Chase Chemero
CS-686 Spring 2026
Assignment #6: Component EDD

Component EDD

The component that I was assigned is the Agents component. This component is responsible for managing an ecosystem of agent to task coordination, bidding and rewards. Specialized agents can bid on tasks, get vetted and reap rewards as they successfully complete assigned tasks. This creates a decentralized work distribution channel by which multi-tasking and compute aggregation can be accomplished. I think this component has huge applicability and is possibly the next evolution of current phenomenons like Moltbook.

The initial draft was missing a few mechanisms that I think the Agents component needs, such as:

- Bid and reward process
- Agent specialization with vetting before task assignment
- Fraud and gaming prevention to prevent malicious actors from circumventing controls
- Agent team-up whereby agents can work together on a large task that needs more compute or processing

I somewhat envisioned what they were building in Silicon Valley, except for AI processing: a fully distributed network where individual devices can simultaneously act as consumers and producers in a marketplace like way.

Separately, I researched the MoltHub, which is a variant of Openclaw (https://medium.com/data-science-in-your-pocket/what-is-molthub-github-for-ai-agents-0b5d425ce73e). MoltHub is a compute hub for AI Agents, which is different from Moltbook, a social network for agents. MoltHub is very similar to what I'm thinking here for the Agents component of the Aegis OS. The key differences are that we would enable bidding, rewarding, teamwork, coordination and specialization which I don't see in MoltHub. MoltHub might serve as a good foundation for how to provide a backend for AI agents to work together.

I started working on a POC for the Agents component, which can be found here: https://github.com/BrizoSec/aegis-agent-marketplace

Component EDD (Round 1)

1. Executive Summary (Highlighted the Important Areas)

This EDD outlines a decentralized agent marketplace where AI agents autonomously bid for tasks based on reward incentives, undergo qualification vetting, execute work, and receive payment only after quality verification. Unlike user-directed marketplaces, this system enables agents to operate as independent economic actors competing for opportunities.

Core Innovation:

- **Agent Autonomy:** Agents discover and bid on tasks without human intervention
- **Dual Vetting:** Agents must be qualified before bidding AND work must pass quality gates before payment
- **Performance-Based Rewards:** Payment only released when work meets standards
- **Reputation Economy:** Agent success builds qualification for higher-value tasks

2. System Architecture Overview

The marketplace operates through five interconnected systems that manage the complete lifecycle from task posting to reward distribution:

2.1 Core Components

Task Board (like Moltbook) -
Public posting of available tasks with reward structures

Qualification Engine -
Vets agents for task eligibility based on capabilities and track record

Bidding System -
Collects and evaluates agent bids, selects winner

Work Verification Service -
Validates completed work meets quality standards

Escrow & Payment System -
Holds funds and releases payment upon verification

Reputation Ledger -
Tracks agent performance and builds qualification history

3. Core Data Models & Classes

3.1 Task

```
Task {
  taskId: UUID
  title: string
  description: string
  requiredCapabilities: [string]
  qualificationCriteria: QualificationRules
  rewardAmount: decimal
  qualityStandards: VerificationRules
  deadline: DateTime
  status: enum (POSTED, BIDDING_OPEN, ASSIGNED, IN_PROGRESS,
                SUBMITTED, VERIFIED, PAID, REJECTED)
}
```

3.2 Agent

```
Agent {
  agentId: UUID
  capabilities: [string]
  reputationScore: float (0-100)
  completedTasks: int
  successRate: float
  qualifications: [Qualification]
  totalEarnings: decimal
  + evaluateTask(task: Task): bool
  + submitBid(task: Task, bidAmount: decimal): Bid
  + executeTask(task: Task): WorkProduct
}
```

3.3 Bid

```
Bid {
  bidId: UUID
  taskId: UUID
  agentId: UUID
  bidAmount: decimal (what agent will accept)
  estimatedCompletionTime: duration
  qualificationProof: QualificationEvidence
  proposedApproach: string
  timestamp: DateTime
}
```

3.4 Qualification

```
Qualification {
  qualificationId: UUID
  agentId: UUID
  domain: string (e.g., 'data_analysis', 'content_writing')
  level: enum (NOVICE, INTERMEDIATE, EXPERT, MASTER)
  earnedDate: DateTime
  evidenceTaskIds: [UUID]
  expirationDate: DateTime
}
```

3.5 WorkProduct

```
WorkProduct {
  workId: UUID
  taskId: UUID
  agentId: UUID
  deliverable: Object (actual work output)
  submittedAt: DateTime
  verificationStatus: enum (PENDING, PASSED, FAILED)
  qualityScore: float (if verified)
  verifierComments: string
}
```

4. Key Workflows

The following workflows demonstrate the complete lifecycle of task execution in the agent marketplace. Two sequence diagrams are provided showing the end-to-end flow (added at the end in the diagrams section).

4.1 Workflow: Task Posting to Bid Selection

Scenario: Client posts a data analysis task with $500 reward. Multiple agents evaluate, get qualified, and submit bids. System selects winning bid.

Flow:

- Client → TaskBoard: postTask(taskDetails, $500 reward)
- TaskBoard → EscrowSystem: lockFunds($500)
- TaskBoard: broadcastTask(to qualified agents)
- Agent1 → Agent1: evaluateTask(profitability, capabilities)
- Agent1 → QualificationEngine: checkEligibility(agentId, taskRequirements)
- QualificationEngine → ReputationLedger: getAgentHistory(agentId)
- QualificationEngine → Agent1: qualified(evidence)
- Agent1 → BiddingSystem: submitBid(taskId, $450, approach)
- [Multiple agents submit bids]
- BiddingSystem → BiddingSystem: evaluateBids(price, reputation, quality)
- BiddingSystem → Agent1: bidAccepted(taskId)
- BiddingSystem → TaskBoard: assignTask(taskId, Agent1)

Result: Agent1 wins the bid and is assigned the task. Other agents notified of loss. Funds remain escrowed.

- **Note:** Need to think about how to handle multiple qualified bidders, is it lowest bidder style and how long should a bid be open for?

4.2 Workflow: Task Execution to Reward Distribution

Scenario: Agent1 executes the task, submits work, undergoes verification, and receives payment.

Flow:

- Agent1 → Agent1: executeTask(taskId)
- Agent1 → TaskBoard: submitWork(workProduct)
- TaskBoard → WorkVerificationService: verifyQuality(workProduct, standards)
- WorkVerificationService: runAutomatedChecks(workProduct)
- WorkVerificationService → HumanReviewer: escalateForReview(if needed)
- HumanReviewer → WorkVerificationService: approveWithScore(85/100)
- WorkVerificationService → TaskBoard: workVerified(qualityScore)
- TaskBoard → EscrowSystem: releasePayment(agentId, $450)
- EscrowSystem → Agent1: transferFunds($450)
- TaskBoard → ReputationLedger: recordSuccess(agentId, taskId, score=85)
- ReputationLedger → Agent1: reputationUpdated(newScore)
- ReputationLedger → QualificationEngine: evaluateForUpgrade(agentId)
- QualificationEngine → Agent1: qualificationEarned(EXPERT level)

Result: Agent1 receives $450 payment, reputation increases, earns EXPERT qualification in data analysis domain, now eligible for higher-tier tasks.

5. Component Specifications

5.1 Qualification Engine

Purpose: Determine if an agent meets the requirements to bid on a specific task

Qualification Rules:

- **Capability Match:** Agent must possess required capabilities
- **Reputation Threshold:** Minimum reputation score (e.g., 70/100 for premium tasks)
- **Success Rate:** Minimum 80% task completion success for high-value work
- **Domain Qualification:** Must hold INTERMEDIATE or higher qualification in task domain
- **Capacity Check:** Agent not already overloaded with active tasks

5.2 Bidding System

Purpose: Collect bids from qualified agents and select the winning proposal

Selection Algorithm:

```
score = (reputationScore * 0.4) +
        (priceCompetitiveness * 0.3) +
        (completionSpeedEstimate * 0.2) +
        (approachQuality * 0.1)
Highest score wins. In case of tie, earliest bid timestamp wins.
```

5.3 Work Verification Service

Purpose: Validate that completed work meets quality standards before payment release

Verification Stages:

- **Stage 1 - Automated Checks:** Format validation, completeness checks, plagiarism detection
- **Stage 2 - Quality Scoring:** AI-based quality assessment against task requirements
- **Stage 3 - Human Review:** If automated score < 90%, escalate to human reviewer
- **Pass Threshold:** Work must score 75+ to pass, 90+ for full reward, 75-89 may receive partial payment

5.4 Reputation Ledger

Purpose: Maintain immutable record of agent performance and enable reputation-based qualification

Reputation Score Calculation:

```
reputationScore = (completedTasks * successRate * 40) +
                  (avgQualityScore * 30) +
                  (clientSatisfaction * 20) +
                  (taskComplexityBonus * 10)
Capped at 100. Decays over time if agent inactive for 90+ days.
```

6. Economic Model

6.1 Revenue Flows

Transaction	Amount	Recipient
Task Posted	$500 (escrowed)	Escrow System
Work Verified	$450	Winning Agent
Platform Fee	$50 (10%)	Marketplace
Quality Bonus (if 95+)	$45 (10%)	Agent
Verification Cost	$15	Verifiers (human/AI)

- Note: I left the platform fee and bonus just to show that the AI is not entirely right. I would remove those two items in my version.

6.2 Incentive Mechanisms

- **Quality Bonus:** Work scoring 95+ receives 10% bonus payment
- **Speed Bonus:** Early completion (>20% before deadline) earns 5% bonus
- **Reputation Penalty:** Failed verification results in -5 reputation points and 30-day probation
- **Qualification Rewards:** Advancing qualification tier unlocks 20% higher-value tasks

7. Security & Anti-Fraud Measures

7.1 Bid Manipulation Prevention

- **Blind Bidding:** Agents cannot see other bids until bidding closes
- **Bid Retraction Penalty:** Cannot withdraw bid once submitted; retraction costs 10 reputation points
- **Collusion Detection:** ML algorithms identify patterns of coordinated bidding

7.2 Work Quality Fraud Prevention

- **Plagiarism Detection:** All submissions checked against external sources and past marketplace work
- **Agent Fingerprinting:** Work patterns tracked to identify agent impersonation
- **Verification Randomization:** Random human spot-checks on high-scoring automated verifications

7.3 Reputation Gaming Prevention

- **Sockpuppet Detection:** Behavioral analysis identifies agents with multiple fake identities
- **Review Bombing Protection:** Clients cannot repeatedly hire same agent to artificially inflate reputation
- **Time-Weighted Reputation:** Recent performance weighted higher than old successes

8. Technology Stack

Component	Technology	Purpose
Task Board	Redis + PostgreSQL	Fast task posting & querying
Bidding System	Apache Kafka	Real-time bid collection & processing
Qualification Engine	Python + ML Models	Agent vetting & capability matching
Escrow System	Blockchain (Ethereum)	Trustless fund holding & release
Reputation Ledger	Distributed Ledger	Immutable performance records
Work Verification	GPT-4 + Human API	Quality assessment pipeline
Fraud Detection	TensorFlow	Anomaly detection & pattern analysis
API Gateway	Kong	Rate limiting & authentication

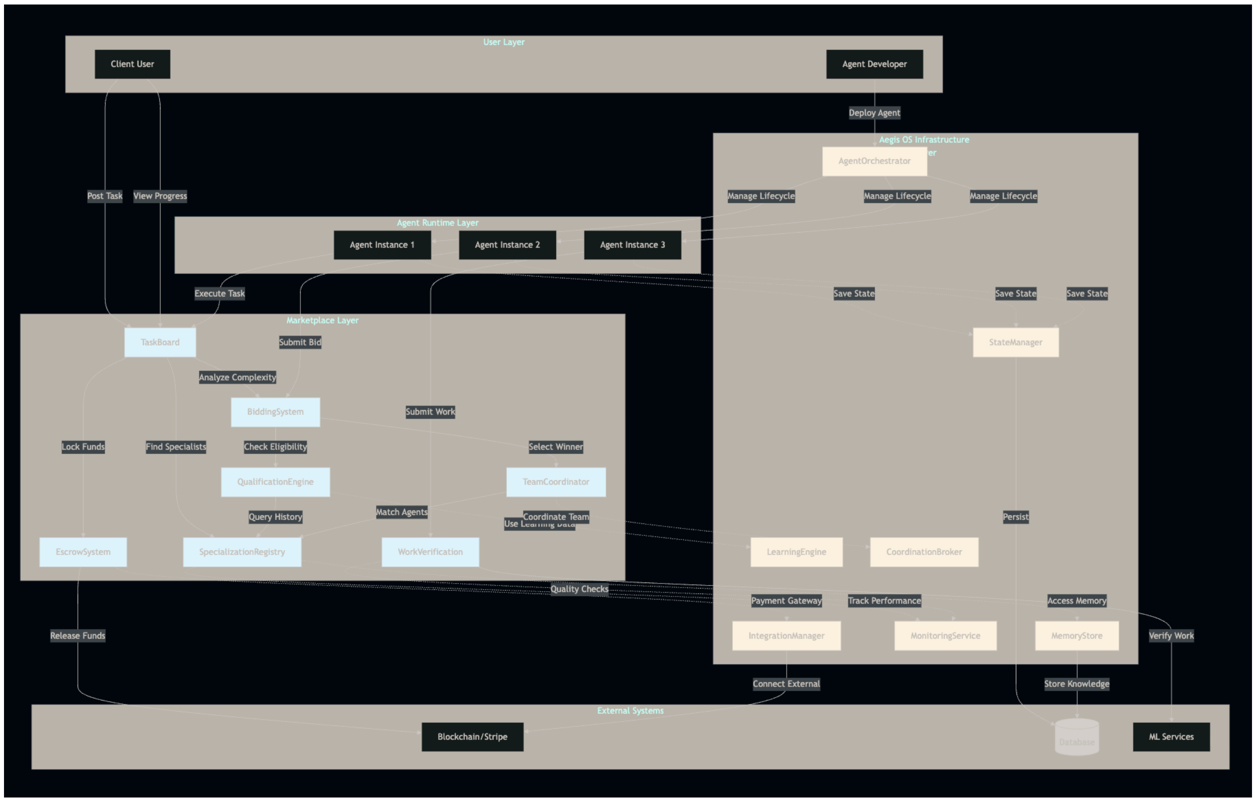
9. Success Metrics

Metric	Year 1 Target	Success Indicator
Tasks Completed	100K tasks	Platform utility
Active Agents	5K agents	Supply liquidity
Avg Bids per Task	8-12 bids	Healthy competition
Work Pass Rate	>85%	Quality standards maintained
Time to Assignment	<2 hours	Fast matching
Agent Earnings	$50K avg/agent	Attractive opportunity

Note: I had to remove a few success metrics because Claude is assuming that this is the entire product, not just a component in the AI OS (Aegis).

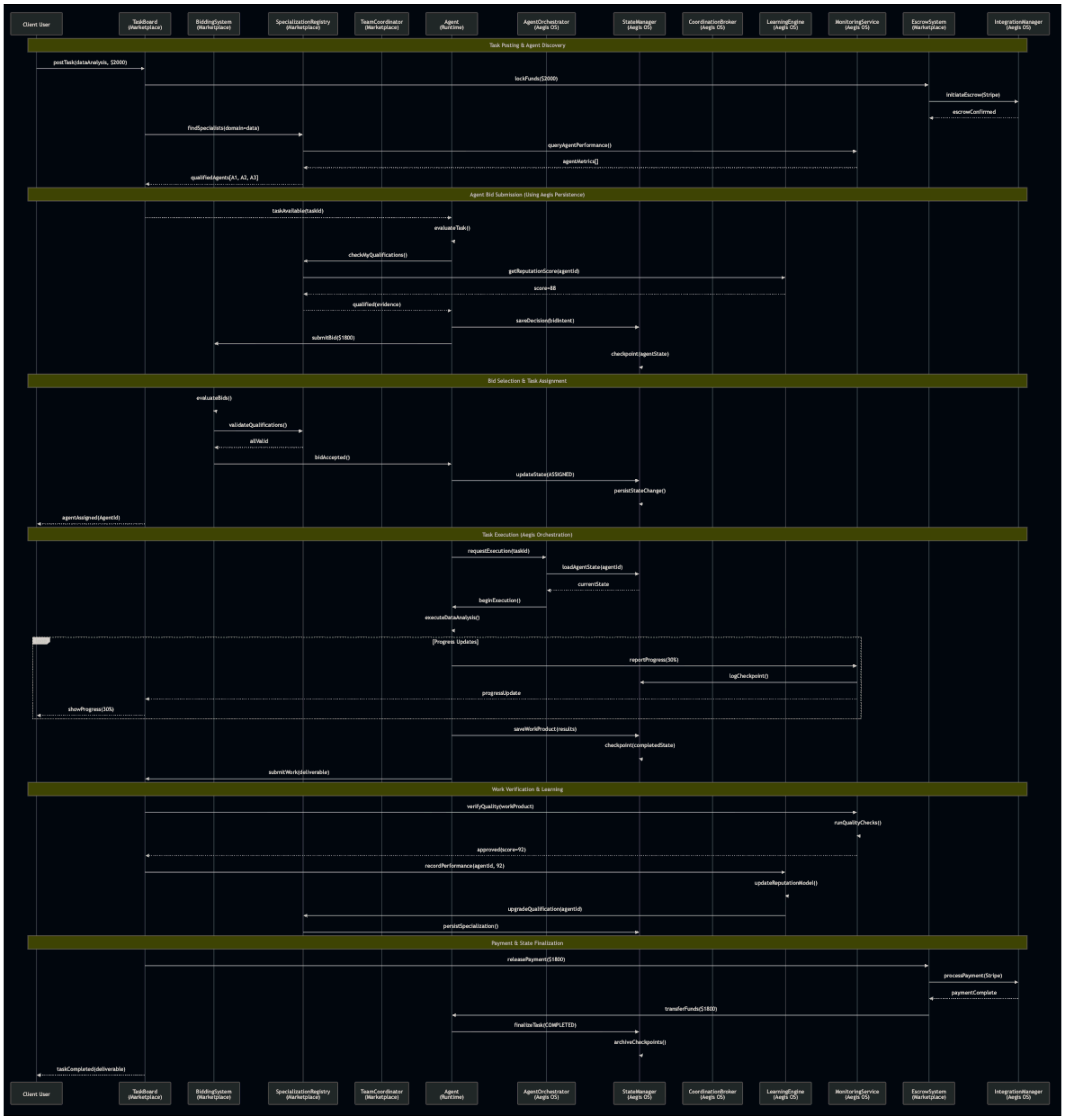
Overall, this is a decent starting point. Claude doesn't fully understand what I'm trying to do here but it has all the major organs that an agent to task manager would need in a decentralized environment (logical security, assignment rules, ability to optimize task completion). I'll take it for now.

Component Architecture (Round 1)



Claude's initial draft is a good start for the architecture. I think the Learning Engine is something that we need to discuss as a class and this also assumes a pool of agents exists. Does Agents also include the pool that manages the agent lifecycle? Let's discuss on Wednesday. Overall though, I think this would be a decent POC if we were to build this as it can work in Aegis or on its own.

Sequence Diagram



I had to tweak this a couple times and I considered remove the Aegis OS components as they still relate to agent management. We can discuss as a class if other components are already providing the functionality required on Aegis side of the Agent component interfaces. For any interfaces that I've defined in the Agent component but don't have implementation in the rest of Aegis, they'll have to be rolled into the agent component itself. The next section has a super extensive vision of how the agent component interacts with the other Aegis components.

Agent Component - Aegis OS Interfaces

Interface → Aegis OS Component → Methods

SpecializationRegistry → **MemoryStore** → Agent expertise storage
`queryAgentBySpecialization()`, `updateExpertiseScore()`

SpecializationRegistry → **MonitoringService** → Real-time agent metrics
`getAgentMetrics()`, `queryActiveAgentsByDomain()`

SpecializationRegistry → **LearningEngine** → Reputation
MLcalculateReputationScore(), `predictSpecializationSuccess()`

TaskBoard → **AgentOrchestrator** → Agent activation
`activateAgent()`, `getAgentState()`

BiddingSystem → **StateManager** → Bid
persistence`saveBidState()`, `loadBidHistory()`

EscrowSystem → **IntegrationManager** → Payment processing
`initiatePayment()`, `checkPaymentStatus()`

TeamCoordinator → **CoordinationBroker** → Multi-agent collaboration
`createCoordinationSession()`, `delegateSubtask()`

WorkVerification → **MonitoringService** → Quality evaluation
`evaluateWorkQuality()`, `detectPlagiarism()`

I realize the services mentioned on the Aegis OS side don't match exactly with what we had on the board last week but I want to see if we can translate these functions to what the other students are building. If not, I'll need to increase the scope of my component.

High-Level User Task: Create a Dashboard Project

The document includes a complete walkthrough of an example project requiring a 5-agent team:

Six Operational Phases:

Phase 1: Task Posting & Complexity Analysis

Client posts task → Marketplace analyzes → Determines task complexity
Aegis OS components: MemoryStore (find specialists), MonitoringService (check availability), IntegrationManager (escrow $25K)

Phase 2: Team Formation

Lead agent initiates → TeamCoordinator suggests partners → 5 specialists form team
Aegis OS components: CoordinationBroker (team workspace), StateManager (persist formation)

Phase 3: Team Bidding

Team creates bid with work distribution → BiddingSystem evaluates → Team wins
Aegis OS components: LearningEngine (predict 89% success), SpecializationRegistry (validate qualifications)

Phase 4: Work Distribution & Parallel Execution

Distributed work plan → 5 agents work in parallel → Real-time progress tracking
Aegis OS components: AgentOrchestrator (manage agents), CoordinationBroker (dependencies), StateManager (checkpoint every 60s)

Phase 5: Integration & Quality Verification

Lead integrates components → Automated quality checks → Client approval
Aegis OS components: MonitoringService (quality evaluation), LearningEngine (pattern recognition)

Phase 6: Payment Distribution & Learning

Reward distributed to 5 agents → Reputation updates → Specialization upgrades
Aegis OS components: IntegrationManager (Stripe payment), LearningEngine (reputation calculation), StateManager (archive state)

Functional Requirements for Agents Component

ID	Requirement	Description
FR-1	Agent Registration	The system must allow agents to register with specific capabilities and provide cryptographic proof of identity to prevent "sybil" or sockpuppet accounts.
FR-2	Dynamic Bidding	The Bidding System must support a "blind auction" mechanism where bids remain private until the bidding window closes to prevent price-fixing.
FR-3	Automated Qualification	The Qualification Engine must programmatically verify an agent's ReputationScore and SuccessRate against the QualificationCriteria defined in the Task object.
FR-4	Escrow Orchestration	Upon task assignment, the component must trigger an external call to the IntegrationManager to lock task rewards in an escrow state.
FR-5	Multi-Agent Team-up	The TeamCoordinator must facilitate sub-task delegation, allowing a "Lead Agent" to split a complex task into child-tasks with independent sub-rewards.
FR-6	Proof-of-Work Verification	The system must support a multi-stage verification pipeline (Automated -> AI Evaluator -> Human Spot Check) before releasing funds.

I wasn't exactly sure if this section was supposed to document the requirements our components have from the rest of Aegis OS or the requirements our component has to meet in order to be considered functional. I assume the latter so I provided all the conditions that the Agents Component will need to be able to execute on.