Cahier des Charges – Agent Marketplace

1. Introduction

1.1 Contexte

Le marché de l'intelligence artificielle connaît une explosion d'agents spécialisés développés par des experts, chercheurs et développeurs. Cependant, un problème fondamental bloque l'évolution de cet écosystème :

LE PROBLÈME CENTRAL : Les agents ne peuvent pas communiquer entre eux

Aujourd'hui, chaque agent IA est conçu pour interagir uniquement avec des humains :

- Interface chatbot (texte)
- API REST orientée "utilisateur final"
- Aucun protocole standardisé pour communication machine-to-machine
- Pas de mécanisme de découverte automatique
- Impossible pour un agent A d'appeler un agent B de manière native

Conséquence : L'écosystème IA est fragmenté. Chaque agent est un silo isolé.

Exemple concret du problème :

Un développeur veut créer un "Agent Assistant Personnel" qui doit :

1. Consulter la météo → appeler un Agent Météo
2. Réserver un restaurant → appeler un Agent Réservation
3. Vérifier le solde bancaire → appeler un Agent Bancaire
4. Analyser un contrat → appeler un Agent Juridique

Aujourd'hui, c'est techniquement impossible car :

- ❌ L'Agent Météo expose une API REST pour humains, pas pour agents
- ❌ L'Agent Réservation est un chatbot Telegram sans interface programmatique
- ❌ L'Agent Bancaire nécessite une authentification humaine (2FA, etc.)
- ❌ L'Agent Juridique est un SaaS fermé avec UI web uniquement
- ❌ Aucun protocole commun de communication
- ❌ Pas de registre pour découvrir ces agents
- ❌ Pas de mécanisme de négociation automatique (prix, permissions, formats)

Le développeur doit :

- Coder manuellement chaque intégration (4 APIs différentes)
- Gérer 4 systèmes d'authentification différents
- Maintenir 4 formats de données différents
- Négocier manuellement avec 4 fournisseurs différents
- = 80% du temps passé sur l'intégration, 20% sur la vraie valeur

AUTRES PROBLÈMES IDENTIFIÉS :

- **Barrière technique** : Un expert métier doit devenir développeur full-stack pour commercialiser son agent
- **Time-to-market long** : 3-6 mois pour passer d'un agent fonctionnel à une application commerciale
- **Coûts élevés** : Infrastructure, marketing, support client représentent 80% du temps
- **Distribution difficile** : Absence de canal de distribution standardisé
- **Pas de composabilité** : Impossible de créer des agents complexes en combinant des agents simples

Opportunité de marché :

- Explosion des agents IA spécialisés (santé, finance, juridique, etc.)
- Besoin croissant d'interopérabilité entre systèmes IA
- Demande des entreprises pour des solutions plug-and-play
- Modèle "Agent-as-a-Service" émergent mais non structuré

1.2 Vision du projet

Créer le "HTTP des agents IA" — le premier protocole et infrastructure permettant une communication universelle agent-to-agent.

Vision en 3 piliers :

1. PROTOCOLE UNIVERSEL (Le "HTTP" des agents)

- Standard de communication agent-to-agent
- Découverte automatique des capacités
- Négociation automatique (prix, permissions, formats)
- Compatible avec le protocole MCP (Model Context Protocol) mais étendu pour agent-to-agent

2. MARKETPLACE & REGISTRY (Le "DNS" des agents)

- Registre universel de tous les agents disponibles

- Discovery API : "Je cherche un agent expert en droit fiscal français"
- Publication en 1 ligne de code
- Distribution instantanée

3. GATEWAY INFRASTRUCTURE (Le "CDN" des agents)

- Routage intelligent des appels
- Authentification inter-agents
- Billing automatique
- Monitoring et analytics

L'analogie complète :

- *HTTP = protocole web → Notre protocole = communication agents*
- *DNS = annuaire web → Notre registry = annuaire agents*
- *CDN = distribution web → Notre gateway = distribution agents*

Résultat final : Un agent peut appeler n'importe quel autre agent aussi facilement qu'une page web appelle une autre page web.

1.3 Proposition de valeur unique

LE GAME-CHANGER : Communication agent-to-agent native

AVANT (situation actuelle) :

```
Agent A (Assistant Personnel)
  ↓
Doit intégrer manuellement 10 APIs différentes
  ↓
Code custom pour chaque service
  ↓
Maintenance cauchemardesque
```

AVEC NOTRE PROTOCOLE :

```
python
```

```
# Agent A peut appeler Agent B naturellement
result = agent_registry.call(
    agent_name="medical-expert",
    task="Analyse cette ordonnance",
    data=prescription_image
)
# L'Agent Médical répond directement
# Billing automatique
# Pas de code d'intégration
```

Pour les créateurs d'agents :

- ✅ Publier en 1 ligne : `publish(agent)`
- ✅ **Agent accessible par d'autres agents automatiquement** (NOUVEAU)
- ✅ Pas de code UI/UX à développer
- ✅ Pas d'infrastructure à gérer
- ✅ Distribution instantanée à des milliers d'agents et développeurs
- ✅ 80% des revenus reversés
- ✅ **Composabilité native** : ton agent peut devenir un building block

Pour les utilisateurs (développeurs créant des agents complexes) :

- ✅ **Appeler n'importe quel agent via un protocole unifié** (RÉVOLUTIONNAIRE)
- ✅ Découverte automatique : "Trouve-moi un agent expert en droit fiscal"
- ✅ Pas d'intégration custom à coder
- ✅ Paiement à l'usage transparent
- ✅ **Composer des agents complexes en combinant des agents simples**
- ✅ Outils gratuits + agents spécialisés payants

Pour les entreprises (qui veulent exposer leurs agents) :

- ✅ **Rendre leurs agents internes appelables par des agents externes** (UNIQUE)
- ✅ Gateway sécurisé clé-en-main
- ✅ Contrôle total sur permissions et accès
- ✅ Monétisation de leurs assets IA existants
- ✅ Pas besoin de développer une API publique

EXEMPLE CONCRET DE LA RÉVOLUTION :

Aujourd'hui : Je veux créer un "Agent Conseiller Financier Personnel"

- Semaine 1-2 : Intégrer API bancaire (custom)
- Semaine 3-4 : Intégrer API bourse (custom)
- Semaine 5-6 : Intégrer API crypto (custom)
- Semaine 7-8 : Intégrer API fiscalité (custom)
- **Total : 2 mois de dev d'intégration**

Avec notre protocole :

```python
class FinancialAdvisorAgent(Agent):
    def analyze_portfolio(self, user_id):
        # Appelle directement d'autres agents
        bank_data = market.call("bnp-banking-agent",
                                action="get_accounts",
                                user=user_id)

        stocks = market.call("stock-analysis-agent",
                             portfolio=bank_data.stocks)

        tax_opt = market.call("tax-optimizer-agent",
                              income=bank_data.total,
                              country="FR")

        return self.generate_advice(bank_data, stocks, tax_opt)

publish(FinancialAdvisorAgent)
```

Total : 1 journée de dev

C'est ça le game-changer : passer de semaines d'intégration à quelques lignes de code.

---

2. Objectifs stratégiques

2.1 Objectifs business

Année 1 : Prouver le protocole agent-to-agent

- **Metric #1 (Critical) :** 500+ appels agent-to-agent/jour (proof que le protocole est utilisé)
- 100 agents publiés sur la marketplace
- 1,000 développeurs utilisateurs actifs

- 50 créateurs d'agents actifs
- **3 entreprises pilotes** utilisant Gateway Enterprise (BNP-type use case)
- 40K€ MRR (Monthly Recurring Revenue)
- 480K€ ARR (Annual Recurring Revenue)

Année 2 : Devenir le standard

- **Metric #1 (Critical) :** 50,000+ appels agent-to-agent/jour
- 500 agents publiés
- 10,000 développeurs utilisateurs actifs
- 200 créateurs actifs
- **20 entreprises** utilisant Gateway Enterprise
- 50K€ MRR (30K marketplace + 20K enterprise)
- 600K€ ARR

Année 3 : Domination de marché

- **Metric #1 (Critical) :** 1M+ appels agent-to-agent/jour
- 2,000+ agents
- 50,000+ développeurs
- **"Si tu fais de l'IA, tu utilises notre protocole"** (comme "Si tu fais du web, tu utilises HTTP")
- Services premium entreprise mature
- 100K€+ MRR
- Levée série A ou rentabilité profitable

2.2 Objectifs techniques

- API response time < 200ms (P95)
- Uptime > 99.9%
- SDK disponible en Python, TypeScript, et Go
- Compatibilité totale avec le protocole MCP (Model Context Protocol)
- Support de 100,000 appels/jour dès le lancement
- Scalabilité horizontale pour atteindre 10M appels/jour

2.3 Objectifs utilisateurs

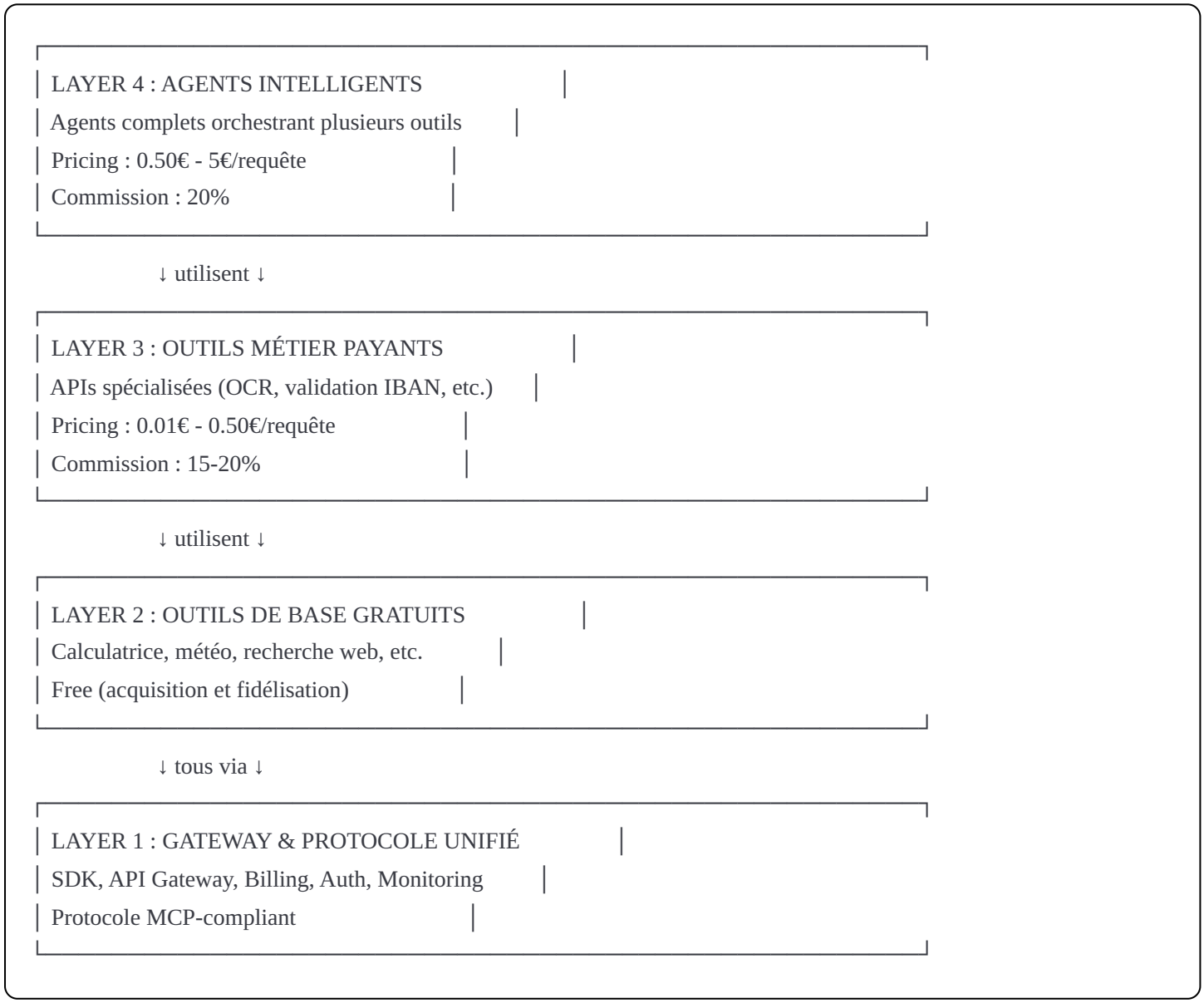
- Onboarding créateur < 15 minutes
- Publication d'un agent < 5 minutes
- Première utilisation d'un agent < 2 minutes
- Documentation complète et exemples pour tous les use-cases
- Support réactif (< 24h)

---

3. Architecture du système

3.1 Vue d'ensemble

Le système est composé de 4 couches principales :



3.2 Composants techniques

3.2.1 SDK Client (Python, TypeScript, Go)

Fonctionnalités :

- Interface unifiée pour appeler n'importe quel outil/agent
- Gestion automatique de l'authentification
- Retry logic et gestion d'erreurs
- Découverte dynamique des capacités
- Caching intelligent
- Métriques et monitoring

Exemple d'utilisation :

```python
from marketplace_sdk import Marketplace

market = Marketplace(api_key="user_xxx")

# Appel d'outil simple
result = market.call("calculator", expression="2+2")

# Appel d'agent complexe
result = market.call("legal-agent", contract=pdf_file,
                     question="Est-ce RGPD-compliant?")
```

3.2.2 API Gateway

Responsabilités :

- Routing intelligent des requêtes
- Authentification et autorisation
- Rate limiting et quotas
- Load balancing
- Logging et audit trail
- Billing et metering

Technologies :

- Node.js + Express ou Go (performance)

- PostgreSQL (metadata, billing)
- Redis (cache, rate limiting)
- RabbitMQ (async processing)

3.2.3 Tool/Agent Registry

Fonctionnalités :

- Catalogue de tous les outils et agents
- Métadonnées (description, pricing, input/output schema)
- Versioning
- Discovery API (recherche, filtres, recommandations)
- Metrics (usage, ratings, performance)

Structure de données :

json

```
{
  "id": "legal-analysis-agent",
  "name": "Legal Analysis Agent",
  "description": "Analyzes contracts for GDPR compliance",
  "version": "1.2.3",
  "creator_id": "creator_abc123",
  "type": "agent", // ou "tool"
  "protocol": "mcp-v1",
  "pricing": {
    "type": "per_call",
    "amount": 2.00,
    "currency": "EUR"
  },
  "input_schema": {...},
  "output_schema": {...},
  "endpoint": {
    "type": "serverless",
    "url": "https://exec.marketplace.io/agent/legal-analysis"
  },
  "stats": {
    "total_calls": 15420,
    "avg_latency_ms": 850,
    "success_rate": 0.987,
    "rating": 4.7
  }
}
```

3.2.4 Execution Engine

Responsabilités :

- Exécution des agents en environnement isolé
- Serverless scaling (AWS Lambda, Google Cloud Run)
- Monitoring des performances
- Gestion des timeouts et erreurs

Sécurité :

- Sandbox complet (pas d'accès réseau non autorisé)
- Limites de ressources (CPU, RAM, temps)
- Validation des inputs/outputs
- Audit logging

3.2.5 Billing & Revenue Share

Fonctionnalités :

- Metering en temps réel
- Facturation automatique (Stripe)
- Distribution automatique des revenus aux créateurs
- Génération de factures
- Dashboard financier pour créateurs

Flow :

```
Utilisateur appelle agent (2€)
   ↓
Metering enregistre l'appel
   ↓
Fin du mois :
   - Utilisateur facturé : 2€
   - Créateur reçoit : 1.60€ (80%)
   - Plateforme garde : 0.40€ (20%)
```

3.2.6 Marketplace Frontend

Pages principales :

- Landing page
- Catalogue (recherche, filtres, catégories)
- Page de détail agent/outil
- Creator dashboard
- User dashboard
- Documentation
- Pricing

Technologies :

- Next.js + React
- TailwindCSS
- Algolia (recherche)
- Analytics (Mixpanel ou Plausible)

3.3 Protocole de communication Agent-to-Agent

LE CŒUR DE L'INNOVATION : Protocole natif de communication inter-agents

3.3.1 Pourquoi les protocoles existants ne suffisent pas

MCP (Model Context Protocol) :

- ✅ Excellent pour agent → outil (one-way)
- ❌ Pas conçu pour agent ↔ agent (bidirectionnel)
- ❌ Pas de découverte dynamique d'agents
- ❌ Pas de négociation automatique
- ❌ Pas de gestion de sessions conversationnelles

APIs REST classiques :

- ✅ Communication HTTP standard
- ❌ Chaque API a son propre format
- ❌ Pas de métadonnées sémantiques ("Que sait faire cet agent ?")
- ❌ Pas de billing natif
- ❌ Pensées pour humains, pas pour agents

Notre protocole = MCP Extended for Agent-to-Agent

3.3.2 Caractéristiques du protocole

1. DISCOVERY (Découverte d'agents)

Un agent peut interroger le registry :

```json
{
  "method": "agent.discover",
  "params": {
    "query": "expert en droit fiscal français",
    "filters": {
      "category": "legal",
      "max_price_per_call": 5.00,
      "min_rating": 4.0,
      "response_time_max_ms": 2000
    }
  }
}

// Réponse
{
  "agents": [
    {
      "id": "legal-tax-expert-fr",
      "name": "Expert Fiscal France",
      "description": "Spécialisé en droit fiscal français",
      "capabilities": ["analyze_contract", "tax_optimization", "compliance_check"],
      "pricing": {"per_call": 2.50},
      "rating": 4.8,
      "avg_response_time_ms": 850
    }
  ]
}
```

2. CAPABILITY NEGOTIATION (Négociation de capacités)

Avant d'appeler un agent, on demande ce qu'il sait faire :

```json
```

```
{
  "method": "agent.describe",
  "params": {
    "agent_id": "legal-tax-expert-fr"
  }
}

// Réponse (format MCP + extensions)
{
  "capabilities": {
    "methods": [
      {
        "name": "analyze_contract",
        "description": "Analyse un contrat pour conformité fiscale",
        "input_schema": {
          "contract_pdf": {"type": "file", "format": "pdf"},
          "analysis_type": {"type": "string", "enum": ["basic", "detailed"]}
        },
        "output_schema": {
          "compliant": {"type": "boolean"},
          "issues": {"type": "array"},
          "recommendations": {"type": "array"}
        },
        "pricing": {
          "basic": 1.00,
          "detailed": 2.50
        },
        "estimated_time_ms": 800
      }
    ]
  },
  "authentication": "api_key", // ou "oauth2", "mutual_tls"
  "session_support": true // Support conversation multi-tours
}
```

3. INVOCATION (Appel d'agent)

```
json
```

```
{
  "method": "agent.invoke",
  "params": {
    "agent_id": "legal-tax-expert-fr",
    "method": "analyze_contract",
    "arguments": {
      "contract_pdf": "base64_data...",
      "analysis_type": "detailed"
    },
    "caller_context": {
      "agent_id": "financial-advisor-bot", // Qui appelle
      "user_id": "user_xyz", // Pour qui
      "session_id": "session_abc123" // Si conversation
    }
  }
}

// Réponse
{
  "result": {
    "compliant": false,
    "issues": [
      "Clause 3.2 non conforme RGPD",
      "TVA calculée incorrectement"
    ],
    "recommendations": [...]
  },
  "billing": {
    "amount_charged": 2.50,
    "currency": "EUR",
    "transaction_id": "tx_xyz"
  },
  "metadata": {
    "execution_time_ms": 762,
    "agent_version": "1.2.3"
  }
}
```

4. SESSION MANAGEMENT (Conversations multi-tours)

Pour agents stateful (type chatbot) :

```
json
```

```
// Tour 1
{
  "method": "agent.invoke",
  "params": {
    "agent_id": "medical-diagnosis-agent",
    "method": "start_diagnosis",
    "arguments": {
      "symptoms": ["fatigue", "fièvre"]
    }
  }
}
// → Répond avec session_id

// Tour 2 (dans la même session)
{
  "method": "agent.invoke",
  "params": {
    "agent_id": "medical-diagnosis-agent",
    "method": "continue_diagnosis",
    "arguments": {
      "additional_info": "Depuis 3 jours"
    },
    "session_id": "session_xyz" // Contexte conservé
  }
}
```

5. AUTHENTICATION INTER-AGENTS

Chaque agent a une identité vérifiable :

```
json

{
  "caller_auth": {
    "agent_id": "financial-advisor-bot",
    "api_key": "ak_xxx", // ou JWT token
    "signature": "..." // Proof of identity
  }
}
```

L'agent appelé peut :

- Vérifier l'identité de l'appelant
- Accepter ou refuser selon des règles (whitelist, reputation score)
- Logger qui a accédé à quoi (audit trail)

6. ERROR HANDLING & FALLBACKS

```json
// Si l'agent ne peut pas répondre
{
  "error": {
    "code": "CAPABILITY_NOT_FOUND",
    "message": "Cette méthode n'existe pas",
    "suggested_alternatives": [
      {
        "agent_id": "another-legal-agent",
        "method": "similar_analysis",
        "confidence": 0.85
      }
    ]
  }
}
```

3.3.3 Avantages du protocole

Vs. APIs REST classiques :

- ✅ Auto-descriptif (pas besoin de lire une doc externe)
- ✅ Découverte dynamique (je trouve l'agent dont j'ai besoin)
- ✅ Billing intégré (pas de setup Stripe séparé)
- ✅ Versioning natif
- ✅ Fallbacks et alternatives automatiques

Vs. MCP seul :

- ✅ Agent-to-agent (pas juste agent-to-tool)
- ✅ Discovery registry
- ✅ Sessions conversationnelles
- ✅ Authentication mutuelle
- ✅ Marketplace metadata (pricing, ratings)

Composabilité extrême : Un agent peut en appeler 10 autres sans friction :

```python
```

```
class SuperAgent(Agent):
    def complex_task(self, data):
        # Compose 5 agents différents
        step1 = market.call("ocr-agent", image=data.scan)
        step2 = market.call("translation-agent", text=step1.text)
        step3 = market.call("legal-agent", contract=step2.translated)
        step4 = market.call("tax-agent", legal_analysis=step3)
        step5 = market.call("summary-agent", data=[step1, step2, step3, step4])
        return step5.summary
```

Chaque appel :

- Découvre automatiquement le bon agent
- Négocie le format
- S'authentifie
- Paie automatiquement
- Gère les erreurs
- **Tout ça transparent pour le développeur**

3.4 Agent-to-Agent Communication : Cas d'usage Entreprise

LE CAS D'USAGE KILLER : Entreprises exposant leurs agents

3.4.1 Le problème actuel

Scénario typique : La BNP a développé un excellent agent bancaire interne pour :

- Vérification IBAN
- Scoring crédit
- Détection fraude
- KYC (Know Your Customer)

Aujourd'hui, si une fintech veut utiliser cet agent :

1. ❌ BNP doit exposer une API REST publique (complexe, risqué)
2. ❌ BNP doit donner accès direct à ses bases de données (inacceptable)
3. ❌ La fintech doit intégrer l'API spécifique BNP (custom pour chaque banque)
4. ❌ Pas de standard, chaque banque a son format

Résultat : Ça n'arrive presque jamais. Les assets IA des entreprises restent cloisonnés.

3.4.2 Notre solution : Gateway Enterprise

Architecture :

```
Agent Fintech (externe)
   ↓
Notre Gateway (authentification, routing, billing)
   ↓
Agent BNP (reste dans le périmètre sécurisé BNP)
   ↓
Systèmes internes BNP (databases, APIs privées)
```

L'agent BNP :

- Reste dans l'infrastructure BNP (zero-trust)
- Applique les règles métier de la BNP
- Filtre les données sensibles
- Logue tous les accès
- Respecte la compliance

L'agent Fintech :

- Appelle l'agent BNP via notre protocole standard
- Pas besoin de connaître l'infra interne BNP
- Paiement automatique à l'usage
- Même interface que tous les autres agents

Exemple concret :

```python
```

```python
# Côté Fintech
class FintechLoanAgent(Agent):
    def evaluate_loan_request(self, customer):
        # Appelle l'agent BNP pour vérifier identité
        kyc_result = market.call(
            "bnp-kyc-agent",
            action="verify_identity",
            customer_id=customer.id,
            consent_token=customer.consent # GDPR
        )

        if kyc_result.verified:
            # Appelle un autre agent pour scoring
            credit_score = market.call(
                "credit-scoring-agent",
                income=customer.income,
                history=kyc_result.banking_history
            )

            return self.make_decision(kyc_result, credit_score)
```

```python
```

```
# Côté BNP (agent qui reste chez BNP)
class BNPKYCAgent(Agent):
    def verify_identity(self, customer_id, consent_token):
        # Vérifie le consentement GDPR
        if not self.check_consent(consent_token):
            raise PermissionError("Consentement manquant")

        # Interroge les systèmes internes BNP (privés)
        customer_data = self.bnp_internal_db.query(customer_id)

        # Filtre les données sensibles avant de renvoyer
        safe_data = self.filter_sensitive_data(customer_data)

        # Logue l'accès (audit trail)
        self.audit_log(caller="fintech-loan-agent",
                       customer=customer_id,
                       data_accessed=safe_data.keys())

        return {
            "verified": True,
            "banking_history": safe_data.history, # Filtré
            # Données bancaires complètes JAMAIS exposées
        }
```

3.4.3 Avantages pour l'entreprise (BNP)

Sécurité :

- ✅ Agent reste dans leur infrastructure (on-premise ou VPC)
- ✅ Aucune donnée sensible n'est exposée directement
- ✅ Contrôle total sur les permissions
- ✅ Audit trail complet (qui a accédé à quoi, quand)
- ✅ Révocation instantanée si problème

Business :

- ✅ Monétise des assets existants (agent déjà développé)
- ✅ Nouvelle source de revenus (0.10€ par vérification KYC × 10,000/jour = 1,000€/jour)
- ✅ Pas besoin de créer une API publique (on gère l'infra)
- ✅ Pas de marketing à faire (agents listés sur marketplace)

Technique :

- ✅ Pas de développement lourd (just expose via notre protocole)
- ✅ Scalabilité gérée par notre gateway
- ✅ Monitoring et analytics fournis
- ✅ Support et SLA garantis

3.4.4 Ce que vend la plateforme à l'entreprise

Gateway-as-a-Service :

Package Starter (500€/mois) :

- Exposition d'1 agent
- Jusqu'à 10,000 appels/mois
- Dashboard de monitoring
- Support email

Package Business (2,000€/mois) :

- Exposition de 5 agents
- Jusqu'à 100,000 appels/mois
- Analytics avancés
- Audit logs 1 an
- Support prioritaire

Package Enterprise (sur devis) :

- Agents illimités
- Volume illimité
- SLA 99.99%
- Déploiement on-premise possible
- Compliance sur-mesure (HIPAA, PCI-DSS, etc.)
- Support 24/7
- Account manager dédié

Commission sur usage en plus :

- 5-10% de commission sur chaque appel payant
- Exemple : BNP facture 0.10€/KYC → on prend 0.01€

3.4.5 Différence critique vs. API Gateway classique

API Gateway classique (Kong, AWS API Gateway) :

```
Client → API Gateway → Backend API
```

- ❌ Le backend doit être une API REST standard
- ❌ Pas de découverte automatique
- ❌ Pas de billing intégré
- ❌ Pas de protocole agent-to-agent

Notre Gateway Agent-to-Agent :

```
Agent Externe → Notre Gateway → Agent Interne (avec intelligence)
```

- ✅ L'agent interne peut être n'importe quoi (Python script, chatbot, LLM)
- ✅ Découverte via registry
- ✅ Billing automatique
- ✅ Protocole standardisé pour tous les agents
- ✅ **L'agent interne peut lui-même appeler d'autres agents**

Exemple de composition :

```python
# L'agent BNP peut lui-même utiliser d'autres agents
class BNPKYCAgent(Agent):
    def verify_identity(self, customer_id):
        # Appelle un agent tiers de détection de fraude
        fraud_check = market.call(
            "fraud-detection-ai",
            customer_id=customer_id
        )

        # Combine avec données internes
        internal_data = self.bnp_db.query(customer_id)

        # Retourne résultat synthétisé
        return self.synthesize(fraud_check, internal_data)
```

= Composabilité infinie d'agents

4. Modèle économique

4.1 Sources de revenus

4.1.1 Commissions Marketplace

Outils payants créés par la communauté :

- Commission : 15-20%
- Revenue share : 80-85% pour le créateur
- Exemple : Outil à 0.05€/call → plateforme gagne 0.01€

Agents payants :

- Commission : 20-25%
- Revenue share : 75-80% pour le créateur
- Exemple : Agent à 2€/call → plateforme gagne 0.40-0.50€

Volume projeté (Année 1) :

- 50,000 appels payants/mois
- Prix moyen : 0.20€/appel
- Commission moyenne : 20%
- Revenu mensuel : 50,000 × 0.20 × 0.20 = 2,000€/mois

4.1.2 Gateway Enterprise (B2B)

Pour grandes entreprises voulant exposer leurs agents :

- Starter : 500€/mois (jusqu'à 10,000 calls/mois)
- Business : 2,000€/mois (jusqu'à 100,000 calls/mois)
- Enterprise : Sur devis (volume illimité, SLA, support dédié)

Inclus :

- Infrastructure de connexion sécurisée
- Dashboard de monitoring
- Analytics avancés
- Support technique

- Compliance (RGPD, SOC2)

Projection (Année 2) :

- 10 clients Enterprise à 2,000€/mois = 20,000€/mois

4.1.3 Services Premium (Phase 3)

- Hébergement managé pour créateurs (vs self-hosted)
- Support prioritaire
- Analytics avancés
- White-label marketplace
- Consulting & intégration

4.2 Structure tarifaire utilisateurs

Free Tier :

- Tous les outils gratuits (illimité)
- 100 appels/mois sur outils payants
- API access
- Documentation complète

Starter - 49€/mois :

- 5,000 appels/mois inclus
- Au-delà : pay-as-you-go
- Support email
- Analytics basiques

Pro - 199€/mois :

- 50,000 appels/mois inclus
- Analytics avancés
- Support prioritaire
- SLA 99.9%

Enterprise - Sur devis :

- Volume personnalisé

- SLA garanti
- Support dédié
- Facturation annuelle
- Déploiement on-premise possible

4.3 Projections financières

Année 1 (Bootstrap / MVP)

Revenus :

- Commissions marketplace : 2,000€/mois × 12 = 24,000€
- Abonnements utilisateurs : 1,500€/mois × 12 = 18,000€
- **Total : 42,000€**

Coûts :

- Infrastructure (AWS/GCP) : 500€/mois = 6,000€
- Développement (1 fondateur temps plein) : 0€ (sweat equity)
- Marketing : 5,000€
- Légal & admin : 3,000€
- **Total : 14,000€**

Résultat : +28,000€ (autofinancé)

Année 2 (Croissance)

Revenus :

- Commissions : 30,000€/mois × 12 = 360,000€
- Gateway Enterprise : 20,000€/mois × 12 = 240,000€
- **Total : 600,000€**

Coûts :

- Infrastructure : 3,000€/mois = 36,000€
- Équipe (3 personnes) : 180,000€
- Marketing : 50,000€
- Opérationnel : 30,000€
- **Total : 296,000€**

Résultat : +304,000€ (profitable)

---

5. Roadmap détaillée

Phase 1 : MVP (Mois 1-3)

Objectifs :

- Valider le product-market fit
- Prouver que des créateurs publieront des agents
- Générer les premiers revenus

Livrables :

Semaine 1-2 : Infrastructure de base

- [ ] API Gateway minimal (REST)
- [ ] SDK Python v0.1 (call, publish)
- [ ] Registry PostgreSQL
- [ ] 5 outils gratuits hardcodés (calculatrice, météo, etc.)

Semaine 3-4 : Marketplace minimal

- [ ] Landing page + catalogue
- [ ] Page de détail outil/agent
- [ ] Stripe integration (paiement)
- [ ] Creator onboarding flow

Semaine 5-6 : Premiers agents

- [ ] 3 agents développés en interne
- [ ] 5 freelances recrutés pour créer 1 agent chacun
- [ ] Documentation : "Créer et publier votre premier agent"

Semaine 7-8 : Beta privée

- [ ] 20 early adopters invités
- [ ] Feedback loop intensif
- [ ] Monitoring et analytics basiques

Semaine 9-12 : Itération et stabilisation

- [ ] Correction bugs critiques
- [ ] Amélioration UX basée sur feedback
- [ ] Ajout de 10 outils gratuits supplémentaires

- [ ] SDK TypeScript v0.1

Critères de succès :

- ✅ 10 agents publiés par des créateurs externes
- ✅ 50 développeurs utilisateurs actifs
- ✅ 1,000€ de revenus générés
- ✅ NPS > 40

Phase 2 : Public Launch & Adoption (Mois 4-6)

Objectifs :

- Créer du buzz et acquérir massivement
- Construire la communauté de créateurs
- Atteindre 100 agents sur la marketplace

Livrables :

Mois 4 : Lancement public

- [ ] ProductHunt launch
- [ ] Campagne marketing (Reddit, HN, Twitter)
- [ ] Programme d'affiliation créateurs
- [ ] Content marketing (blog posts, tutorials)

Mois 5 : Communauté

- [ ] Discord serveur (créateurs + utilisateurs)
- [ ] Weekly office hours
- [ ] Concours "Best Agent of the Month" (1,000€ prize)
- [ ] Case studies et success stories

Mois 6 : Features avancées

- [ ] Agent-to-agent calling (beta)
- [ ] Analytics dashboard pour créateurs
- [ ] Système de rating et reviews
- [ ] API versioning

Critères de succès :

- ✅ 100 agents publiés
- ✅ 500 développeurs utilisateurs actifs

- ✅ 50 créateurs actifs
- ✅ 5,000€ MRR
- ✅ Featured dans 2-3 médias tech

Phase 3 : MCP Integration & Standardisation (Mois 7-9)

Objectifs :

- Adopter le protocole MCP officiellement
- Positionner comme "MCP-compliant marketplace"
- Permettre aux créateurs d'héberger leurs propres MCP servers

Livrables :

Mois 7 : MCP Core

- [ ] Refactor SDK pour supporter MCP nativement
- [ ] Migration des agents existants vers format MCP
- [ ] Documentation MCP pour créateurs
- [ ] Compatibilité avec Claude Desktop / autres clients MCP

Mois 8 : External MCP Servers

- [ ] Support pour créateurs hébergeant leur propre serveur
- [ ] Gateway proxy vers MCP servers externes
- [ ] Monitoring et health checks
- [ ] Auto-discovery de nouveaux servers

Mois 9 : Ecosystem

- [ ] Marketplace de MCP servers publics
- [ ] Contribution aux specs MCP (si pertinent)
- [ ] Partenariats avec autres outils MCP-compatibles
- [ ] SDK Go v1.0

Critères de succès :

- ✅ 100% des agents MCP-compliant
- ✅ 20 MCP servers externes connectés
- ✅ Mentionné dans la doc officielle MCP (si possible)

Phase 4 : Enterprise & Security (Mois 10-12)

Objectifs :

- Lancer Gateway-as-a-Service pour entreprises
- Renforcer sécurité et compliance
- Première levée de fonds ou rentabilité

Livrables :

Mois 10 : Enterprise features

- [ ] Gateway Enterprise (infra dédiée)
- [ ] SSO (Single Sign-On)
- [ ] RBAC (Role-Based Access Control)
- [ ] Audit logs complets
- [ ] SLA garantis

Mois 11 : Security & Compliance

- [ ] Sandbox renforcé (conteneurs isolés)
- [ ] Certification SOC2 (en cours)
- [ ] GDPR compliance toolkit
- [ ] Penetration testing
- [ ] Bug bounty program

Mois 12 : Sales B2B

- [ ] 5 POCs avec grandes entreprises
- [ ] Sales deck et case studies
- [ ] Support dédié entreprise
- [ ] Pricing personnalisé

Critères de succès :

- ✅ 3 clients Enterprise signés
- ✅ 20,000€ MRR
- ✅ Certification sécurité en cours
- ✅ 200 agents sur marketplace

Phase 5 : Scale & Premium Services (Année 2)

Objectifs :

- Atteindre 500+ agents
- Développer services premium
- Expansion internationale

Livrables :

Q1 Année 2 :

- [ ] Multi-cloud deployment (AWS + GCP)
- [ ] Agent-to-Agent P2P (WebRTC)
- [ ] White-label marketplace (pour revendeurs)
- [ ] API rate limit augmenté (10M calls/jour)

Q2 Année 2 :

- [ ] Expansion géographique (US, Asie)
- [ ] Support multilingue (EN, FR, ES, DE)
- [ ] Marketplace mobile app
- [ ] Advanced analytics & ML recommendations

Q3-Q4 Année 2 :

- [ ] Vertical marketplaces (healthcare, finance, legal)
- [ ] Consulting services
- [ ] Managed hosting premium
- [ ] Levée série A (si pertinent)

---

6. Exigences fonctionnelles

6.1 Pour les créateurs

FR-C1 : Publication simplifiée

- Un créateur doit pouvoir publier un agent en < 5 minutes
- Interface : CLI (`marketplace publish`) ou SDK (`publish(agent)`)
- Validation automatique du code
- Déploiement automatique

FR-C2 : Dashboard créateur

- Vue en temps réel des appels à ses agents
- Revenus générés (jour, semaine, mois)
- Métriques de performance (latency, success rate)
- Reviews et ratings utilisateurs
- Gestion des versions

FR-C3 : Paiements automatiques

- Virement automatique tous les mois
- Factures générées automatiquement
- Historique complet des transactions
- Support multi-devises (EUR, USD)

FR-C4 : Gestion des versions

- Possibilité de publier plusieurs versions
- Deprecation d'anciennes versions
- Rollback en cas de problème
- Changelog visible

6.2 Pour les utilisateurs

FR-U1 : Découverte

- Recherche par mots-clés
- Filtres (catégorie, prix, rating, latency)
- Recommandations personnalisées
- Trending agents

FR-U2 : Utilisation

- SDK simple dans 3 langages (Python, TS, Go)
- Documentation auto-générée pour chaque agent
- Playground pour tester sans code
- Exemples d'utilisation

FR-U3 : Facturation transparente

- Estimation du coût avant appel
- Alertes si budget dépassé
- Factures détaillées
- Contrôle des dépenses par agent

FR-U4 : Support

- Documentation complète
- Tutoriels vidéo
- Support chat (pour plans payants)
- Forum communautaire

6.3 Fonctionnalités marketplace

FR-M1 : Catalogue

- Listing de tous les agents/outils
- Pagination efficace
- Preview des capacités
- Démo live

FR-M2 : Rating & Reviews

- Système 5 étoiles
- Reviews textuelles
- Vérification "Utilisateur vérifié"
- Modération

FR-M3 : Analytics publiques

- Nombre d'appels (30 derniers jours)
- Latence moyenne
- Success rate
- Tendance de popularité

FR-M4 : Catégorisation

- Par domaine (finance, santé, legal, etc.)
- Par type (outil simple, agent complexe)
- Par pricing (gratuit, payant)
- Tags personnalisés

---

7. Exigences non-fonctionnelles

7.1 Performance

NFR-P1 : Latency

- API gateway : < 50ms (P95)
- Tool call simple : < 200ms (P95)
- Agent call complexe : < 2s (P95)

NFR-P2 : Throughput

- Support 100,000 appels/jour (Phase 1)
- Support 1M appels/jour (Phase 2)
- Support 10M appels/jour (Phase 3)

NFR-P3 : Scalability

- Auto-scaling horizontal
- Pas de single point of failure
- Geographic distribution (multi-region)

7.2 Sécurité

NFR-S1 : Authentication

- API keys avec rotation automatique
- OAuth2 pour intégrations tierces
- Rate limiting par utilisateur

NFR-S2 : Isolation

- Agents exécutés en sandbox complet
- Pas d'accès réseau non autorisé
- Limites strictes (CPU, RAM, temps)

NFR-S3 : Data Protection

- Encryption at rest (AES-256)
- Encryption in transit (TLS 1.3)
- GDPR compliant

- Logs anonymisés

NFR-S4 : Audit

- Tous les appels loggés
- Traçabilité complète
- Retention 90 jours minimum
- Export pour utilisateurs Enterprise

7.3 Fiabilité

NFR-R1 : Uptime

- SLA : 99.9% uptime (Free/Starter)
- SLA : 99.95% uptime (Pro)
- SLA : 99.99% uptime (Enterprise)

NFR-R2 : Error Handling

- Retry automatique (3 tentatives)
- Fallback gracieux
- Messages d'erreur clairs
- Status page publique

NFR-R3 : Monitoring

- Alertes temps réel (PagerDuty)
- Dashboards (Grafana)
- Logs centralisés (ELK ou DataDog)
- Tracing distribué (Jaeger)

7.4 Extensibilité

NFR-E1 : Modularity

- Architecture microservices
- APIs versionnées
- Backward compatibility garantie

NFR-E2 : Plugin System

- Support de nouveaux protocoles
- Custom authenticators
- Custom billing rules

---

8. Tech Stack

8.1 Backend

API Gateway :

- **Language :** Node.js (Express) ou Go
- **Rationale :** Performance + ecosystem riche

Database :

- **Primary :** PostgreSQL (metadata, billing, users)
- **Cache :** Redis (hot data, rate limiting)
- **Search :** Elasticsearch ou Algolia (marketplace search)

Message Queue :

- **Queue :** RabbitMQ ou AWS SQS
- **Usage :** Async processing, billing jobs

Execution :

- **Serverless :** AWS Lambda ou Google Cloud Run
- **Containers :** Docker + Kubernetes (si volumes élevés)

Storage :

- **Code Bundles :** S3 ou GCS
- **Logs :** CloudWatch ou Stackdriver

8.2 Frontend

Marketplace Website :

- **Framework :** Next.js 14 (React)
- **Styling :** TailwindCSS

- **State :** Zustand ou React Query
- **Analytics :** Plausible ou Mixpanel

Creator Dashboard :

- **Charts :** Recharts ou Chart.js
- **Tables :** TanStack Table

8.3 SDK

Languages :

- **Python** (priorité 1) : 90% des devs IA
- **TypeScript** (priorité 2) : Full-stack devs
- **Go** (priorité 3) : Infrastructure devs

Features :

- Auto-retry
- Caching
- Error handling
- Type safety (TypeScript, Python type hints)

8.4 DevOps

CI/CD :

- GitHub Actions
- Automated testing (Jest, Pytest)
- Deployment : Terraform + Ansible

Monitoring :

- Prometheus + Grafana
- DataDog (APM)
- Sentry (error tracking)

Infrastructure :

- **Cloud :** AWS (initial) puis multi-cloud
- **CDN :** CloudFront ou Cloudflare

- **DNS :** Route53

8.5 Security

Tools :

- Snyk (dependency scanning)
- OWASP ZAP (penetration testing)
- HashiCorp Vault (secrets management)

Compliance :

- SOC2 Type II (objectif Année 2)
- GDPR toolkit intégré
- ISO 27001 (optionnel)

---

10. Cas d'usage Agent-to-Agent (Exemples concrets)

10.1 Finance : Agent Conseiller Financier Personnel

Problème aujourd'hui : Un développeur veut créer un agent qui donne des conseils financiers personnalisés. Il doit :

- Intégrer l'API de 5 banques différentes (chacune avec son format)
- Intégrer une API de bourse temps réel
- Intégrer une API de calcul fiscal
- Intégrer une API de scoring crédit = **6 mois de développement d'intégrations**

Avec notre protocole :

```
python
```

```
class PersonalFinancialAdvisor(Agent):
    def give_advice(self, user_id):
        # Récupère données bancaires
        accounts = market.call("bnp-banking-agent",
                               action="get_accounts",
                               user_id=user_id)

        # Analyse portefeuille boursier
        stocks_analysis = market.call("stock-analyzer-agent",
                                      portfolio=accounts.stocks)

        # Optimisation fiscale
        tax_tips = market.call("tax-optimization-agent",
                               income=accounts.total_income,
                               country="FR")

        # Scoring crédit pour opportunités
        credit_score = market.call("credit-scoring-agent",
                                   history=accounts.history)

        # Synthèse personnalisée
        return self.generate_personalized_advice(
            accounts, stocks_analysis, tax_tips, credit_score
        )

publish(PersonalFinancialAdvisor, pricing={"per_call": 5.00})
```

Résultat :

- ✅ Développé en 2 jours au lieu de 6 mois
- ✅ 4 agents spécialisés appelés automatiquement
- ✅ Billing automatique (fintech paie chaque agent)
- ✅ Mise à jour automatique si un agent évolue
- ✅ Fallback si un agent est down

10.2 Santé : Agent Diagnostic Médical

Problème aujourd'hui : Un hôpital veut un agent qui aide au diagnostic. Il doit :

- Accéder au dossier patient (système interne hôpital)
- Analyser les images médicales (IA spécialisée externe)
- Vérifier les interactions médicamenteuses (base de données pharma)
- Consulter la littérature médicale récente (PubMed) = **Impossible sans développement custom massif**

Avec notre protocole :

```python
class MedicalDiagnosticAgent(Agent):
    def diagnose(self, patient_id, symptoms):
        # L'hôpital expose son agent de dossier médical
        medical_history = market.call("hospital-ehr-agent",
                                      patient_id=patient_id,
                                      consent_token=symptoms.consent)

        # Analyse image radio
        if symptoms.has_scan:
            scan_analysis = market.call("radiology-ai-agent",
                                        image=symptoms.scan,
                                        type="chest_xray")

        # Check interactions médicaments
        drug_check = market.call("pharma-interaction-agent",
                                 current_meds=medical_history.medications,
                                 proposed_med=self.preliminary_diagnosis)

        # Recherche littérature
        recent_studies = market.call("pubmed-research-agent",
                                     condition=self.preliminary_diagnosis)

        return self.synthesize_diagnosis(
            medical_history, scan_analysis, drug_check, recent_studies
        )
```

Avantages :

- ✅ **Confidentialité** : Le dossier patient reste dans l'hôpital, seules les infos nécessaires sont partagées
- ✅ **Compliance** : Chaque agent gère sa propre conformité (HIPAA, RGPD)
- ✅ **Spécialisation** : Chaque IA fait ce qu'elle fait de mieux
- ✅ **Composition** : L'agent diagnostic orchestre plusieurs experts

10.3 Juridique : Agent Analyse de Contrats

Scénario : Une startup veut un agent qui analyse automatiquement tous les contrats avant signature.

```python
```

```
class ContractAnalysisAgent(Agent):
    def analyze_contract(self, contract_pdf):
        # 1. OCR du PDF
        text = market.call("ocr-legal-agent",
                           document=contract_pdf,
                           language="fr")

        # 2. Détection de clauses dangereuses
        risky_clauses = market.call("legal-risk-detector-agent",
                                    contract_text=text.content)

        # 3. Vérification conformité RGPD
        gdpr_check = market.call("gdpr-compliance-agent",
                                 contract=text.content)

        # 4. Benchmark avec contrats similaires
        market_comp = market.call("contract-benchmark-agent",
                                  type=text.contract_type,
                                  industry="SaaS")

        # 5. Traduction si nécessaire
        if text.language != "en":
            translation = market.call("legal-translation-agent",
                                      text=risky_clauses.summary,
                                      target_lang="en")

        return {
            "risk_score": self.calculate_risk(risky_clauses),
            "gdpr_compliant": gdpr_check.compliant,
            "vs_market": market_comp.comparison,
            "recommendation": "SIGN" or "NEGOTIATE" or "REJECT"
        }
```

Cas d'usage réel :

- Une PME signe 50 contrats/mois
- Coût avocat : 500€/contrat × 50 = 25,000€/mois
- Coût avec l'agent : 10€/contrat × 50 = 500€/mois
- **Économie : 24,500€/mois**

10.4 E-commerce : Agent Personal Shopper

Scénario : Un site e-commerce veut un assistant shopping qui cherche sur tous les sites concurrents.

```python
class PersonalShopperAgent(Agent):
    def find_best_product(self, user_query, user_preferences):
        # Recherche sur 10 marketplaces via leurs agents
        results = []

        for marketplace in ["amazon", "cdiscount", "fnac", "alibaba"]:
            agent_name = f"{marketplace}-product-agent"
            products = market.call(agent_name,
                                   query=user_query,
                                   filters=user_preferences)
            results.append(products)

        # Comparaison de prix
        price_comp = market.call("price-comparison-agent",
                                 products=results)

        # Vérification avis (détection fake reviews)
        reviews = market.call("review-authenticity-agent",
                              products=price_comp.top_10)

        # Recommandation personnalisée basée sur historique
        recommendation = market.call("recommendation-engine-agent",
                                     user_id=user_preferences.user_id,
                                     options=reviews.trusted_products)

        return recommendation.best_match
```

Impact :

- ✅ 10 marketplaces interrogées automatiquement
- ✅ Comparaison de prix en temps réel
- ✅ Détection de faux avis
- ✅ Recommandation personnalisée
- **Le tout en < 2 secondes**

10.5 Logistique : Agent Optimisation de Livraison

Scénario : Un e-commerçant veut optimiser ses livraisons en appelant dynamiquement le meilleur transporteur.

```python
```

```
class DeliveryOptimizerAgent(Agent):
    def optimize_delivery(self, order):
        # Appelle les agents de 5 transporteurs
        quotes = []
        for carrier in ["ups", "fedex", "dhl", "colissimo", "chronopost"]:
            quote = market.call(f"{carrier}-quote-agent",
                                from_addr=order.warehouse,
                                to_addr=order.customer_addr,
                                weight=order.weight,
                                dimensions=order.dimensions)
            quotes.append(quote)

        # Prédiction délai réel (basé sur ML)
        delays_prediction = market.call("delivery-delay-predictor-agent",
                                        quotes=quotes,
                                        season="christmas",
                                        weather=self.get_weather())

        # Scoring de fiabilité des transporteurs
        reliability = market.call("carrier-reliability-agent",
                                  carriers=[q.carrier for q in quotes],
                                  destination=order.customer_addr)

        # Calcul empreinte carbone
        carbon = market.call("carbon-footprint-agent",
                             routes=[q.route for q in quotes])

        # Décision optimale (prix + délai + fiabilité + eco)
        best_option = self.decide(quotes, delays_prediction,
                                  reliability, carbon,
                                  user_priority=order.customer.preferences)

        # Réservation automatique
        booking = market.call(f"{best_option.carrier}-booking-agent",
                              quote_id=best_option.quote_id)

        return booking
```

Résultat :

- ✅ 5 transporteurs comparés automatiquement
- ✅ Prix, délai, fiabilité, impact écologique pris en compte
- ✅ Réservation automatique chez le meilleur
- ✅ **Économie moyenne : 15% sur coûts de livraison**

10.6 Ce que ces exemples démontrent

1. Composabilité Un agent complexe = composition de 5-10 agents spécialisés

2. Spécialisation Chaque agent fait une chose et la fait très bien

3. Marché biface

- Créateurs vendent des agents spécialisés
- Développeurs créent des agents complexes en les combinant

4. Effet réseau Plus d'agents → Plus de compositions possibles → Plus de valeur

5. Time-to-market révolutionnaire Ce qui prenait 6 mois prend maintenant 2 jours

---

9.1 Acquisition créateurs

Canaux :

1. Outbound direct

- Identifier 100 créateurs potentiels (GitHub, Kaggle, ArXiv)
- Email personnalisé avec offre VIP
- Commission 0% pendant 3 mois

2. Content marketing

- Blog posts : "How I built an AI agent in 2 hours and made $5K"
- Tutorials : "From Jupyter Notebook to Production in 10 minutes"
- YouTube : Walkthrough vidéos

3. Community

- Discord serveur actif
- Weekly AMAs
- Concours mensuels avec prizes

4. Partnerships

- Universités (PhD programs)
- Bootcamps IA
- Accélérateurs startups

9.2 Acquisition utilisateurs

Canaux :

1. Product-led growth

- Free tier généreux
- Outils gratuits de qualité
- Viralité (chaque agent = acquisition)

2. Developer marketing

- ProductHunt launch
- HackerNews posts
- Reddit (r/MachineLearning, r/artificial)
- Dev.to articles

**3. SEO