Control Agent Architecture Document

1. System Overview

The Control Agent serves as the central intelligence hub for an AI-powered pizza business automation system. It coordinates all specialized AI agents, manages system-wide workflows, and ensures seamless integration between different components of the pizza business ecosystem.

The Control Agent is responsible for: - Orchestrating workflows between specialized agents (Manus, Roo, Windsurf, Night Agent) - Maintaining system state and transaction history - Providing a unified API for external systems - Monitoring system health and performance - Implementing business rules and decision logic - Ensuring data consistency across the ecosystem - Managing security and compliance requirements

The Control Agent is built on Node.js for compatibility with n8n workflow automation and to leverage its efficient asynchronous processing capabilities, making it ideal for handling multiple concurrent operations in a busy pizza business environment.

2. Component Architecture

The Control Agent system consists of the following major components:

```
graph TD
    CA[Control Agent] --> ORC[Orchestration Engine]
    CA --> API[API Gateway]
    CA --> SM[State Manager]
    CA --> DM[Decision Manager]
    CA --> IM[Integration Manager]
    CA --> SEC[Security Manager]
    CA --> MON[Monitoring & Logging]

    API --> EXT[External Systems]
    IM --> INT[Integration Systems]

    subgraph "Specialized Agents"
        MA[Manus - Inventory]
        RO[Roo - Customer Service]
        WS[Windsurf - Order Processing]
        NA[Night Agent - Marketing]
    end

    ORC --> MA
    ORC --> RO
    ORC --> WS
    ORC --> NA
```

2.1 Orchestration Engine

- **Purpose**: Coordinates workflows between specialized agents
- **Functions**:
  - Task scheduling and distribution
  - Workflow state management
  - Inter-agent communication
  - Error handling and recovery
  - Task prioritization

2.2 API Gateway

- **Purpose**: Provides unified interface for external systems
- **Functions**:
  - Request routing
  - Authentication and authorization
  - Rate limiting
  - Request/response transformation
  - API documentation (Swagger/OpenAPI)

2.3 State Manager

- **Purpose**: Maintains system state and transaction history
- **Functions**:
  - MongoDB data access layer
  - State persistence
  - Transaction logging
  - Data consistency enforcement
  - Cache management

2.4 Decision Manager

- **Purpose**: Implements business rules and decision logic
- **Functions**:
  - Rule engine
  - Business policy enforcement
  - Dynamic decision making
  - Anomaly detection
  - Optimization algorithms

2.5 Integration Manager

- **Purpose**: Connects with external systems
- **Functions**:
  - n8n workflow integration
  - POS system integration
  - Inventory management integration

  - CRM integration
  - Email/SMS service integration
  - Delivery logistics integration

2.6 Security Manager

- **Purpose**: Ensures system security and compliance
- **Functions**:
  - Authentication management
  - Encryption/decryption
  - Access control
  - Audit logging
  - Compliance enforcement (GDPR, PCI)

2.7 Monitoring & Logging

- **Purpose**: Tracks system health and performance
- **Functions**:
  - Performance monitoring
  - Error logging
  - Alerting
  - Analytics
  - Reporting

3. Communication Flows

3.1 Inter-Agent Communication Flow

```
sequenceDiagram
    participant CA as Control Agent
    participant MA as Manus (Inventory)
    participant RO as Roo (Customer Service)
    participant WS as Windsurf (Order Processing)
    participant NA as Night Agent (Marketing)

    RO->>CA: Customer places order
    CA->>MA: Check ingredient availability
    MA->>CA: Inventory status

    alt Ingredients available
        CA->>WS: Process order
        WS->>CA: Order status updates
        CA->>RO: Order confirmation

        WS->>CA: Order completed
        CA->>NA: Trigger post-order marketing
        NA->>CA: Marketing action completed
```

```
    else Ingredients unavailable
        CA->>RO: Notify unavailability
        CA->>MA: Update inventory needs
    end
```

3.2 External System Integration Flow

```
graph LR
    CA[Control Agent]

    subgraph "Integration Systems"
        n8n[n8n Workflow]
        POS[Point of Sale]
        INV[Inventory System]
        CRM[Customer Database]
        MSG[Email/SMS Services]
        DEL[Delivery Logistics]
    end

    CA <-->|Workflow automation| n8n
    CA <-->|Sales transactions| POS
    CA <-->|Inventory updates| INV
    CA <-->|Customer data| CRM
    CA <-->|Notifications| MSG
    CA <-->|Delivery tracking| DEL
```

3.3 Event-Based Communication

The Control Agent implements an event-driven architecture for asynchronous communication:

```
graph TD
    subgraph "Event Bus"
        EB[Message Broker]
    end

    CA[Control Agent] -->|Publish| EB
    MA[Manus] -->|Publish/Subscribe| EB
    RO[Roo] -->|Publish/Subscribe| EB
    WS[Windsurf] -->|Publish/Subscribe| EB
    NA[Night Agent] -->|Publish/Subscribe| EB

    EXT[External Systems] -->|Publish/Subscribe| EB
```

Key event types: - Order events (created, updated, completed) - Inventory events (low stock, restock needed) - Customer events (new customer, feedback received) - Marketing events (campaign triggered, promotion sent) - System events (error,

warning, notification)

4. Database Schema

The Control Agent uses MongoDB for its database needs. Below is the schema design for the main collections:

4.1 Tasks Collection

```
{
  _id: ObjectId,
  taskId: String,           // Unique identifier
  type: String,             // Task type (order, inventory, marketing, etc.)
  status: String,           // pending, in-progress, completed, failed
  priority: Number,         // 1-5 (5 being highest)
  assignedAgent: String,    // ID of the specialized agent
  payload: Object,          // Task-specific data
  source: String,           // Origin of the task
  created: Date,
  updated: Date,
  completed: Date,
  retryCount: Number,
  nextRetry: Date,
  errorDetails: [
    {
      timestamp: Date,
      message: String,
      stackTrace: String
    }
  ],
  metadata: Object          // Additional task metadata
}
```

4.2 Workflows Collection

```
{
  _id: ObjectId,
  workflowId: String,       // Unique identifier
  name: String,             // Workflow name
  description: String,
  status: String,           // active, paused, completed, failed
  tasks: [
    {
      taskId: String,       // Reference to Tasks collection
      sequence: Number,     // Order in workflow
      dependencies: [String], // Task IDs that must complete first
```

```
      status: String       // Task status within this workflow
    }
  ],
  startTime: Date,
  endTime: Date,
  initiatedBy: String,     // User or system that started the workflow
  metadata: Object         // Additional workflow metadata
}
```

4.3 Agents Collection

```
{
  _id: ObjectId,
  agentId: String,         // Unique identifier
  name: String,            // Agent name (Manus, Roo, etc.)
  type: String,            // Agent type
  status: String,          // online, offline, busy, error
  capabilities: [String],  // List of agent capabilities
  apiEndpoint: String,     // Agent API endpoint
  healthCheckEndpoint: String,
  lastSeen: Date,
  performance: {
    responseTime: Number,  // Average response time in ms
    successRate: Number,   // Percentage of successful tasks
    errorRate: Number      // Percentage of failed tasks
  },
  metadata: Object         // Additional agent metadata
}
```

4.4 Events Collection

```
{
  _id: ObjectId,
  eventId: String,         // Unique identifier
  type: String,            // Event type
  source: String,          // Event source
  timestamp: Date,
  payload: Object,         // Event data
  processed: Boolean,      // Whether the event has been processed
  relatedTasks: [String],  // Related task IDs
  metadata: Object         // Additional event metadata
}
```

4.5 Audit Collection

```
{
  _id: ObjectId,
```

```
  timestamp: Date,
  actor: String,           // Agent or user that performed the action
  action: String,          // Action performed
  resource: String,        // Resource affected
  resourceId: String,      // ID of the affected resource
  details: Object,         // Action details
  ipAddress: String,       // IP address (if applicable)
  userAgent: String        // User agent (if applicable)
}
```

4.6 Configuration Collection

```
{
  _id: ObjectId,
  key: String,             // Configuration key
  value: Mixed,            // Configuration value
  description: String,
  lastUpdated: Date,
  updatedBy: String,       // User or system that updated the config
  environment: String,     // dev, test, prod
  isEncrypted: Boolean     // Whether the value is encrypted
}
```

5. API Endpoints

The Control Agent exposes the following RESTful API endpoints:

5.1 Task Management

Endpoint	Method	Description	Request Body	Response
`/api/tasks`	GET	Get all tasks with optional filtering	Query params: status, type, agent	Array of task objects
`/api/tasks`	POST	Create a new task	Task object	Created task object
`/api/tasks/:taskId`	GET	Get task by ID	-	Task object
`/api/tasks/:taskId`	PUT	Update task	Updated task fields	Updated task object
`/api/tasks/:taskId/status`	PATCH	Update task status	{ status: String }	Updated task object
`/api/tasks/:taskId/assign`	PATCH	Assign task to agent	{ agentId: String }	Updated task object
`/api/tasks/:taskId/retry`	POST	Retry a failed task	-	Task object

5.2 Workflow Management

Endpoint	Method	Description	Request Body	Response
`/api/workflows`	GET	Get all workflows	Query params: status	Array of workflow objects
`/api/workflows`	POST	Create a new workflow	Workflow object	Created workflow object
`/api/workflows/:workflowId`	GET	Get workflow by ID	-	Workflow object
`/api/workflows/:workflowId`	PUT	Update workflow	Updated workflow fields	Updated workflow object
`/api/workflows/:workflowId/status`	PATCH	Update workflow status	{ status: String }	Updated workflow object
`/api/workflows/:workflowId/tasks`	POST	Add task to workflow	Task object	Updated workflow object

5.3 Agent Management

Endpoint	Method	Description	Request Body	Response
`/api/agents`	GET	Get all agents	Query params: status, type	Array of agent objects
`/api/agents/:agentId`	GET	Get agent by ID	-	Agent object
`/api/agents/:agentId/status`	PATCH	Update agent status	{ status: String }	Updated agent object
`/api/agents/:agentId/health`	GET	Check agent health	-	Health status object
`/api/agents/:agentId/tasks`	GET	Get tasks assigned to agent	Query params: status	Array of task objects

5.4 Event Management

Endpoint	Method	Description	Request Body	Response
`/api/events`	GET	Get all events	Query params: type, processed	Array of event objects
`/api/events`	POST	Create a new event	Event object	Created event object
`/api/events/:eventId`	GET	Get event by ID	-	Event object
`/api/events/:eventId/process`	POST	Mark event as processed	-	Updated event object

5.5 System Management

Endpoint	Method	Description	Request Body	Response
`/api/system/health`	GET	Get system health status	-	Health status object
`/api/system/metrics`	GET	Get system metrics	Query params: timeframe	Metrics object
`/api/system/config`	GET	Get system configuration	-	Configuration object
`/api/system/config`	PUT	Update system configuration	Updated config fields	Updated configuration object

5.6 Webhook Endpoints

Endpoint	Method	Description	Request Body	Response
`/api/webhooks/n8n`	POST	Webhook for n8n integration	n8n payload	Acknowledgment
`/api/webhooks/pos`	POST	Webhook for POS integration	POS payload	Acknowledgment
`/api/webhooks/inventory`	POST	Webhook for inventory integration	Inventory payload	Acknowledgment
`/api/webhooks/delivery`	POST	Webhook for delivery integration	Delivery payload	Acknowledgment

6. Security Architecture

6.1 Authentication & Authorization

The Control Agent implements a multi-layered security approach:

1. **API Authentication**:
   - JWT (JSON Web Tokens) for authentication
   - Token-based authentication for agent-to-agent communication
   - API keys for external system integration
   - OAuth 2.0 for third-party service integration
2. **Authorization**:
   - Role-based access control (RBAC)
   - Permission-based access for API endpoints
   - IP whitelisting for sensitive operations
   - Rate limiting to prevent abuse
3. **Agent Authentication**:
   - Mutual TLS (mTLS) for agent-to-agent communication
   - Agent registration and verification process
   - Periodic re-authentication

6.2 Data Encryption

1. **Data at Rest**:
   - MongoDB encryption for sensitive data
   - Field-level encryption for PII and payment data
   - Encrypted configuration values
2. **Data in Transit**:
   - TLS 1.3 for all HTTP communications
   - Encrypted payloads for sensitive data
   - Secure WebSockets for real-time communications

6.3 Compliance Measures

1. **GDPR Compliance**:
   - Data minimization principles
   - Purpose limitation for data collection
   - Consent management
   - Data subject rights support (access, rectification, erasure)
   - Data breach notification procedures
   - Privacy by design implementation
2. **PCI Compliance**:
   - Tokenization of payment information
   - Restricted access to payment data
   - Secure payment processing workflows
   - Regular security assessments
   - Compliance with PCI DSS requirements

6.4 Audit & Logging

1. **Security Audit**:
   - Comprehensive audit logging of all security events
   - Immutable audit trail

- Regular security audit reviews

2. **Access Logging**:
   - Logging of all API access attempts
   - Failed authentication tracking
   - Suspicious activity detection

6.5 Security Monitoring

1. **Real-time Monitoring**:
   - Intrusion detection system
   - Anomaly detection for unusual patterns
   - Rate limiting and abuse prevention
2. **Vulnerability Management**:
   - Regular security scanning
   - Dependency vulnerability tracking
   - Security patch management

7. Error Handling & Retry Mechanisms

7.1 Error Classification

The Control Agent classifies errors into the following categories:

1. **Transient Errors**:
   - Network timeouts
   - Temporary service unavailability
   - Rate limiting errors
   - Database connection issues
2. **Permanent Errors**:
   - Authentication failures
   - Authorization issues
   - Invalid input data
   - Business rule violations
3. **System Errors**:
   - Internal server errors
   - Unhandled exceptions
   - Resource exhaustion
   - Critical service failures

7.2 Retry Strategy

For transient errors, the Control Agent implements an exponential backoff retry strategy:

```
// Pseudo-code for retry strategy
function calculateRetryDelay(attempt) {
  const baseDelay = 1000; // 1 second
  const maxDelay = 60000; // 1 minute
```

```
  const jitter = Math.random() * 1000; // Random jitter up to 1 second

  // Exponential backoff with jitter
  return Math.min(baseDelay * Math.pow(2, attempt) + jitter, maxDelay);
}
```

Retry configuration: - Maximum retry attempts: 5 (configurable) - Retry delay: Exponential backoff with jitter - Circuit breaker pattern to prevent cascading failures

7.3 Error Recovery

1. **Task Recovery**:
   - Failed tasks are marked for retry
   - Task state is preserved for recovery
   - Partial results are saved when possible
2. **Workflow Recovery**:
   - Workflows can resume from point of failure
   - Compensation actions for partial workflow completion
   - Rollback mechanisms for transactional workflows
3. **System Recovery**:
   - Graceful degradation during partial system failures
   - Self-healing mechanisms
   - Automatic failover for critical components

7.4 Error Reporting

1. **Centralized Error Logging**:
   - Structured error logs with context
   - Error correlation across components
   - Error categorization and prioritization
2. **Alerting**:
   - Real-time alerts for critical errors
   - Error threshold monitoring
   - Escalation procedures for persistent issues
3. **Error Analytics**:
   - Error trend analysis
   - Failure rate monitoring
   - Mean time to recovery (MTTR) tracking

8. Scalability Considerations

8.1 Horizontal Scaling

The Control Agent is designed for horizontal scalability:

1. **Stateless Architecture**:
   - API Gateway and service components are stateless

- Session data stored in distributed cache
- Load balancing across multiple instances

2. **Microservices Approach**:
   - Components can scale independently
   - Service discovery for dynamic scaling
   - Container orchestration (Kubernetes) ready
3. **Database Scaling**:
   - MongoDB sharding for horizontal scaling
   - Read replicas for query-heavy workloads
   - Database connection pooling

8.2 Vertical Scaling

Considerations for vertical scaling:

1. **Resource Optimization**:
   - Memory usage optimization
   - CPU utilization monitoring
   - I/O performance tuning
2. **Performance Tuning**:
   - Node.js worker threads for CPU-intensive tasks
   - Garbage collection optimization
   - Event loop monitoring

8.3 Caching Strategy

Multi-level caching strategy:

1. **In-Memory Cache**:
   - Redis for frequently accessed data
   - Cache invalidation patterns
   - Time-to-live (TTL) configuration
2. **Application-Level Cache**:
   - Response caching for API endpoints
   - Query result caching
   - Computed value caching
3. **Database Cache**:
   - MongoDB query cache
   - Index optimization
   - Read preference configuration

8.4 Load Management

Techniques for handling increased load:

1. **Rate Limiting**:
   - API rate limiting by client
   - Gradual degradation under heavy load

- Priority-based request processing

2. **Queue Management**:
   - Task prioritization
   - Work queue sharding
   - Background processing for non-critical tasks
3. **Asynchronous Processing**:
   - Event-driven architecture
   - Message queues for task distribution
   - Asynchronous API endpoints for long-running operations

8.5 Growth Planning

Strategies for handling business growth:

1. **Capacity Planning**:
   - Regular load testing
   - Performance benchmarking
   - Growth forecasting
2. **Geographic Distribution**:
   - Multi-region deployment capability
   - Content delivery network (CDN) integration
   - Data locality considerations
3. **Monitoring & Alerting**:
   - Proactive scaling triggers
   - Resource utilization monitoring
   - Performance anomaly detection