Product Requirements Document

Intelligent Subscription Management MCP for Mistral Chat

Version: 1.0

Date: September 2025

Status: Draft

---

1. Executive Summary

1.1 Purpose

Develop a Model Context Protocol (MCP) server that enables Mistral Chat to help users track, analyze, and optimize their recurring subscriptions and expenses. This MCP will provide intelligent insights, automated detection, and actionable recommendations to save money and prevent subscription waste.

1.2 Problem Statement

- Users lose an average of €150-300/month on forgotten or underutilized subscriptions
- No centralized, intelligent system to track all recurring payments
- Difficult to detect price increases and optimize subscription overlap
- Manual tracking is time-consuming and error-prone

1.3 Solution Overview

A Python-based MCP server that integrates with Mistral Chat to provide:

- Automated subscription detection and tracking
- Usage analysis and cost optimization
- Proactive alerts and recommendations
- Privacy-first local data storage

---

2. Technical Architecture

2.1 Technology Stack

- **Language:** Python 3.11+
- **MCP Framework:** `mcp` Python SDK

- **Database:** SQLite (local storage)
- **Email Parsing:** `imaplib`, `email`, `beautifulsoup4`
- **Data Analysis:** `pandas`, `numpy`
- **ML/Pattern Recognition:** `scikit-learn`
- **Scheduling:** `APScheduler`
- **API Framework:** FastAPI (for MCP server)
- **Encryption:** `cryptography` library

2.2 System Components

```
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│  Mistral Chat   │────▶│   MCP Server    │────▶│  Local SQLite   │
│                 │◀────│    (Python)     │◀────│    Database     │
└─────────────────┘     └─────────────────┘     └─────────────────┘
         │
         ▼
┌─────────────────┐
│ External Services│
│ - Email APIs    │
│ - Bank CSV import│
└─────────────────┘
```

2.3 Data Model

```python
```

```
# Core entities
Subscription {
  id: UUID
  name: string
  provider: string
  cost: decimal
  currency: string
  billing_cycle: enum (monthly, yearly, weekly)
  category: string
  status: enum (active, paused, cancelled, trial)
  start_date: date
  next_billing_date: date
  last_used: date
  usage_frequency: json
  payment_method: string
  notes: text
  created_at: timestamp
  updated_at: timestamp
}

Transaction {
  id: UUID
  subscription_id: UUID
  amount: decimal
  date: date
  status: enum (pending, completed, failed)
  source: enum (email, csv, manual)
  raw_data: json
}

Alert {
  id: UUID
  subscription_id: UUID
  type: enum (unused, price_increase, renewal, duplicate)
  severity: enum (low, medium, high)
  message: text
  action_taken: boolean
  created_at: timestamp
}

UsagePattern {
  id: UUID
  subscription_id: UUID
```

```
    period: date_range
    usage_count: integer
    usage_hours: decimal
    cost_per_use: decimal
}
```

---

3. Core Features & Requirements

3.1 Subscription Detection & Import

3.1.1 Email Scanning

- **Requirement:** Automatically detect subscriptions from email receipts
- **Implementation:**

```python
- Connect to Gmail/Outlook via IMAP
- Search for keywords: "subscription", "recurring", "monthly", "renewal"
- Parse email headers and body for transaction details
- Extract: service name, amount, date, frequency
```

- **Acceptance Criteria:**
  - 90% accuracy in detecting known subscription services
  - Support for top 100 subscription services
  - Process 1000 emails in < 30 seconds

3.1.2 CSV Bank Import

- **Requirement:** Import transactions from bank statements
- **Implementation:**
  - Support common bank CSV formats
  - Pattern matching for recurring transactions
  - ML-based detection of subscription patterns
- **Acceptance Criteria:**
  - Support for 5+ major bank formats
  - Detect recurring patterns with 85% accuracy

3.2 Analysis & Optimization Engine

3.2.1 Usage Analysis

- **Requirement:** Track and analyze subscription usage
- **Metrics to track:**
  - Cost per use
  - Days since last use
  - Usage trend (increasing/decreasing)
  - Time spent (when applicable)
- **Implementation:**

```python
def analyze_subscription_value(subscription_id):
    # Calculate ROI based on usage patterns
    # Compare with category averages
    # Generate optimization score
```

3.2.2 Duplicate Detection

- **Requirement:** Identify overlapping services
- **Logic:**
  - Category-based overlap (multiple streaming services)
  - Feature comparison
  - Cost-benefit analysis
- **Acceptance Criteria:**
  - Detect 95% of obvious duplicates
  - Provide clear explanation of overlap

3.3 MCP Tools Implementation

```python
```

```
# Core MCP tools to expose

@mcp_tool
async def scan_subscriptions(source: str, credentials: dict):
    """Scan for subscriptions from email or bank data"""

@mcp_tool
async def add_subscription(name: str, cost: float, cycle: str, category: str):
    """Manually add a subscription"""

@mcp_tool
async def analyze_spending():
    """Generate spending analysis and insights"""

@mcp_tool
async def get_recommendations():
    """Get optimization recommendations"""

@mcp_tool
async def set_alert(subscription_id: str, alert_type: str, threshold: dict):
    """Set custom alerts for subscriptions"""

@mcp_tool
async def cancel_subscription(subscription_id: str, generate_email: bool):
    """Help cancel a subscription"""

@mcp_tool
async def find_alternatives(subscription_id: str):
    """Find cheaper alternatives"""

@mcp_tool
async def export_report(format: str, period: str):
    """Export subscription report"""
```

3.4 Alert System

3.4.1 Alert Types

- **Unused Alert:** Service not used for X days
- **Price Increase:** Detected price change
- **Renewal Alert:** X days before renewal
- **Trial Ending:** Trial period about to convert

- **Duplicate Alert:** Similar service detected
- **Optimization Alert:** Better deal available

3.4.2 Alert Configuration

```python
AlertConfig {
  unused_threshold_days: 30
  renewal_reminder_days: 7
  price_increase_notification: immediate
  optimization_check_frequency: weekly
}
```

3.5 Privacy & Security

3.5.1 Data Protection

- **Requirement:** All data stored locally with encryption
- **Implementation:**
  - SQLite database with SQLCipher encryption
  - No cloud storage unless explicitly requested
  - Credentials stored in system keyring
  - Option to redact sensitive data in exports

3.5.2 Data Retention

- Transaction history: 24 months
- Usage patterns: 12 months
- Alerts: 6 months
- User configurable retention policies

---

4. User Interactions & Flows

4.1 Initial Setup Flow

```
User: "Help me track my subscriptions"
Mistral: "I'll help you set up subscription tracking. Would you like me to:
1. Scan your emails for subscriptions
```

```
2. Import a bank statement CSV
3. Manually add subscriptions"

User: "Scan my emails"
Mistral: [Uses scan_subscriptions tool]
"I found 23 subscriptions totaling €487/month. Here's the breakdown..."
```

4.2 Regular Analysis Flow

```
User: "Analyze my subscriptions"
Mistral: [Uses analyze_spending tool]
"Analysis complete:
- Total monthly spend: €487
- Unused services (3): €67/month
- Potential duplicates: €45/month
- Optimization opportunities: Save €143/month"
```

4.3 Cancellation Assistance

```
User: "Help me cancel Adobe"
Mistral: [Uses cancel_subscription tool]
"I've prepared a cancellation email for Adobe. I've also found
Canva Pro as an alternative at €12/month (saving €12/month)."
```

---

5. Development Phases

Phase 1: Core Infrastructure (Week 1-2)

- [ ] Set up MCP server structure
- [ ] Implement SQLite database schema
- [ ] Create basic CRUD operations
- [ ] Develop MCP tool interfaces

Phase 2: Detection Engine (Week 3-4)

- [ ] Email parsing implementation
- [ ] CSV import functionality
- [ ] Pattern recognition algorithm
- [ ] Subscription service database

Phase 3: Analysis Features (Week 5-6)

- [ ] Usage tracking system
- [ ] Cost analysis algorithms
- [ ] Duplicate detection logic
- [ ] Recommendation engine

Phase 4: Alerts & Automation (Week 7-8)

- [ ] Alert system implementation
- [ ] Scheduled job system
- [ ] Email generation for cancellations
- [ ] Report generation

Phase 5: Testing & Optimization (Week 9-10)

- [ ] Unit testing (>80% coverage)
- [ ] Integration testing with Mistral
- [ ] Performance optimization
- [ ] Security audit

---

6. Success Metrics

6.1 Technical KPIs

- Response time: < 2 seconds for analysis
- Detection accuracy: > 90%
- Database query performance: < 100ms
- Memory usage: < 200MB
- Uptime: 99.9%

6.2 User Value KPIs

- Average savings identified: > €100/month
- Subscription detection rate: > 85%
- False positive rate: < 5%
- User engagement: Weekly active usage
- Actionable recommendations: > 3 per analysis

---

7. Testing Strategy

7.1 Unit Tests

```python
# Example test cases
test_email_parser_gmail()
test_email_parser_outlook()
test_pattern_detection_monthly()
test_duplicate_detection_streaming()
test_cost_calculation_yearly()
test_alert_generation_unused()
```

7.2 Integration Tests

- MCP server communication with Mistral
- Database operations under load
- Email API connections
- Concurrent user sessions

7.3 Test Data

- Mock email datasets (100+ examples)
- Sample bank CSVs (10+ formats)
- Subscription service catalog (500+ services)

---

8. Configuration & Deployment

8.1 Environment Variables

```env
MCP_SERVER_PORT=3000
DATABASE_PATH=~/.subscription_manager/data.db
ENCRYPTION_KEY=<auto-generated>
LOG_LEVEL=INFO
MAX_EMAIL_SCAN=1000
ANALYSIS_CACHE_TTL=3600
```

8.2 MCP Server Configuration

```json
{
  "mcpServers": {
    "subscription-manager": {
      "command": "python",
      "args": ["-m", "subscription_mcp.server"],
      "env": {
        "DATABASE_PATH": "~/.subscription_manager/data.db"
      }
    }
  }
}
```

8.3 Installation Steps

```bash
# Clone repository
git clone https://github.com/yourusername/subscription-mcp

# Install dependencies
pip install -r requirements.txt

# Initialize database
python -m subscription_mcp.init_db

# Run MCP server
python -m subscription_mcp.server

# Configure Mistral Chat
# Add MCP server configuration to Mistral settings
```

---

9. Future Enhancements

Version 2.0 Features

- Family sharing optimization
- Business subscription management
- Multi-currency support
- Subscription marketplace integration

- API for third-party apps
- Mobile app companion
- Browser extension for detection
- Negotiation bot integration
- Community price sharing
- Tax deduction tracking

Long-term Vision

- AI-powered usage prediction
- Automated negotiation system
- Group buying coordination
- Subscription lending/sharing
- Corporate expense integration

---

10. Appendices

A. Subscription Service Database Schema

```json
{
  "netflix": {
    "patterns": ["netflix", "nflx"],
    "category": "streaming",
    "billing_cycles": ["monthly"],
    "price_ranges": {"min": 7.99, "max": 19.99}
  }
}
```

B. Email Parser Patterns

```python
EMAIL_PATTERNS = {
    "amount": r"(?:€|\$|£)\s*(\d+(?:\.\d{2})?)",
    "date": r"(\d{1,2}[/-]\d{1,2}[/-]\d{2,4})",
    "service": r"(?:from|by|at)\s+(\w+(?:\s+\w+)?)"
}
```

C. API Response Examples

json

```json
{
  "tool": "analyze_spending",
  "result": {
    "total_monthly": 487.50,
    "by_category": {
      "streaming": 65.97,
      "software": 234.00,
      "storage": 45.00
    },
    "recommendations": [
      {
        "action": "cancel",
        "service": "Adobe Creative Cloud",
        "reason": "unused_90_days",
        "savings": 54.99
      }
    ]
  }
}
```

Document Control:

- Author: Product Team
- Review: Engineering Lead
- Approval: Product Manager
- Last Updated: September 2025