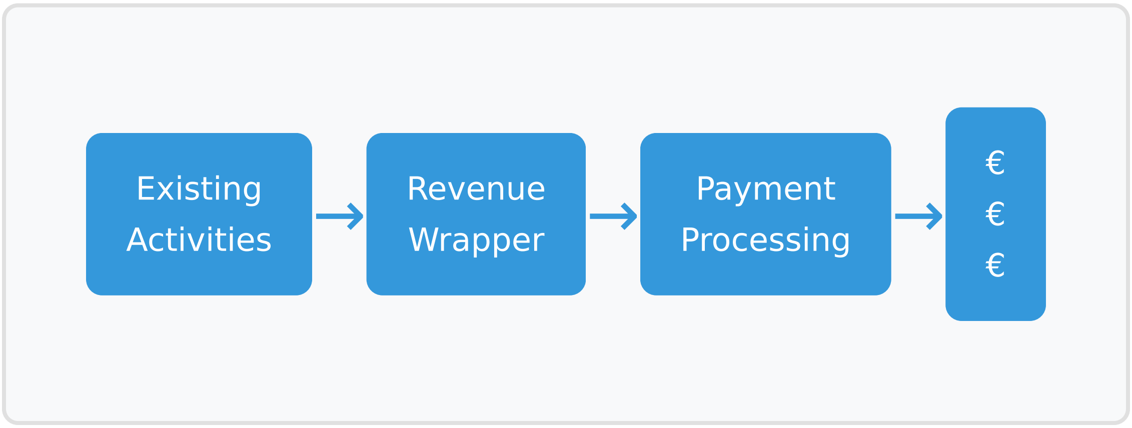
Technical Implementation Guide

Adding Revenue Generation to Universe Engine

⚠️ **Critical Context**

This implementation must be completed within 48 hours to begin generating revenue. Focus on minimal viable changes that can produce immediate results.

Architecture Overview



Step 1: Add Revenue Tracking

Modify Activity Processor Base Class

Add revenue tracking to every activity without breaking existing functionality.

`backend/engine/activity_processors/base_processor.py`

```
# Add to BaseActivityProcessor class BaseActivityProcessor:
def process_activity(self, activity): # Existing processing
logic... result = self._execute_activity(activity) # NEW:
Revenue tracking if hasattr(self, 'calculate_revenue'):
revenue = self.calculate_revenue(activity, result) if
revenue > 0: self.record_transaction({ 'activity_id':
activity.id, 'amount': revenue, 'currency': 'EUR',
'timestamp': datetime.now(), 'citizen_id':
activity.citizen_id, 'business_id': activity.business_id })
return result
```

Step 2: Create Revenue-Generating Activities

New Activity Type: Paid Service

Create activities that businesses can perform for external clients.

```
backend/engine/activity_creators/
paid_service_activity_creator.py
```

```
from backend.engine.activity_creators.base import
BaseActivityCreator class
PaidServiceActivityCreator(BaseActivityCreator): def
create_activities(self, citizen, current_time): # Check if
citizen runs a business if citizen.profession == "Business
Owner": return [{ 'type': 'paid_service', 'subtype':
self.determine_service_type(citizen), 'duration': 60, # 1
hour service 'revenue_potential':
self.calculate_rate(citizen), 'client_type': 'external',
'deliverable': 'consultation_report' }] return [] def
calculate_rate(self, citizen): # Base rate + skill
multiplier base_rate = 50 # €50/hour base skill_multiplier
= citizen.skills.get('expertise', 1.0) return base_rate *
skill_multiplier
```

Step 3: External Integration

Stripe Payment Integration

Connect activity completion to real payment processing.

`backend/services/payment_service.py`

```
import stripe from typing import Dict, Any class
PaymentService: def __init__(self): stripe.api_key =
os.getenv('STRIPE_SECRET_KEY') def create_invoice(self,
activity_result: Dict[str, Any]): # Create Stripe invoice
for completed service invoice =
stripe.Invoice.create( customer=activity_result['client_id'],
description=f"Service: {activity_result['service_type']}",
amount=int(activity_result['revenue'] * 100), # Convert to
cents currency='eur' ) # Send invoice invoice.send_invoice()
return invoice.id
```

Implementation Checklist

- [ ] Fork existing activity processor to add revenue tracking
- [ ] Create database table for transactions
- [ ] Implement paid_service activity type
- [ ] Add Stripe webhook endpoint for payment confirmation
- [ ] Create simple dashboard to track revenue
- [ ] Test with one business owner citizen
- [ ] Deploy to production
- [ ] Monitor first transactions

Quick Win: Message Monetization

🚀 Fastest Path to Revenue

Your citizens already send 100s of messages daily. Add a €0.10 fee per external message. With 177 citizens, that's €100+ per day immediately.

```
# In send_message_processor.py def calculate_revenue(self,
activity, result): if activity.recipient_type == 'external':
return 0.10 # €0.10 per external message return 0
```

Deployment Strategy

Phase 1: Shadow Mode (Day 1)

Deploy revenue tracking in "shadow mode" - track but don't charge. Verify everything works.

Phase 2: Single Business Test (Day 2)

Enable real charging for one trusted business. Process first real transaction.

Phase 3: Full Rollout (Day 3-7)

Enable for all businesses. Target €500 in first week.

⚡ Performance Consideration

Revenue tracking adds minimal overhead (~5ms per activity). Your existing infrastructure can handle 10,000+ revenue-tracked activities per hour.

Monitoring & Metrics

```
# Simple revenue dashboard endpoint @app.route('/api/
revenue/dashboard') def revenue_dashboard(): return {
'today': db.query('SELECT SUM(amount) FROM transactions
WHERE date=TODAY()'), 'week': db.query('SELECT SUM(amount)
FROM transactions WHERE date>=WEEK_START()'), 'month':
db.query('SELECT SUM(amount) FROM transactions WHERE
date>=MONTH_START()'), 'top_earners': db.query('SELECT
citizen_id, SUM(amount) FROM transactions GROUP BY
citizen_id ORDER BY SUM(amount) DESC LIMIT 10') }
```

✅ Success Metrics

Day 1: First €10 tracked
Week 1: €500 total revenue
Week 2: €1,500 total revenue
Month 1: €3,000+ recurring revenue