# Manager Agent - Trung tâm điều phối hệ thống

## 🎯 Tổng quan

**Manager Agent** là thành phần trung tâm và điểm vào chính của hệ thống MAS-Planning. Nó đóng vai trò như một conductor trong dàn nhạc, orchestrating và điều phối hoạt động của tất cả các specialized agents khác.

## 🏗️ Kiến trúc và Vị trí

graph TD  
 A[User Request] --> B[Manager Agent]  
 B --> C[Query Analysis]  
 B --> D[Routing Logic]  
 B --> E[Agent Delegation]  
   
 E --> F[Plan Agent]  
 E --> G[Meta Agent]   
 E --> H[Tool Agent]  
 E --> I[Direct Response]  
   
 F --> J[Plan Creation/Execution]  
 G --> K[Task Analysis]  
 H --> L[Device Control]  
   
 J --> M[Response Formatting]  
 K --> M  
 L --> M  
 I --> M  
   
 M --> N[Final Response]  
   
 style B fill:#e1f5fe,stroke:#01579b,stroke-width:3px  
 style C fill:#f3e5f5  
 style D fill:#e8f5e8  
 style E fill:#fff3e0

## 🎯 Vai trò và Trách nhiệm

### **Primary Responsibilities**

1. **Query Analysis & Classification**
   * Phân tích ý định người dùng từ natural language input
   * Phân loại query types (planning, control, information, selection)
   * Context-aware analysis sử dụng conversation history
2. **Intelligent Routing**
   * Quyết định agent nào phù hợp nhất cho từng request
   * Load balancing và resource optimization
   * Fallback routing khi primary analysis fails
3. **Multi-Agent Orchestration**
   * Coordinating workflows giữa specialized agents
   * Managing agent lifecycle và resource allocation
   * Handling complex multi-step processes
4. **Session Management**
   * Duy trì conversation context và history
   * Plan options caching cho user selections
   * State persistence across interactions
5. **Response Formatting**
   * Chuẩn hóa responses từ different agents
   * User-friendly output formatting
   * Error handling và graceful degradation

## 🧠 Core Components

### **1. Query Analysis Engine**

def analyze\_query(self, state: ManagerState) -> ManagerState:  
 """  
 Phân tích query với conversation context  
 """  
 # Build context từ chat history  
 messages = [SystemMessage(self.system\_prompt)]  
   
 # Add recent conversation history  
 if self.chat\_history.messages:  
 recent\_messages = self.chat\_history.messages[-6:]  
 for msg in recent\_messages:  
 # Convert to appropriate message format  
   
 # Add current query  
 messages.append(HumanMessage(f"Current User Query: {user\_input}"))  
   
 # LLM analysis  
 llm\_response = self.llm.invoke(messages)  
 reasoning\_result = extract\_manager\_response(llm\_response.content)

**Features:** - **Context-Aware Analysis**: Sử dụng 6 tin nhắn gần nhất làm context - **Confidence Scoring**: Đánh giá độ tin cậy của routing decisions - **Multi-factor Classification**: Kết hợp keyword matching và semantic analysis

### **2. Intelligent Routing System**

def \_context\_aware\_fallback\_routing(self, query: str) -> str:  
 """  
 Enhanced fallback routing với conversation context  
 """  
 # Priority 1: Explicit keywords  
 if any(word in query\_lower for word in ['create plan', 'plan']):  
 return 'plan'  
   
 # Priority 2: Conversation context analysis  
 if self.chat\_history.messages:  
 recent\_messages = self.chat\_history.messages[-4:]  
 # Analyze patterns in recent conversation  
   
 # Priority 3: Standard keyword-based routing  
 return self.\_fallback\_routing(query)

**Routing Strategy:** - **Plan Agent**: Plan creation, selection, automation requests - **Tool Agent**: Device control, immediate actions - **Meta Agent**: Analysis, reasoning, complex questions  
- **Direct Response**: Information queries, help requests

### **3. Agent Delegation Framework**

def route\_to\_agent(self, state: ManagerState) -> ManagerState:  
 """  
 Route và delegate requests to appropriate agents  
 """  
 agent\_type = state.get('agent\_type', 'direct')  
   
 if agent\_type == 'plan':  
 # Handle plan creation/selection with caching  
 if plan\_selection and self.\_cached\_plan\_options:  
 delegation\_result = self.plan\_agent.invoke(  
 user\_input,   
 selected\_plan\_id=plan\_selection,  
 plan\_options=self.\_cached\_plan\_options,  
 token=token  
 )  
 else:  
 delegation\_result = self.plan\_agent.invoke(user\_input, token=token)

**Delegation Features:** - **Lazy Loading**: Agents chỉ được khởi tạo khi cần thiết - **Resource Management**: Optimal memory và CPU usage - **Error Recovery**: Graceful handling khi agents fail - **Result Validation**: Ensuring output quality và format consistency

### **4. Session State Management**

class ManagerAgent:  
 def \_\_init\_\_(self, session\_id: str, conversation\_id: str):  
 # Chat history cho conversation context  
 self.chat\_history = RedisSupportChatHistory(  
 session\_id=session\_id,  
 conversation\_id=conversation\_id,  
 ttl=env.TTL\_SECONDS  
 )  
   
 # Cached plan options for user selections  
 self.\_cached\_plan\_options = {}

**State Components:** - **Conversation History**: Redis-backed persistent storage - **Plan Options Cache**: In-memory caching cho quick selections - **Session Context**: User preferences và interaction patterns - **Agent States**: Tracking specialized agent statuses

## 🔄 Workflow Processing

### **1. Standard Request Flow**

sequenceDiagram  
 participant U as User  
 participant M as Manager Agent  
 participant H as Chat History  
 participant A as Specialized Agent  
   
 U->>M: Input Request  
 M->>H: Retrieve Context  
 M->>M: Analyze Query  
 M->>M: Routing Decision  
 M->>A: Delegate to Agent  
 A->>M: Return Result  
 M->>M: Format Response  
 M->>H: Save Conversation  
 M->>U: Final Response

### **2. Plan Selection Flow**

sequenceDiagram  
 participant U as User  
 participant M as Manager Agent  
 participant C as Plan Cache  
 participant P as Plan Agent  
   
 U->>M: "Create smart home plan"  
 M->>P: Generate Plans  
 P->>M: 3 Priority Plans  
 M->>C: Cache Plan Options  
 M->>U: Present Options  
   
 U->>M: "Select plan 2"  
 M->>C: Retrieve Cached Plans  
 M->>P: Execute Selected Plan  
 P->>M: Execution Results  
 M->>U: Completion Report

### **3. Error Recovery Flow**

sequenceDiagram  
 participant M as Manager Agent  
 participant A as Agent  
 participant F as Fallback System  
   
 M->>A: Delegate Request  
 A-->>M: Error/Timeout  
 M->>M: Detect Failure  
 M->>F: Trigger Fallback  
 F->>M: Alternative Response  
 M->>M: Log Error  
 M->>User: Graceful Response

## 🧩 Integration Patterns

### **1. Lazy Agent Loading**

@property  
def plan\_agent(self):  
 """Lazy load Plan Agent with async initialization"""  
 if self.\_plan\_agent is None:  
 from template.agent.plan import PlanAgent  
 self.\_plan\_agent = PlanAgent(verbose=self.verbose)  
   
 # Async initialization in separate thread  
 def run\_async\_init():  
 loop = asyncio.new\_event\_loop()  
 asyncio.set\_event\_loop(loop)  
 try:  
 loop.run\_until\_complete(self.\_plan\_agent.init\_async())  
 finally:  
 loop.close()  
   
 thread = threading.Thread(target=run\_async\_init)  
 thread.start()  
 thread.join(timeout=10)

**Benefits:** - **Memory Efficiency**: Chỉ load agents khi cần - **Faster Startup**: Reduced initial loading time - **Resource Optimization**: Better CPU và memory utilization

### **2. Conversation Context Integration**

def analyze\_query(self, state: ManagerState) -> ManagerState:  
 """Context-aware query analysis"""  
 # Add conversation history for context  
 if self.chat\_history.messages:  
 recent\_messages = self.chat\_history.messages[-6:]  
   
 for msg in recent\_messages:  
 if 'Human' in msg.\_\_class\_\_.\_\_name\_\_:  
 messages.append(HumanMessage(f"Previous User: {content}"))  
 elif 'AI' in msg.\_\_class\_\_.\_\_name\_\_:  
 messages.append(SystemMessage(f"Previous AI: {content}"))

**Context Features:** - **Historical Awareness**: Understanding conversation flow - **Intent Persistence**: Maintaining user goals across interactions - **Adaptive Responses**: Tailoring replies based on conversation history

### **3. Token-based Authentication Flow**

def invoke(self, input\_data: Dict[str, Any]) -> Dict[str, Any]:  
 """Main entry point with token handling"""  
 state = {  
 'input': user\_message,  
 'token': input\_data.get('token', ''),  
 # ... other state data  
 }  
   
 # Pass token to agents that need authentication  
 if agent\_type == 'tool':  
 tool\_input = {  
 'input': user\_input,  
 'token': token  
 }  
 delegation\_result = self.tool\_agent.invoke(tool\_input)

## 📊 Performance Optimizations

### **1. Caching Strategy**

* **Plan Options Cache**: TTL-based in-memory storage
* **Conversation History**: Redis-backed với configurable TTL
* **Agent Results**: Temporary caching cho repeated queries
* **LLM Responses**: Selective caching cho common patterns

### **2. Resource Management**

* **Agent Lifecycle**: Lazy loading và resource cleanup
* **Memory Optimization**: Efficient state management
* **Connection Pooling**: Reusing database connections
* **Async Processing**: Non-blocking operations where possible

### **3. Response Time Optimization**

* **Parallel Processing**: Concurrent analysis where applicable
* **Preemptive Loading**: Predictive agent initialization
* **Response Streaming**: Immediate feedback cho long operations
* **Timeout Management**: Preventing hanging requests

## 🔒 Security Features

### **1. Input Validation**

def invoke(self, input\_data: Dict[str, Any]) -> Dict[str, Any]:  
 """Validate and sanitize inputs"""  
 if isinstance(input\_data, dict):  
 user\_message = input\_data.get('message', '') or input\_data.get('input', '')  
 else:  
 user\_message = str(input\_data)  
   
 # Input validation và sanitization  
 # Token validation  
 # Rate limiting checks

### **2. Secure Token Handling**

* **No Token Storage**: Tokens are passed through, never stored
* **Secure Transmission**: Encrypted communication channels
* **Token Validation**: Verification before delegation
* **Access Control**: Role-based permissions

### **3. Error Information Security**

* **Sanitized Error Messages**: No sensitive data in user responses
* **Secure Logging**: Protected error logs with rotation
* **Audit Trail**: Comprehensive access logging
* **Privacy Protection**: User data anonymization

## 📈 Monitoring và Diagnostics

### **1. Performance Metrics**

def invoke(self, input\_data: Dict[str, Any]) -> Dict[str, Any]:  
 """Main invoke với performance tracking"""  
 start\_time = time.time()  
   
 try:  
 result = self.graph.invoke(state)  
 execution\_time = time.time() - start\_time  
   
 if self.verbose:  
 logger.info(f"✅ Request processed in {execution\_time:.2f}s")  
   
 return {  
 'execution\_time': execution\_time,  
 'success': True,  
 # ... other response data  
 }

### **2. Health Checks**

def get\_system\_status(self) -> Dict[str, Any]:  
 """System health status"""  
 return {  
 'manager\_status': 'active',  
 'sub\_agents': {  
 'plan\_agent': self.\_plan\_agent is not None,  
 'meta\_agent': self.\_meta\_agent is not None,  
 'tool\_agent': self.\_tool\_agent is not None  
 },  
 'cached\_plans': len(self.\_cached\_plan\_options) > 0,  
 'capabilities': get\_agent\_capabilities()  
 }

### **3. Debugging Support**

* **Verbose Logging**: Detailed execution tracing
* **State Inspection**: Runtime state debugging
* **Agent Communication**: Inter-agent message tracking
* **Performance Profiling**: Execution bottleneck identification

## 🔮 Advanced Features

### **1. Adaptive Learning**

* **Pattern Recognition**: Learning từ user interaction patterns
* **Preference Modeling**: Adapting to user preferences over time
* **Success Rate Optimization**: Improving routing accuracy
* **Predictive Routing**: Anticipating user needs

### **2. Multi-Modal Support**

* **Text Processing**: Natural language understanding
* **Voice Integration**: Speech-to-text support ready
* **Visual Input**: Image analysis capabilities (future)
* **Gesture Recognition**: Motion-based commands (future)

### **3. Extensibility Framework**

* **Plugin Architecture**: Easy agent integration
* **Custom Routing Rules**: Configurable routing logic
* **External Service Integration**: API và webhook support
* **Custom Response Formatters**: Flexible output customization

## 🚀 Best Practices

### **1. Development Guidelines**

* **Separation of Concerns**: Clear responsibility boundaries
* **Error Handling**: Comprehensive exception management
* **Testing Strategy**: Unit và integration testing
* **Documentation**: Inline comments và API documentation

### **2. Performance Guidelines**

* **Resource Optimization**: Efficient memory và CPU usage
* **Caching Strategy**: Strategic data caching
* **Connection Management**: Proper resource cleanup
* **Monitoring**: Continuous performance tracking

### **3. Security Guidelines**

* **Input Validation**: Always validate user inputs
* **Secure Communication**: Use encrypted channels
* **Access Control**: Implement proper permissions
* **Audit Logging**: Comprehensive security logging

*Manager Agent là trái tim của hệ thống MAS-Planning, đảm bảo tất cả components hoạt động harmony để cung cấp experience tốt nhất cho users. Để hiểu sâu hơn về các specialized agents, vui lòng tham khảo tài liệu riêng của từng agent.*