RWA Tokenization POC - Step-by-Step Implementation Manual

Table of Contents

Pre-Implementation Checklist

System Requirements Verification

- [ ] **Operating System**: Windows 10+, macOS 10.15+, or Ubuntu 20.04+
- [ ] **Python**: Version 3.8+ installed and accessible via `python3` command
- [ ] **pip**: Python package installer available
- [ ] **Git**: Version control system installed (optional but recommended)
- [ ] **Text Editor**: VS Code, PyCharm, or similar IDE
- [ ] **Browser**: Modern browser (Chrome, Firefox, Safari, Edge)
- [ ] **Terminal/Command Prompt**: Access to command line interface

Check Python Installation

```bash
# Verify Python version (should be 3.8+)
python3 --version

# Verify pip is available
pip3 --version

# Check if virtual environment module is available
python3 -m venv --help
```

Environment Setup

Step 1: Create Project Directory

```bash
# Create main project directory
mkdir rwa-tokenization-poc
cd rwa-tokenization-poc

# Create subdirectories
mkdir -p app/{models,agents,utils}
mkdir -p {static/{css,js},templates,data,logs,uploads,tests}
```

Expected structure:

```
rwa-tokenization-poc/
├── app/
│   ├── models/
│   ├── agents/
│   └── utils/
├── static/
│   ├── css/
│   └── js/
├── templates/
├── data/
├── logs/
├── uploads/
└── tests/
```

Step 2: Create Virtual Environment

```bash
# Create virtual environment
python3 -m venv venv

# Activate virtual environment
# On Linux/macOS:
source venv/bin/activate

# On Windows:
venv\Scripts\activate

# Verify activation (should show venv in prompt)
which python  # Should point to venv/bin/python
```

Step 3: Install Dependencies

```bash
# Upgrade pip first
pip install --upgrade pip

# Create requirements.txt (copy from provided code)
# Then install dependencies
pip install -r requirements.txt

# Download spaCy language model
python -m spacy download en_core_web_sm

# Download NLTK data
python -c "
import nltk
import ssl
try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('vader_lexicon')
print('NLTK data downloaded successfully!')
"
```

File Structure Creation

Step 3: Create Configuration Files

Create `.env` file:

bash

```bash
# Create environment configuration
cat > .env << 'EOL'
SECRET_KEY=your-secret-key-change-this-in-production
FLASK_ENV=development
PORT=5000
DATABASE_URL=sqlite:///rwa_tokenization.db
LOG_LEVEL=INFO
LOG_FILE=logs/app.log
MAX_CONTENT_LENGTH=16777216
UPLOAD_FOLDER=uploads
EOL
```

Create `.gitignore` file:

bash

```bash
cat > .gitignore << 'EOL'
# Python
__pycache__/
*.py[cod]
*$py.class
*.so
.Python
venv/
ENV/
env/

# Database
*.db
*.sqlite3

# Logs
logs/
*.log

# Uploads
uploads/

# IDE
.vscode/
.idea/
*.swp
*.swo

# OS
.DS_Store
Thumbs.db

# Environment
.env
EOL
```

Backend Implementation

Step 4: Database Models

Create `app/models/__init__.py`:

```python
# Empty file to make it a Python package
```

Create `app/models/database.py`: Copy the complete database models code from the provided components.

Step 5: Agent System Implementation

Create `app/agents/__init__.py`:

```python
# Empty file to make it a Python package
```

Create each agent file:

1. `app/agents/nlp_agent.py` - Copy NLP Agent code
2. `app/agents/verification_agent.py` - Copy Verification Agent code
3. `app/agents/tokenization_agent.py` - Copy Tokenization Agent code

Step 6: Flask Application

Create `app/__init__.py`:

```python
# Empty file to make it a Python package
```

Create `app/main.py`: Copy the complete Flask application code.

Step 7: Configuration File

Create `config.py` in root directory: Copy the configuration file code.

Frontend Implementation

Step 8: HTML Templates

Create `templates/index.html`: Copy the complete HTML template code.

Step 9: Static Assets

Create `static/js/app.js`: Copy the complete JavaScript application code.

Create `static/css/style.css`: Copy the complete CSS styling code.

Step 10: Deployment Scripts

Create `deploy.sh`:

```bash
# Copy the deployment script code
chmod +x deploy.sh
```

Create `run.sh`:

```bash
cat > run.sh << 'EOL'
#!/bin/bash
echo "🚀 Starting RWA Tokenization POC..."

# Activate virtual environment
source venv/bin/activate

# Set environment variables
export FLASK_APP=app/main.py
export FLASK_ENV=development

# Create logs directory if it doesn't exist
mkdir -p logs

# Start the application
python app/main.py
EOL

chmod +x run.sh
```

Testing and Validation

Step 11: Basic Testing

Create `tests/test_basic.py`:

python

```python
import pytest
import sys
import os

# Add the app directory to the Python path
sys.path.insert(0, os.path.join(os.path.dirname(__file__), '..'))

def test_python_version():
    """Test that Python version is compatible"""
    assert sys.version_info >= (3, 8), "Python 3.8 or higher required"

def test_imports():
    """Test that basic imports work"""
    try:
        import flask
        import sqlalchemy
        import spacy
        import nltk
        print("✅ All imports successful")
        return True
    except ImportError as e:
        print(f"❌ Import error: {e}")
        return False

def test_spacy_model():
    """Test that spaCy model is available"""
    try:
        import spacy
        nlp = spacy.load("en_core_web_sm")
        print("✅ spaCy model loaded successfully")
        return True
    except Exception as e:
        print(f"❌ spaCy model error: {e}")
        return False

def test_nltk_data():
    """Test that NLTK data is available"""
    try:
        import nltk
        from nltk.sentiment import SentimentIntensityAnalyzer
        analyzer = SentimentIntensityAnalyzer()
        print("✅ NLTK data loaded successfully")
        return True
    except Exception as e:
        print(f"❌ NLTK data error: {e}")
        return False
```

```
if __name__ == '__main__':
    print("🧪 Running basic tests...")

    tests = [
        test_python_version,
        test_imports,
        test_spacy_model,
        test_nltk_data
    ]

    passed = 0
    for test in tests:
        try:
            if test():
                passed += 1
        except Exception as e:
            print(f"❌ Test failed: {e}")

    print(f"\n📊 Results: {passed}/{len(tests)} tests passed")

    if passed == len(tests):
        print("🎉 All tests passed! Ready to start the application.")
    else:
        print("⚠️ Some tests failed. Please resolve issues before continuing.")
```

Run basic tests:

```bash
python tests/test_basic.py
```

Step 12: Database Initialization

Create database initialization script:

```python
# Create init_db.py
cat > init_db.py << 'EOL'
#!/usr/bin/env python3

import sys
import os
sys.path.append('.')

try:
    from app.main import app, db

    print("🗄️ Initializing database...")

    with app.app_context():
        # Drop all tables (use with caution!)
        db.drop_all()
        print("📤 Dropped existing tables")

        # Create all tables
        db.create_all()
        print("📥 Created new tables")

        # Verify tables were created
        from sqlalchemy import inspect
        inspector = inspect(db.engine)
        tables = inspector.get_table_names()
        print(f"✅ Created tables: {tables}")

    print("🎉 Database initialization complete!")

except Exception as e:
    print(f"❌ Database initialization failed: {e}")
    sys.exit(1)
EOL

python init_db.py
```

Deployment Process

Step 13: Application Startup

Method 1: Using run script

```bash
./run.sh
```

Method 2: Manual startup

```bash
# Activate virtual environment
source venv/bin/activate

# Set environment variables
export FLASK_APP=app/main.py
export FLASK_ENV=development

# Start application
python app/main.py
```

Expected output:

```
🚀 Starting RWA Tokenization POC...
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://[your-ip]:5000
```

Step 14: Application Verification

Open browser and test:

1. Navigate to `http://localhost:5000`
2. Verify dashboard loads with statistics cards
3. Test asset submission form
4. Check browser console for errors

Test API endpoints:

```bash
# Health check
curl http://localhost:5000/api/health

# Should return:
{
  "status": "healthy",
  "timestamp": "2024-01-01T00:00:00Z",
  "version": "1.0.0"
}
```

Post-Deployment Verification

Step 15: Functional Testing

Test 1: Asset Submission

1. Fill out the asset form with sample data:
   - Wallet: `0x742d35Cc6e34d8d7C15fE14c123456789abcdef0`
   - Description: `"I want to tokenize my $500,000 apartment in Manhattan"`
   - Email: `test@example.com`
2. Click "Submit Asset"
3. Verify success message and follow-up questions appear
4. Check that asset appears in "Your Assets" section

Test 2: Asset Verification

1. Click "View" on submitted asset
2. Click "Verify Asset" button
3. Verify verification results appear
4. Check asset status updates to "verified"

Test 3: Asset Tokenization

1. For verified asset, click "Tokenize Asset"
2. Verify tokenization success message
3. Check that token ID is generated
4. Verify asset status shows as tokenized

Step 16: Log Monitoring

Check application logs:

```bash
# View recent logs
tail -f logs/app.log

# Search for errors
grep ERROR logs/app.log

# Check specific operations
grep "Asset created successfully" logs/app.log
```

Common Issues and Solutions

Issue 1: Import Errors

Problem: `ModuleNotFoundError: No module named 'app'`

Solution:

```bash
# Ensure you're in the correct directory
pwd  # Should show rwa-tokenization-poc

# Verify virtual environment is activated
which python  # Should show venv path

# Check Python path
python -c "import sys; print(sys.path)"

# Add current directory to Python path
export PYTHONPATH=$PYTHONPATH:$(pwd)
```

Issue 2: spaCy Model Not Found

Problem: `OSError: [E050] Can't find model 'en_core_web_sm'`

Solution:

```bash
# Reinstall spaCy model
python -m spacy download en_core_web_sm

# Verify installation
python -c "import spacy; nlp = spacy.load('en_core_web_sm'); print('Model loaded succe
```

Issue 3: NLTK Data Missing

Problem: `LookupError: NLTK data not found`

Solution:

```bash
# Download NLTK data interactively
python -c "
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('vader_lexicon')
"
```

Issue 4: Database Errors

Problem: `sqlite3.OperationalError: no such table`

Solution:

```bash
# Reinitialize database
python init_db.py

# Or manually create tables
python -c "
from app.main import app, db
with app.app_context():
    db.create_all()
    print('Tables created')
"
```

Issue 5: Port Already in Use

Problem: `OSError: [Errno 48] Address already in use`

Solution:

```bash
# Find process using port 5000
lsof -i :5000

# Kill the process
kill -9 <PID>

# Or use different port
export PORT=5001
python app/main.py
```

Issue 6: Permission Errors

Problem: `Permission denied` when running scripts

Solution:

```bash
# Make scripts executable
chmod +x deploy.sh
chmod +x run.sh

# Or run with bash
bash deploy.sh
bash run.sh
```

Issue 7: Frontend Not Loading

Problem: Static files (CSS/JS) not loading or 404 errors

Solution:

```bash
# Verify file structure
ls -la static/css/style.css
ls -la static/js/app.js
ls -la templates/index.html

# Check Flask static folder configuration in app/main.py
# Ensure: template_folder='../templates', static_folder='../static'

# Clear browser cache and reload
```

Issue 8: Database Connection Issues

Problem: `sqlite3.OperationalError: database is locked`

Solution:

```bash
# Stop all running Flask instances
pkill -f "python app/main.py"

# Remove database lock (if exists)
rm -f rwa_tokenization.db-journal

# Restart application
./run.sh
```

Advanced Configuration

Step 17: Production Configuration

Create production environment file:

```bash
cat > .env.production << 'EOL'
SECRET_KEY=your-very-secure-production-key-change-this
FLASK_ENV=production
PORT=5000
DATABASE_URL=postgresql://user:password@localhost/rwa_tokenization
LOG_LEVEL=WARNING
LOG_FILE=logs/production.log
MAX_CONTENT_LENGTH=16777216
UPLOAD_FOLDER=uploads
EOL
```

Create production requirements:

bash

```bash
cat > requirements-prod.txt << 'EOL'
Flask==2.3.3
Flask-SQLAlchemy==3.0.5
Flask-CORS==4.0.0
spacy==3.6.1
nltk==3.8.1
requests==2.31.0
python-dateutil==2.8.2
Werkzeug==2.3.7
gunicorn==21.2.0
psycopg2-binary==2.9.7
redis==4.6.0
python-dotenv==1.0.0
EOL
```

Step 18: Docker Deployment (Optional)

Create Dockerfile:

dockerfile

```dockerfile
FROM python:3.9-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements and install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Download spaCy model
RUN python -m spacy download en_core_web_sm

# Download NLTK data
RUN python -c "import nltk; nltk.download('punkt'); nltk.download('stopwords'); nltk.d

# Copy application code
COPY . .

# Create necessary directories
RUN mkdir -p logs uploads

# Expose port
EXPOSE 5000

# Set environment variables
ENV FLASK_APP=app/main.py
ENV FLASK_ENV=production

# Run the application
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "--workers", "4", "app.main:app"]
```

Create docker-compose.yml:

```yaml
version: '3.8'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=production
      - SECRET_KEY=your-production-secret-key
      - DATABASE_URL=sqlite:///data/rwa_tokenization.db
    volumes:
      - ./data:/app/data
      - ./logs:/app/logs
      - ./uploads:/app/uploads
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - web
    restart: unless-stopped
```

Docker deployment commands:

```bash
# Build and run with Docker
docker-compose up -d

# View logs
docker-compose logs -f web

# Stop services
docker-compose down
```

Monitoring and Maintenance

Step 19: Log Analysis

Create log monitoring script:

```bash
cat > monitor_logs.sh << 'EOL'
#!/bin/bash

LOG_FILE="logs/app.log"

echo "📊 RWA Tokenization Log Analysis"
echo "================================"

if [ ! -f "$LOG_FILE" ]; then
    echo "❌ Log file not found: $LOG_FILE"
    exit 1
fi

echo "📈 Recent Activity (Last 50 lines):"
tail -50 "$LOG_FILE"

echo -e "\n🔍 Error Summary:"
grep -c "ERROR" "$LOG_FILE" && echo "Total errors found" || echo "No errors found"

echo -e "\n✅ Success Summary:"
grep -c "Asset created successfully" "$LOG_FILE" && echo "Assets created" || echo "No 
grep -c "Verification completed" "$LOG_FILE" && echo "Verifications completed" || echo
grep -c "Tokenization completed" "$LOG_FILE" && echo "Tokenizations completed" || echo

echo -e "\n🕐 Recent Errors (Last 10):"
grep "ERROR" "$LOG_FILE" | tail -10

EOL

chmod +x monitor_logs.sh
```

Step 20: Health Monitoring

Create health check script:

bash

```
cat > health_check.sh << 'EOL'
#!/bin/bash

BASE_URL="http://localhost:5000"

echo "🏥 RWA Tokenization Health Check"
echo "================================="

# Test 1: Health endpoint
echo "Testing health endpoint..."
HEALTH_RESPONSE=$(curl -s "$BASE_URL/api/health")
if echo "$HEALTH_RESPONSE" | grep -q '"status": "healthy"'; then
    echo "✅ Health endpoint: OK"
else
    echo "❌ Health endpoint: FAILED"
    echo "Response: $HEALTH_RESPONSE"
fi

# Test 2: Main page
echo "Testing main page..."
MAIN_RESPONSE=$(curl -s -o /dev/null -w "%{http_code}" "$BASE_URL/")
if [ "$MAIN_RESPONSE" = "200" ]; then
    echo "✅ Main page: OK"
else
    echo "❌ Main page: FAILED (HTTP $MAIN_RESPONSE)"
fi

# Test 3: Static files
echo "Testing static files..."
CSS_RESPONSE=$(curl -s -o /dev/null -w "%{http_code}" "$BASE_URL/static/css/style.css"
JS_RESPONSE=$(curl -s -o /dev/null -w "%{http_code}" "$BASE_URL/static/js/app.js")

if [ "$CSS_RESPONSE" = "200" ] && [ "$JS_RESPONSE" = "200" ]; then
    echo "✅ Static files: OK"
else
    echo "❌ Static files: FAILED (CSS: $CSS_RESPONSE, JS: $JS_RESPONSE)"
fi

# Test 4: Database connectivity
echo "Testing database connectivity..."
STATS_RESPONSE=$(curl -s "$BASE_URL/api/stats")
if echo "$STATS_RESPONSE" | grep -q '"total_assets"'; then
    echo "✅ Database: OK"
else
    echo "❌ Database: FAILED"
    echo "Response: $STATS_RESPONSE"
```

```
fi

echo -e "\n📊 System Status Summary:"
echo "Time: $(date)"
echo "Application URL: $BASE_URL"

EOL

chmod +x health_check.sh
```

Performance Optimization

Step 21: Database Optimization

Create database optimization script:

python

```
# create optimize_db.py
cat > optimize_db.py << 'EOL'
#!/usr/bin/env python3

import sys
sys.path.append('.')

from app.main import app, db
from sqlalchemy import text

def optimize_database():
    """Optimize database performance"""
    print("🔧 Optimizing database...")

    with app.app_context():
        try:
            # Add indexes for better query performance
            indexes = [
                "CREATE INDEX IF NOT EXISTS idx_assets_user_id ON asset(user_id);",
                "CREATE INDEX IF NOT EXISTS idx_assets_verification_status ON asset(ve
                "CREATE INDEX IF NOT EXISTS idx_assets_token_id ON asset(token_id);",
                "CREATE INDEX IF NOT EXISTS idx_transactions_asset_id ON transaction(a
                "CREATE INDEX IF NOT EXISTS idx_transactions_type ON transaction(trans
                "CREATE INDEX IF NOT EXISTS idx_users_wallet ON user(wallet_address);"
            ]

            for index_sql in indexes:
                try:
                    db.session.execute(text(index_sql))
                    print(f"✅ Created index: {index_sql.split('idx_')[1].split(' ')[0
                except Exception as e:
                    print(f"⚠️  Index creation skipped (may already exist): {e}")

            db.session.commit()

            # Analyze database (SQLite specific)
            db.session.execute(text("ANALYZE;"))
            db.session.commit()

            print("🎉 Database optimization complete!")

        except Exception as e:
            print(f"❌ Database optimization failed: {e}")
            db.session.rollback()

if __name__ == '__main__':
    optimize_database()
EOL

python optimize_db.py
```

Step 22: Application Profiling

Create performance testing script:

python

```
# create performance_test.py
cat > performance_test.py << 'EOL'
#!/usr/bin/env python3

import time
import requests
import json
import sys
from concurrent.futures import ThreadPoolExecutor

BASE_URL = "http://localhost:5000"

def test_endpoint(endpoint, method="GET", data=None):
    """Test a single endpoint and measure response time"""
    start_time = time.time()

    try:
        if method == "GET":
            response = requests.get(f"{BASE_URL}{endpoint}")
        elif method == "POST":
            response = requests.post(f"{BASE_URL}{endpoint}",
                                      json=data,
                                      headers={'Content-Type': 'application/json'})

        end_time = time.time()
        response_time = (end_time - start_time) * 1000  # Convert to milliseconds

        return {
            'endpoint': endpoint,
            'method': method,
            'status_code': response.status_code,
            'response_time_ms': round(response_time, 2),
            'success': response.status_code < 400
        }

    except Exception as e:
        return {
            'endpoint': endpoint,
            'method': method,
            'error': str(e),
            'success': False
        }

def run_performance_tests():
    """Run performance tests on key endpoints"""
    print("🚀 Running Performance Tests")
```

```python
    print("============================")

    # Test cases
    test_cases = [
        ('/api/health', 'GET'),
        ('/api/stats', 'GET'),
        ('/', 'GET'),
        ('/static/css/style.css', 'GET'),
        ('/static/js/app.js', 'GET'),
    ]

    # Asset intake test data
    intake_data = {
        'wallet_address': '0x742d35Cc6e34d8d7C15fE14c123456789abcdef0',
        'user_input': 'I want to tokenize my $100,000 car, a 2020 Honda Civic',
        'email': 'test@example.com'
    }
    test_cases.append(('/api/intake', 'POST', intake_data))

    results = []

    for test_case in test_cases:
        endpoint = test_case[0]
        method = test_case[1]
        data = test_case[2] if len(test_case) > 2 else None

        print(f"Testing {method} {endpoint}...")
        result = test_endpoint(endpoint, method, data)
        results.append(result)

        if result['success']:
            print(f"✅ {result['response_time_ms']}ms")
        else:
            print(f"❌ Failed: {result.get('error', f'HTTP {result.get(\"status_code\"

    # Performance summary
    print(f"\n📊 Performance Summary:")
    print("=" * 50)

    successful_tests = [r for r in results if r['success']]
    if successful_tests:
        avg_response_time = sum(r['response_time_ms'] for r in successful_tests) / len
        max_response_time = max(r['response_time_ms'] for r in successful_tests)
        min_response_time = min(r['response_time_ms'] for r in successful_tests)

        print(f"✅ Successful requests: {len(successful_tests)}/{len(results)}")
        print(f"📈 Average response time: {avg_response_time:.2f}ms")
        print(f"🔺 Max response time: {max_response_time:.2f}ms")
        print(f"🔻 Min response time: {min_response_time:.2f}ms")

        # Performance recommendations
        if avg_response_time > 1000:
            print("⚠️  Warning: Average response time > 1 second")
        if max_response_time > 5000:
            print("⚠️  Warning: Some requests taking > 5 seconds")

    else:
        print("❌ No successful requests")

def load_test(num_requests=10):
    """Simple load test"""
    print(f"\n🔥 Load Testing ({num_requests} concurrent requests)")
    print("=" * 50)

    def make_request():
        return test_endpoint('/api/health')

    start_time = time.time()

    with ThreadPoolExecutor(max_workers=num_requests) as executor:
        futures = [executor.submit(make_request) for _ in range(num_requests)]
        results = [future.result() for future in futures]

    end_time = time.time()
    total_time = end_time - start_time

    successful = len([r for r in results if r['success']])
    avg_response_time = sum(r['response_time_ms'] for r in results if r['success']) /

    print(f"✅ Successful requests: {successful}/{num_requests}")
    print(f"⏱️  Total time: {total_time:.2f}s")
    print(f"📊 Requests per second: {num_requests/total_time:.2f}")
    print(f"📈 Average response time: {avg_response_time:.2f}ms")

if __name__ == '__main__':
    # Check if server is running
    try:
        requests.get(f"{BASE_URL}/api/health", timeout=5)
    except:
        print("❌ Server not running. Please start the application first.")
        sys.exit(1)

    run_performance_tests()
    load_test()
```

```
EOL

# Install requests if not already installed
pip install requests

python performance_test.py
```

Backup and Recovery

Step 23: Backup Strategy

Create backup script:

bash

```bash
cat > backup.sh << 'EOL'
#!/bin/bash

BACKUP_DIR="backups"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_NAME="rwa_backup_$DATE"

echo "💾 Creating backup: $BACKUP_NAME"

# Create backup directory
mkdir -p "$BACKUP_DIR"

# Create backup archive
tar -czf "$BACKUP_DIR/$BACKUP_NAME.tar.gz" \
    --exclude='venv' \
    --exclude='__pycache__' \
    --exclude='*.pyc' \
    --exclude='logs/*.log' \
    --exclude='backups' \
    .

echo "✅ Backup created: $BACKUP_DIR/$BACKUP_NAME.tar.gz"

# Keep only last 5 backups
cd "$BACKUP_DIR"
ls -t *.tar.gz | tail -n +6 | xargs -r rm --
echo "🧹 Old backups cleaned up"

# Show backup info
echo "📊 Backup Information:"
echo "  File: $BACKUP_NAME.tar.gz"
echo "  Size: $(du -h $BACKUP_NAME.tar.gz | cut -f1)"
echo "  Date: $(date)"
EOL

chmod +x backup.sh
```

Create restore script:

bash

```bash
cat > restore.sh << 'EOL'
#!/bin/bash

if [ -z "$1" ]; then
    echo "Usage: ./restore.sh <backup_file>"
    echo "Available backups:"
    ls -la backups/*.tar.gz 2>/dev/null || echo "No backups found"
    exit 1
fi

BACKUP_FILE="$1"

if [ ! -f "$BACKUP_FILE" ]; then
    echo "❌ Backup file not found: $BACKUP_FILE"
    exit 1
fi

echo "⚠️  This will restore from backup: $BACKUP_FILE"
echo "⚠️  Current data will be backed up first"
read -p "Continue? (y/N): " -n 1 -r
echo

if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    echo "Restore cancelled"
    exit 1
fi

# Create backup of current state
echo "💾 Backing up current state..."
./backup.sh

# Restore from backup
echo "📥 Restoring from backup..."
tar -xzf "$BACKUP_FILE"

echo "✅ Restore completed"
echo "🔄 Please restart the application"
EOL

chmod +x restore.sh
```

Final Verification

Step 24: Complete System Test

Create comprehensive test script:

bash

```
cat > system_test.sh << 'EOL'
#!/bin/bash

echo "🧪 RWA Tokenization System Test"
echo "================================"

# Test 1: Environment check
echo "1. Environment Check..."
if [ -d "venv" ] && [ -f "app/main.py" ] && [ -f "requirements.txt" ]; then
    echo "✅ Project structure OK"
else
    echo "❌ Project structure incomplete"
    exit 1
fi

# Test 2: Dependencies check
echo "2. Dependencies Check..."
source venv/bin/activate
python tests/test_basic.py > /dev/null 2>&1
if [ $? -eq 0 ]; then
    echo "✅ Dependencies OK"
else
    echo "❌ Dependencies failed"
    exit 1
fi

# Test 3: Database check
echo "3. Database Check..."
python init_db.py > /dev/null 2>&1
if [ $? -eq 0 ]; then
    echo "✅ Database OK"
else
    echo "❌ Database failed"
    exit 1
fi

# Test 4: Application startup
echo "4. Application Startup..."
python app/main.py &
APP_PID=$!
sleep 5

# Test 5: Health check
echo "5. Health Check..."
./health_check.sh > /dev/null 2>&1
if [ $? -eq 0 ]; then
```

```
    echo "✅ Application health OK"
else
    echo "❌ Application health failed"
    kill $APP_PID 2>/dev/null
    exit 1
fi

# Test 6: Performance check
echo "6. Performance Check..."
python performance_test.py > /dev/null 2>&1
if [ $? -eq 0 ]; then
    echo "✅ Performance OK"
else
    echo "⚠️  Performance issues detected"
fi

# Clean up
kill $APP_PID 2>/dev/null
wait $APP_PID 2>/dev/null

echo ""
echo "🎉 System test completed successfully!"
echo "🚀 Your RWA Tokenization POC is ready to use!"
echo ""
echo "Next steps:"
echo "1. Start the application: ./run.sh"
echo "2. Open browser: http://localhost:5000"
echo "3. Begin tokenizing assets!"
EOL

chmod +x system_test.sh
```

Quick Reference Guide

Essential Commands

```bash
# Setup and Installation
./deploy.sh                    # Initial deployment
source venv/bin/activate       # Activate virtual environment
pip install -r requirements.txt # Install dependencies

# Application Management
./run.sh                       # Start application
python app/main.py             # Start manually
./health_check.sh              # Check system health
./system_test.sh               # Complete system test

# Database Management
python init_db.py              # Initialize database
python optimize_db.py          # Optimize performance

# Monitoring and Maintenance
./monitor_logs.sh              # View log analysis
python performance_test.py     # Performance testing
./backup.sh                    # Create backup
./restore.sh <backup_file>     # Restore from backup
```

File Checklist

- [ ] `app/main.py` - Flask application
- [ ] `app/models/database.py` - Database models
- [ ] `app/agents/nlp_agent.py` - NLP processing
- [ ] `app/agents/verification_agent.py` - Asset verification
- [ ] `app/agents/tokenization_agent.py` - Token creation
- [ ] `templates/index.html` - Frontend template
- [ ] `static/js/app.js` - Frontend JavaScript
- [ ] `static/css/style.css` - Styling
- [ ] `requirements.txt` - Python dependencies
- [ ] `config.py` - Configuration settings
- [ ] `.env` - Environment variables

Troubleshooting Quick Fixes

```bash
# Common fixes
chmod +x *.sh                 # Fix permissions
rm -rf __pycache__            # Clear Python cache
python init_db.py             # Reset database
pkill -f "python app/main.py" # Kill stuck processes
```

🎉 **Congratulations! Your RWA Tokenization POC is now fully implemented and ready for use.**