] = value_score

            # Compliance check
            compliance_score = self._verify_compliance(asset_data)
            verification_result['breakdown']['compliance'] = compliance_score

            # Asset-specific verification
            specific_score = self._verify_asset_specific(asset_data)
            verification_result['breakdown']['asset_specific'] = specific_score

            # Calculate overall score
            scores = [basic_score, value_score, compliance_score, specific_score]
```

```
            verification_result['overall_score'] = sum(scores) / len(scores)

            # Determine status
            if verification_result['overall_score'] >= self.verification_threshold:
                verification_result['status'] = 'verified'
            elif verification_result['overall_score'] >= 0.5:
                verification_result['status'] = 'requires_review'
            else:
                verification_result['status'] = 'rejected'

            # Generate recommendations
            verification_result['recommendations'] = self._generate_recommendations(
                asset_data, verification_result
            )

            # Define next steps
            verification_result['next_steps'] = self._define_next_steps(
                verification_result['status'], asset_data
            )

        except Exception as e:
            verification_result['status'] = 'error'
            verification_result['issues'].append(f"Verification error: {str(e)}")

        return verification_result

    def _verify_basic_information(self, asset_data: Dict) -> float:
        """Verify basic asset information completeness"""
        score = 0.0
        required_fields = ['asset_type', 'description', 'estimated_value', 'location']

        for field in required_fields:
            if field in asset_data and asset_data[field]:
                if field == 'description' and len(str(asset_data[field])) >= 10:
                    score += 0.25
                elif field == 'estimated_value' and asset_data[field] > 0:
                    score += 0.25
                elif field in ['asset_type', 'location'] and len(str(asset_data[field])) >= 2:
                    score += 0.25

        return min(score, 1.0)

    def _verify_value(self, asset_data: Dict) -> float:
        """Verify asset value reasonableness"""
        if 'estimated_value' not in asset_data or not asset_data['estimated_value']:
            return 0.0

        value = asset_data['estimated_value']
        asset_type = asset_data.get('asset_type', 'unknown')

        if asset_type in self.value_ranges:
            range_info = self.value_ranges[asset_type]
            if range_info['min'] <= value <= range_info['max']:
                return 1.0
            elif value < range_info['min']:
                return 0.3  # Too low
            else:
                return 0.6  # Too high, needs extra verification

        return 0.5  # Unknown asset type
    def _verify_compliance(self, asset_data: Dict) -> float:
```

```
    """Verify regulatory compliance requirements"""
    jurisdiction = self._extract_jurisdiction(asset_data.get('location', ''))

    if jurisdiction in self.supported_jurisdictions:
        return 0.9
    elif jurisdiction:
        return 0.5  # Partial support
    else:
        return 0.3  # Unknown jurisdiction

def _verify_asset_specific(self, asset_data: Dict) -> float:
    """Asset-type specific verification"""
    asset_type = asset_data.get('asset_type', 'unknown')

    if asset_type == 'real_estate':
        return self._verify_real_estate(asset_data)
    elif asset_type == 'vehicle':
        return self._verify_vehicle(asset_data)
    elif asset_type == 'artwork':
        return self._verify_artwork(asset_data)
    elif asset_type == 'equipment':
        return self._verify_equipment(asset_data)
    elif asset_type == 'commodity':
        return self._verify_commodity(asset_data)
    else:
        return 0.4  # Unknown type gets lower score

def _verify_real_estate(self, asset_data: Dict) -> float:
    """Real estate specific verification"""
    score = 0.5  # Base score
    description = asset_data.get('description', '').lower()

    # Look for property indicators
    property_indicators = ['sqft', 'bedroom', 'bathroom', 'acre', 'floor', 'apartment']
    for indicator in property_indicators:
        if indicator in description:
            score += 0.1

    return min(score, 1.0)

def _verify_vehicle(self, asset_data: Dict) -> float:
    """Vehicle specific verification"""
    score = 0.5
    description = asset_data.get('description', '').lower()

    # Look for vehicle indicators
    vehicle_indicators = ['year', 'model', 'make', 'mileage', 'engine', 'transmission']
    for indicator in vehicle_indicators:
        if indicator in description:
            score += 0.1

    return min(score, 1.0)

def _verify_artwork(self, asset_data: Dict) -> float:
    """Artwork specific verification"""
    score = 0.5
    description = asset_data.get('description', '').lower()

    # Look for art indicators
    art_indicators = ['artist', 'canvas', 'oil', 'watercolor', 'sculpture', 'signed']
    for indicator in art_indicators:
```

```
            if indicator in description:
                score += 0.1

        return min(score, 1.0)

    def _verify_equipment(self, asset_data: Dict) -> float:
        """Equipment specific verification"""
        score = 0.5
        description = asset_data.get('description', '').lower()

        # Look for equipment indicators
        equipment_indicators = ['serial', 'model', 'manufacturer', 'warranty', 'condition']
        for indicator in equipment_indicators:
            if indicator in description:
                score += 0.1

        return min(score, 1.0)

    def _verify_commodity(self, asset_data: Dict) -> float:
        """Commodity specific verification"""
        score = 0.5
        description = asset_data.get('description', '').lower()

        # Look for commodity indicators
        commodity_indicators = ['grade', 'purity', 'weight', 'certificate', 'assay', 'quality']
        for indicator in commodity_indicators:
            if indicator in description:
                score += 0.1

        return min(score, 1.0)

    def _extract_jurisdiction(self, location: str) -> str:
        """Extract jurisdiction from location string"""
        if not location:
            return ''

        location_upper = location.upper()

        # Simple jurisdiction mapping
        jurisdiction_mappings = {
            'US': ['USA', 'UNITED STATES', 'AMERICA', 'NEW YORK', 'CALIFORNIA', 'TEXAS'],
            'UK': ['UNITED KINGDOM', 'ENGLAND', 'SCOTLAND', 'WALES', 'LONDON'],
            'CA': ['CANADA', 'TORONTO', 'VANCOUVER', 'MONTREAL'],
            'EU': ['GERMANY', 'FRANCE', 'SPAIN', 'ITALY', 'NETHERLANDS'],
            'SG': ['SINGAPORE']
        }

        for jurisdiction, keywords in jurisdiction_mappings.items():
            for keyword in keywords:
                if keyword in location_upper:
                    return jurisdiction

        return 'OTHER'

    def _generate_recommendations(self, asset_data: Dict, verification_result: Dict) -> List[str]:
        """Generate recommendations based on verification results"""
        recommendations = []

        if verification_result['breakdown']['basic_info'] < 0.8:
            recommendations.append("Provide more detailed asset description")
        if verification_result['breakdown']['value_assessment'] < 0.8:
```

```
            recommendations.append("Consider professional appraisal for accurate valuation")

        if verification_result['breakdown']['compliance'] < 0.8:
            recommendations.append("Verify jurisdiction-specific compliance requirements")

        if verification_result['breakdown']['asset_specific'] < 0.8:
            asset_type = asset_data.get('asset_type', 'unknown')
            if asset_type == 'real_estate':
                recommendations.append("Obtain property deeds and recent appraisal")
            elif asset_type == 'vehicle':
                recommendations.append("Provide vehicle title and registration documents")
            elif asset_type == 'artwork':
                recommendations.append("Obtain authenticity certificate and professional appraisal")

        return recommendations

    def _define_next_steps(self, status: str, asset_data: Dict) -> List[str]:
        """Define next steps based on verification status"""
        if status == 'verified':
            return [
                "Asset ready for tokenization",
                "Prepare smart contract deployment",
                "Set up marketplace listing"
            ]
        elif status == 'requires_review':
            return [
                "Submit additional documentation",
                "Schedule manual review",
                "Address verification concerns"
            ]
        else:  # rejected or error
            return [
                "Review asset information",
                "Provide missing documentation",
                "Contact support for assistance"
            ]
```

# File: app/agents/tokenization_agent.py

```
import hashlib
import json
import time
from datetime import datetime
from typing import Dict, Optional
import uuid

class TokenizationAgent:
    def __init__(self):
        self.token_standard = "RWA-721"  # Mock token standard
        self.network = "RWA-TestNet"  # Mock blockchain network

    def tokenize_asset(self, asset_data: Dict, verification_result: Dict) -> Dict:
        """Create a tokenized representation of the asset"""
        if verification_result.get('status') != 'verified':
            return {
                'success': False,
                'error': 'Asset must be verified before tokenization',
                'status': 'failed'
            }

        try:
            # Generate token metadata
            token_metadata = self._generate_token_metadata(asset_data)

            # Create mock smart contract
            contract_data = self._create_mock_contract(asset_data, token_metadata)

            # Generate token ID
            token_id = self._generate_token_id(asset_data)

            # Create transaction record
            transaction_hash = self._generate_transaction_hash(contract_data)

            tokenization_result = {
                'success': True,
                'token_id': token_id,
                'contract_address': contract_data['address'],
                'transaction_hash': transaction_hash,
                'metadata': token_metadata,
                'network': self.network,
                'standard': self.token_standard,
                'created_at': datetime.utcnow().isoformat(),
                'status': 'minted'
            }

            return tokenization_result

        except Exception as e:
            return {
                'success': False,
                'error': f'Tokenization failed: {str(e)}',
                'status': 'failed'
            }

    def _generate_token_metadata(self, asset_data: Dict) -> Dict:
        """Generate NFT-style metadata for the asset"""
```

```
        return {
            'name': f"RWA Token - {asset_data.get('asset_type', 'Asset').title()}",
            'description': asset_data.get('description', 'Real World Asset Token'),
            'image': f"https://placeholder.com/400x400?text={asset_data.get('asset_type', 'asset')}",
            'external_url': f"https://rwa-marketplace.com/asset/{asset_data.get('id', 'unknown')}",
            'attributes': [
                {
                    'trait_type': 'Asset Type',
                    'value': asset_data.get('asset_type', 'unknown').title()
                },
                {
                    'trait_type': 'Estimated Value',
                    'value': f"${asset_data.get('estimated_value', 0):,.2f}"
                },
                {
                    'trait_type': 'Location',
                    'value': asset_data.get('location', 'Unknown')
                },
                {
                    'trait_type': 'Verification Status',
                    'value': 'Verified'
                },
                {
                    'trait_type': 'Token Standard',
                    'value': self.token_standard
                },
                {
                    'trait_type': 'Network',
                    'value': self.network
                },
                {
                    'trait_type': 'Tokenization Date',
                    'value': datetime.utcnow().strftime('%Y-%m-%d')
                }
            ],
            'properties': {
                'category': 'Real World Asset',
                'subcategory': asset_data.get('asset_type', 'unknown'),
                'fractional': False,  # For POC, we'll keep it simple
                'transferable': True
            }
        }

    def _create_mock_contract(self, asset_data: Dict, metadata: Dict) -> Dict:
        """Create a mock smart contract representation"""
        contract_address = self._generate_contract_address(asset_data)

        contract_data = {
            'address': contract_address,
            'abi': self._get_mock_abi(),
            'bytecode': self._generate_mock_bytecode(asset_data),
            'constructor_args': {
                'name': metadata['name'],
                'symbol': 'RWA',
                'baseURI': 'https://api.rwa-tokenization.com/metadata/'
            },
            'functions': {
                'tokenURI': f'https://api.rwa-tokenization.com/metadata/{contract_address}',
                'ownerOf': asset_data.get('user_id', 'unknown'),
                'approve': 'function approve(address to, uint256 tokenId)',
                'transfer': 'function transfer(address to, uint256 tokenId)'
```

```
        },
        'events': [
            {
                'name': 'Transfer',
                'signature': 'Transfer(address indexed from, address indexed to, uint256 indexed
            },
            {
                'name': 'AssetTokenized',
                'signature': 'AssetTokenized(uint256 indexed tokenId, address indexed owner, stri
            }
        ]
    }

    return contract_data

def _generate_token_id(self, asset_data: Dict) -> str:
    """Generate a unique token ID"""
    # Create deterministic but unique token ID
    content = f"{asset_data.get('id', 'unknown')}_{asset_data.get('asset_type', 'asset')}_{int(ti
    token_hash = hashlib.sha256(content.encode()).hexdigest()
    return f"RWA_{token_hash[:16].upper()}"

def _generate_contract_address(self, asset_data: Dict) -> str:
    """Generate a mock contract address"""
    content = f"contract_{asset_data.get('asset_type', 'unknown')}_{uuid.uuid4()}"
    address_hash = hashlib.sha256(content.encode()).hexdigest()
    return f"0x{address_hash[:40]}"

def _generate_transaction_hash(self, contract_data: Dict) -> str:
    """Generate a mock transaction hash"""
    content = f"tx_{contract_data['address']}_{int(time.time())}"
    tx_hash = hashlib.sha256(content.encode()).hexdigest()
    return f"0x{tx_hash}"

def _generate_mock_bytecode(self, asset_data: Dict) -> str:
    """Generate mock contract bytecode"""
    # This is just for demonstration - real bytecode would be much longer
    content = f"bytecode_{asset_data.get('asset_type', 'unknown')}"
    bytecode_hash = hashlib.sha256(content.encode()).hexdigest()
    return f"0x{bytecode_hash}"

def _get_mock_abi(self) -> list:
    """Return mock ABI for the contract"""
    return [
        {
            "inputs": [
                {"name": "to", "type": "address"},
                {"name": "tokenId", "type": "uint256"}
            ],
            "name": "approve",
            "outputs": [],
            "type": "function"
        },
        {
            "inputs": [
                {"name": "tokenId", "type": "uint256"}
            ],
            "name": "tokenURI",
            "outputs": [
                {"name": "", "type": "string"}
            ],
```

```
            "type": "function"
        },
        {
            "inputs": [
                {"name": "tokenId", "type": "uint256"}
            ],
            "name": "ownerOf",
            "outputs": [
                {"name": "", "type": "address"}
            ],
            "type": "function"
        }
    ]

def verify_token_ownership(self, token_id: str, wallet_address: str) -> bool:
    """Verify token ownership (mock implementation)"""
    # In a real implementation, this would query the blockchain
    return True  # Mock verification always passes

def transfer_token(self, token_id: str, from_address: str, to_address: str) -> Dict:
    """Transfer token between addresses (mock implementation)"""
    transaction_hash = hashlib.sha256(
        f"transfer_{token_id}_{from_address}_{to_address}_{int(time.time())}".encode()
    ).hexdigest()

    return {
        'success': True,
        'transaction_hash': f"0x{transaction_hash}",
        'from_address': from_address,
        'to_address': to_address,
        'token_id': token_id,
        'timestamp': datetime.utcnow().isoformat()
    }
```

# File: app/models/database.py

```
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
import json

db = SQLAlchemy()

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    wallet_address = db.Column(db.String(42), unique=True, nullable=False)
    email = db.Column(db.String(120), nullable=True)
    kyc_status = db.Column(db.String(20), default='pending')
    jurisdiction = db.Column(db.String(10), nullable=True)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    def to_dict(self):
        return {
            'id': self.id,
            'wallet_address': self.wallet_address,
            'email': self.email,
            'kyc_status': self.kyc_status,
            'jurisdiction': self.jurisdiction,
            'created_at': self.created_at.isoformat()
        }

class Asset(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    asset_type = db.Column(db.String(50), nullable=False)
    description = db.Column(db.Text, nullable=False)
    estimated_value = db.Column(db.Float, nullable=False)
    location = db.Column(db.String(200), nullable=False)
    verification_status = db.Column(db.String(20), default='pending')
    token_id = db.Column(db.String(100), nullable=True)
    requirements = db.Column(db.Text, nullable=True)  # JSON string
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)

    user = db.relationship('User', backref=db.backref('assets', lazy=True))

    def to_dict(self):
        return {
            'id': self.id,
            'user_id': self.user_id,
            'asset_type': self.asset_type,
            'description': self.description,
            'estimated_value': self.estimated_value,
            'location': self.location,
            'verification_status': self.verification_status,
            'token_id': self.token_id,
            'requirements': json.loads(self.requirements) if self.requirements else {},
            'created_at': self.created_at.isoformat(),
            'updated_at': self.updated_at.isoformat()
        }

class Transaction(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    asset_id = db.Column(db.Integer, db.ForeignKey('asset.id'), nullable=False)
```

```
transaction_type = db.Column(db.String(50), nullable=False)  # tokenize, transfer, etc.
transaction_hash = db.Column(db.String(100), nullable=True)
status = db.Column(db.String(20), default='pending')
details = db.Column(db.Text, nullable=True)  # JSON string
created_at = db.Column(db.DateTime, default=datetime.utcnow)

asset = db.relationship('Asset', backref=db.backref('transactions', lazy=True))

def to_dict(self):
    return {
        'id': self.id,
        'asset_id': self.asset_id,
        'transaction_type': self.transaction_type,
        'transaction_hash': self.transaction_hash,
        'status': self.status,
        'details': json.loads(self.details) if self.details else {},
        'created_at': self.created_at.isoformat()
    }
```

## File: static/js/app.js

```
class RWAApp {
    constructor() {
        this.baseURL = window.location.origin;
        this.currentWallet = null;
        this.currentAssets = [];
        this.currentAsset = null;
        this.init();
    }

    init() {
        this.setupEventListeners();
        this.loadStats();
        this.loadSampleWallet();
    }

    setupEventListeners() {
        // Asset form submission
        document.getElementById('asset-form').addEventListener('submit', (e) => {
            e.preventDefault();
            this.submitAsset();
        });

        // Refresh assets button
        document.getElementById('refresh-assets').addEventListener('click', () => {
            this.loadUserAssets();
        });

        // Modal action buttons
        document.getElementById('verify-btn').addEventListener('click', () => {
            this.verifyAsset();
        });

        document.getElementById('tokenize-btn').addEventListener('click', () => {
            this.tokenizeAsset();
        });
    }

    loadSampleWallet() {
        // Load a sample wallet for demo purposes
        const sampleWallet = '0x742d35Cc6e34d8d7C15fE14c123456789abcdef0';
        document.getElementById('wallet-address').value = sampleWallet;
        this.currentWallet = sampleWallet;
        document.getElementById('wallet-display').textContent = `${sampleWallet.substr(0, 6)}...${sam
        this.loadUserAssets();
    }

    async submitAsset() {
        const submitBtn = document.getElementById('submit-btn');
        const spinner = document.getElementById('submit-spinner');

        try {
            this.setLoading(submitBtn, spinner, true);

            const walletAddress = document.getElementById('wallet-address').value;
            const assetDescription = document.getElementById('asset-description').value;
            const email = document.getElementById('email').value;
            const response = await fetch(`${this.baseURL}/api/intake`, {
```

```
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                wallet_address: walletAddress,
                user_input: assetDescription,
                email: email
            })
        });

        const result = await response.json();

        if (result.success) {
            this.showAlert('success', 'Asset submitted successfully!');
            this.showFollowUpQuestions(result.follow_up_questions);
            this.resetForm();
            this.loadUserAssets();
            this.loadStats();
        } else {
            this.showAlert('danger', `Error: ${result.error}`);
        }
    } catch (error) {
        console.error('Error submitting asset:', error);
        this.showAlert('danger', 'Failed to submit asset. Please try again.');
    } finally {
        this.setLoading(submitBtn, spinner, false);
    }
}

async loadUserAssets() {
    const walletAddress = document.getElementById('wallet-address').value;
    if (!walletAddress) return;

    try {
        const response = await fetch(`${this.baseURL}/api/assets/${walletAddress}`);
        const result = await response.json();

        this.currentAssets = result.assets || [];
        this.renderAssets(this.currentAssets);
    } catch (error) {
        console.error('Error loading assets:', error);
    }
}

async loadStats() {
    try {
        const response = await fetch(`${this.baseURL}/api/stats`);
        const stats = await response.json();

        document.getElementById('total-assets').textContent = stats.total_assets || 0;
        document.getElementById('verified-assets').textContent = stats.verified_assets || 0;
        document.getElementById('tokenized-assets').textContent = stats.tokenized_assets || 0;
        document.getElementById('total-users').textContent = stats.total_users || 0;
    } catch (error) {
        console.error('Error loading stats:', error);
    }
}

renderAssets(assets) {
    const assetsList = document.getElementById('assets-list');
```

```
    if (assets.length === 0) {
        assetsList.innerHTML = `
            <div class="text-center text-muted">
                <p>No assets found. Submit your first asset above!</p>
            </div>
        `;
        return;
    }

    assetsList.innerHTML = assets.map(asset => `
        <div class="card mb-2">
            <div class="card-body">
                <div class="d-flex justify-content-between align-items-start">
                    <div>
                        <h6 class="card-title">${this.getAssetTypeIcon(asset.asset_type)} ${asset
                        <p class="card-text small">${asset.description.substring(0, 80)}...</p>
                        <div class="d-flex gap-2">
                            <span class="badge bg-secondary">${asset.estimated_value?.toLocaleStr

                            ${asset.token_id ? '<span class="badge bg-success">Tokenized</span>'
                        </div>
                    </div>
                    <button class="btn btn-outline-primary btn-sm" onclick="app.showAssetDetails(
                        View
                    </button>
                </div>
            </div>
        </div>
    `).join('');
}

async showAssetDetails(assetId) {
    try {
        const response = await fetch(`${this.baseURL}/api/asset/${assetId}`);
        const result = await response.json();
        const asset = result.asset;
        const transactions = result.transactions || [];

        // Store current asset for modal actions
        this.currentAsset = asset;

        // Populate modal
        document.getElementById('asset-modal-body').innerHTML = `
            <div class="row">
                <div class="col-md-6">
                    <h6>Asset Information</h6>
                    <table class="table table-sm">
                        <tr><td><strong>Type:</strong></td><td>${asset.asset_type.replace('_', '
                        <tr><td><strong>Value:</strong></td><td>${asset.estimated_value?.toLocale

                        <tr><td><strong>Location:</strong></td><td>${asset.location}</td></tr>
                        <tr><td><strong>Status:</strong></td><td><span class="badge bg-${this.get
                        <tr><td><strong>Created:</strong></td><td>${new Date(asset.created_at).to
                        ${asset.token_id ? `<tr><td><strong>Token ID:</strong></td><td><code>${as
                    </table>
                    <h6>Description</h6>
                    <p class="small">${asset.description}</p>
                </div>
                <div class="col-md-6">
                    <h6>Transaction History</h6>
                    ${transactions.length > 0 ? `
```

```
                        <div class="list-group">
                            ${transactions.map(tx => `
                                <div class="list-group-item">
                                    <div class="d-flex w-100 justify-content-between">
                                        <h6 class="mb-1">${tx.transaction_type}</h6>
                                        <small>${new Date(tx.created_at).toLocaleDateString()}</s
                                    </div>
                                    <p class="mb-1"><span class="badge bg-${this.getStatusColor(t
                                    ${tx.transaction_hash ? `<small>Hash: <code>${tx.transaction_
                                </div>
                            `).join('')}
                        </div>
                    ` : '<p class="text-muted">No transactions yet</p>'}
                </div>
            </div>
        `;

        // Show appropriate action buttons
        document.getElementById('verify-btn').classList.toggle('d-none', asset.verification_statu
        document.getElementById('tokenize-btn').classList.toggle('d-none', asset.verification_sta

        // Show modal
        new bootstrap.Modal(document.getElementById('asset-modal')).show();

    } catch (error) {
        console.error('Error loading asset details:', error);
        this.showAlert('danger', 'Failed to load asset details');
    }
}

async verifyAsset() {
    if (!this.currentAsset) return;

    try {
        const response = await fetch(`${this.baseURL}/api/verify/${this.currentAsset.id}`, {
            method: 'POST'
        });

        const result = await response.json();

        if (result.success) {
            this.showAlert('success', 'Asset verification completed!');
            this.loadUserAssets();
            this.loadStats();

            // Close modal and show results
            bootstrap.Modal.getInstance(document.getElementById('asset-modal')).hide();

            // Show verification details
            this.showVerificationResults(result.verification_result);
        } else {
            this.showAlert('danger', `Verification failed: ${result.error}`);
        }
    } catch (error) {
        console.error('Error verifying asset:', error);
        this.showAlert('danger', 'Failed to verify asset');
    }
}

async tokenizeAsset() {
    if (!this.currentAsset) return;
```

```
    try {
        const response = await fetch(`${this.baseURL}/api/tokenize/${this.currentAsset.id}`, {
            method: 'POST'
        });

        const result = await response.json();

        if (result.success) {
            this.showAlert('success', 'Asset tokenized successfully!');
            this.loadUserAssets();
            this.loadStats();

            // Close modal and show results
            bootstrap.Modal.getInstance(document.getElementById('asset-modal')).hide();

            // Show tokenization details
            this.showTokenizationResults(result.tokenization_result);
        } else {
            this.showAlert('danger', `Tokenization failed: ${result.error}`);
        }
    } catch (error) {
        console.error('Error tokenizing asset:', error);
        this.showAlert('danger', 'Failed to tokenize asset');
    }
}

showVerificationResults(verificationResult) {
    const alertHtml = `
        <div class="alert alert-info alert-dismissible fade show" role="alert">
            <h6>■ Verification Results</h6>
            <p><strong>Overall Score:</strong> ${(verificationResult.overall_score * 100).toFixed
            <p><strong>Status:</strong> <span class="badge bg-${this.getStatusColor(verificationR
            ${verificationResult.recommendations.length > 0 ? `
                <p><strong>Recommendations:</strong></p>
                <ul>${verificationResult.recommendations.map(rec => `<li>${rec}</li>`).join('')}<
            ` : ''}
            <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
        </div>
    `;
    document.getElementById('alerts').innerHTML = alertHtml;
}

showTokenizationResults(tokenizationResult) {
    const alertHtml = `
        <div class="alert alert-success alert-dismissible fade show" role="alert">
            <h6>■ Tokenization Successful!</h6>
            <p><strong>Token ID:</strong> <code>${tokenizationResult.token_id}</code></p>
            <p><strong>Contract:</strong> <code>${tokenizationResult.contract_address}</code></p>
            <p><strong>Transaction:</strong> <code>${tokenizationResult.transaction_hash}</code><
            <p><strong>Network:</strong> ${tokenizationResult.network}</p>
            <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
        </div>
    `;
    document.getElementById('alerts').innerHTML = alertHtml;
}

showFollowUpQuestions(questions) {
    if (questions.length === 0) return;

    const questionsHtml = questions.map(question => `
        <div class="alert alert-info mb-2">
```

```
            <small><strong>■ ${question}</strong></small>
        </div>
    `).join('');

    document.getElementById('follow-up-questions').innerHTML = questionsHtml;
    document.getElementById('follow-up-section').classList.remove('d-none');

    // Hide after 10 seconds
    setTimeout(() => {
        document.getElementById('follow-up-section').classList.add('d-none');
    }, 10000);
}

showAlert(type, message) {
    const alertHtml = `
        <div class="alert alert-${type} alert-dismissible fade show" role="alert">
            ${message}
            <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
        </div>
    `;
    document.getElementById('alerts').innerHTML = alertHtml;

    // Auto-dismiss after 5 seconds
    setTimeout(() => {
        const alertElement = document.querySelector('.alert');
        if (alertElement) {
            const alert = new bootstrap.Alert(alertElement);
            alert.close();
        }
    }, 5000);
}

resetForm() {
    document.getElementById('asset-description').value = '';
    document.getElementById('email').value = '';
}

setLoading(button, spinner, isLoading) {
    if (isLoading) {
        button.disabled = true;
        spinner.classList.remove('d-none');
        button.textContent = 'Processing...';
    } else {
        button.disabled = false;
        spinner.classList.add('d-none');
        button.textContent = 'Submit Asset';
    }
}

getStatusColor(status) {
    const colors = {
        'pending': 'warning',
        'verified': 'success',
        'rejected': 'danger',
        'requires_review': 'info',
        'completed': 'success',
        'failed': 'danger'
    };
    return colors[status] || 'secondary';
}
getAssetTypeIcon(assetType) {
```

```
        const icons = {
            'real_estate': '■',
            'vehicle': '■',
            'artwork': '■',
            'equipment': '■■',
            'commodity': '■',
            'unknown': '■'
        };
        return icons[assetType] || '■';
    }
}

// Initialize the app when DOM is loaded
document.addEventListener('DOMContentLoaded', function() {
    window.app = new RWAApp();
});
```

# ■ File: templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>RWA Tokenization POC</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesh
    <link href="{{ url_for('static', filename='css/style.css') }}" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
        <div class="container">
            <a class="navbar-brand" href="#">■ RWA Tokenization</a>
            <div class="navbar-nav ms-auto">
                <span class="navbar-text" id="wallet-display">Not Connected</span>
            </div>
        </div>
    </nav>

    <div class="container mt-4">
        <!-- Alert Section -->
        <div id="alerts"></div>

        <!-- Stats Section -->
        <div class="row mb-4">
            <div class="col-md-3">
                <div class="card text-center">
                    <div class="card-body">
                        <h5 class="card-title">Total Assets</h5>
                        <h2 class="text-primary" id="total-assets">0</h2>
                    </div>
                </div>
            </div>
            <div class="col-md-3">
                <div class="card text-center">
                    <div class="card-body">
                        <h5 class="card-title">Verified Assets</h5>
                        <h2 class="text-success" id="verified-assets">0</h2>
                    </div>
                </div>
            </div>
            <div class="col-md-3">
                <div class="card text-center">
                    <div class="card-body">
                        <h5 class="card-title">Tokenized Assets</h5>
                        <h2 class="text-info" id="tokenized-assets">0</h2>
                    </div>
                </div>
            </div>
            <div class="col-md-3">
                <div class="card text-center">
                    <div class="card-body">
                        <h5 class="card-title">Total Users</h5>
                        <h2 class="text-warning" id="total-users">0</h2>
                    </div>
                </div>
            </div>
```

```
</div>

<!-- Main Content -->
<div class="row">
    <!-- Asset Submission Form -->
    <div class="col-md-6">
        <div class="card">
            <div class="card-header">
                <h5>■ Submit Asset for Tokenization</h5>
            </div>
            <div class="card-body">
                <form id="asset-form">
                    <div class="mb-3">
                        <label for="wallet-address" class="form-label">Wallet Address</label>
                        <input type="text" class="form-control" id="wallet-address"
                               placeholder="0x..." required>
                        <div class="form-text">Your blockchain wallet address</div>
                    </div>
                    <div class="mb-3">
                        <label for="asset-description" class="form-label">Asset Description</
                        <textarea class="form-control" id="asset-description" rows="4"
                                  placeholder="Describe your asset in detail. For example: 'I
                                  required></textarea>
                        <div class="form-text">Provide detailed information about your asset<
                    </div>
                    <div class="mb-3">
                        <label for="email" class="form-label">Email (Optional)</label>
                        <input type="email" class="form-control" id="email"
                               placeholder="your@email.com">
                    </div>
                    <button type="submit" class="btn btn-primary w-100" id="submit-btn">
                        <span class="spinner-border spinner-border-sm d-none me-2" id="submit
                        Submit Asset
                    </button>
                </form>
            </div>
        </div>

        <!-- Follow-up Questions -->
        <div class="card mt-3 d-none" id="follow-up-section">
            <div class="card-header">
                <h6>■ Follow-up Questions</h6>
            </div>
            <div class="card-body" id="follow-up-questions">
            </div>
        </div>
    </div>

    <!-- Asset Management -->
    <div class="col-md-6">
        <div class="card">
            <div class="card-header d-flex justify-content-between align-items-center">
                <h5>■ Your Assets</h5>
                <button class="btn btn-outline-primary btn-sm" id="refresh-assets">
                    ■ Refresh
                </button>
            </div>
            <div class="card-body">
                <div id="assets-list">
                    <div class="text-center text-muted">
                        <p>No assets found. Submit your first asset above!</p>
```

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

    <!-- Asset Details Modal -->
    <div class="modal fade" id="asset-modal" tabindex="-1">
        <div class="modal-dialog modal-lg">
            <div class="modal-content">
                <div class="modal-header">
                    <h5 class="modal-title">Asset Details</h5>
                    <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
                </div>
                <div class="modal-body" id="asset-modal-body">
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</bu
                    <button type="button" class="btn btn-primary d-none" id="verify-btn">Verify Asset
                    <button type="button" class="btn btn-success d-none" id="tokenize-btn">Tokenize A
                </div>
            </div>
        </div>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></scri
    <script src="{{ url_for('static', filename='js/app.js') }}"></script>
</body>
</html>
```