AI-Powered Task Scheduling Agent - AI Agents & Operations Documentation (Project Review Edition)

Project name: AI-Powered Task Scheduling Agent
System version: v2.1 (README "Latest Features"), API title indicates v2.0 (Week 4)
Backend: FastAPI + MongoDB + Firebase Auth + Socket.IO
AI runtime: Local Ollama models configured in settings (ollama_model, ollama_chat_model) via backend/app/config.py

---

1) Executive Summary (Business + Technical)

This system embeds multiple task-focused AI agents into student/teacher workflows to reduce manual effort and increase planning quality. The AI agents are not separate microservices; they are internal modules invoked from FastAPI routers and services, primarily through a shared Ollama integration layer (backend/app/services/ollama_service.py).

Business outcomes (as documented):
- Grading: ~70% faster
- Class monitoring: ~83% faster
- Task creation: ~90% faster

Technical approach:
- Most agents use prompted LLM inference (Ollama) with JSON-constrained outputs where structured data is required.
- Several agents include algorithmic fallbacks when AI fails (schedule generator fallbacks, JSON parsing fallbacks).

---

2) System AI Architecture (How Agents Fit Together)

2.1 High-level component flow

```
flowchart LR
  UI[React Frontend] --> API[FastAPI API]
  API --> DB[(MongoDB)]
  API --> WS[Socket.IO WebSocket]
  API --> AI[Ollama Service Layer]
  AI --> OLLAMA[(Local Ollama Runtime)]
  API --> GC[Google Calendar API]
  API --> OCR[Document Processor / OCR]

  subgraph Agents[AI Agents (Internal Modules)]
    A1[Task Analysis Agent]
    A2[Study Planner Agent]
    A3[Chat Assistant Agent]
    A4[Document Analysis Agent]
    A5[Resource Summarizer Agent]
    A6[Flashcard Agent]
    A7[Stress Recommendations Agent]
    A8[Extension Review Agent]
    A9[Grading Explanation Agent]
  end
  API --> Agents
  Agents --> AI
  OCR --> Agents
  GC --> API
  WS --> UI
```

2.2 Shared AI integration layer

The backend centralizes model calls in backend/app/services/ollama_service.py:
- generate_ai_response(prompt, json_mode=False) uses ollama.generate for single-turn generation.
- generate_json_response(prompt) uses Ollama's JSON mode (format='json') for better structured output.
- generate_chat_response(user_message, system_prompt, chat_history, document_content) uses ollama.chat for multi-turn, context-rich interactions.

---

3) AI Agent Catalog (All Implemented Agents)

Conventions used in this section

- Agent Name + Version: The repository does not define per-agent semantic versions. Versions below reflect module maturity and the project "Week X" milestone notes. Where explicit versioning is unavailable, it is noted as "internal module (no explicit version tag)".
- I/O Interfaces: Listed as API endpoints and/or Python functions actually called.
- Dependencies: Python packages + internal modules + DB collections and integration APIs.

3.1 Task Complexity & Subtask Generation Agent (Week 1 core)

Official agent name: Task Complexity & Subtask Agent
Version: internal module (no explicit tag); used by Tasks API

Primary function & responsibilities
- Estimate complexity (1-10), estimated hours, priority, and suggested deadline offset
- Generate 3-5 actionable subtasks to accelerate task breakdown

Technical specifications
- Architecture: Prompted LLM ? JSON parsing with fallback defaults
- Algorithms: LLM-based estimation (no deterministic scoring)
- Structured output enforcement: "Respond ONLY in JSON" prompt + best-effort JSON extraction

Input/Output interfaces
- Python functions:
- backend/app/services/ai_task_service.py::analyze_task_complexity(title, description)
- backend/app/services/ai_task_service.py::generate_subtasks(title, description)
- API integration:
- POST /api/tasks/ calls both functions (backend/app/routers/tasks.py)

Dependencies and integration points
- AI: backend/app/services/ollama_service.py::generate_ai_response
- MongoDB: tasks_collection (task insertion)

- WebSocket: broadcasts task updates via backend/app/websocket/broadcaster.py
- Optional Calendar: background sync via backend/app/services/google_calendar_service.py if enabled

Concrete example
- User action: Student creates a task via POST /api/tasks/
- Agent input: title + description
- Expected AI output shape:
  ```json
  {"complexity": 6, "hours": 5, "deadline_days": 7, "priority": "high"}
  ```
- System result: Task stored with complexity_score, estimated_hours, ai_suggested_deadline, and AI-generated subtasks.

3.2 Smart Study Planner Agent (Week 3)

Official agent name: Smart Study Planner Agent
Version: internal module (Week 3)

Primary function & responsibilities
- Generate a daily (or weekly) study schedule with:
- Deadline urgency ordering
- Complexity/stress balancing
- Break insertion (short/long)
- Session typing (pomodoro/deep_work/short_burst)

Technical specifications
- Hybrid architecture:
  1) Deterministic scoring + slot calculation
  2) LLM attempts a schedule first
  3) Deterministic greedy fallback if LLM fails
- Algorithms:
- Weighted scoring model: backend/app/services/ai_scheduling_service.py::score_tasks
- LLM schedule proposal (JSON): backend/app/services/ai_scheduling_service.py::ai_generate_schedule
- Greedy fallback: backend/app/services/ai_scheduling_service.py::generate_fallback_schedule
- Orchestrator: backend/app/services/ai_scheduling_service.py::generate_study_schedule

Input/Output interfaces
- API endpoints:
- POST /api/study-planner/generate (daily): backend/app/routers/study_planner.py
- POST /api/study-planner/generate-week (weekly): backend/app/routers/study_planner.py
- Inputs:
- target_date, regenerate
- user preferences (study hours, session length, breaks, complexity pattern)
- latest stress score (from stress logs)
- Output:
- JSON schedule doc with study_blocks, break_blocks, total_study_hours, ai_reasoning

Dependencies and integration points
- MongoDB:
- reads tasks_collection (active tasks)
- reads stress_logs_collection (latest stress)
- writes study_plans_collection
- AI: backend/app/services/ollama_service.py::generate_ai_response

- Preferences: user_preferences_collection

Concrete example
- User action: Student clicks "Generate Today's Plan"
- System call: POST /api/study-planner/generate
- Typical output shape:
  ```json
  {
    "study_blocks":[{"task_title":"Math Assignment","start_time":"10:00","end_time":"10:25
","duration_minutes":25,"session_type":"pomodoro"}],
    "break_blocks":[{"start_time":"10:25","end_time":"10:30","duration_minutes":5,"break_t
ype":"short"}],
    "total_study_hours": 3.5,
    "ai_reasoning": "High urgency tasks first; shorter sessions due to stress."
  }
  ```

3.3 Context-Aware AI Chat Assistant Agent (Week 4+ Enhanced)

Official agent name: Context-Aware AI Assistant (Chat)
Version: internal module; "basic" and "enhanced" endpoints implemented

Primary function & responsibilities
- Provide conversational assistance about:
- tasks, deadlines, priorities
- study plans and wellbeing signals (enhanced)
- slash-command execution (enhanced)
- document Q&A (enhanced)

Technical specifications
- Basic mode: prompt string composition + ollama.generate (single-turn)
- Enhanced mode: system prompt + scoped context + recent chat history + optional document content ? ollama.chat
- Context aggregation: multi-collection context builder: backend/app/services/user_context_service.py

Input/Output interfaces
- API endpoints:
- POST /api/chat/ai (basic): backend/app/routers/chat.py
- POST /api/chat/ai/enhanced (enhanced, file + scope + commands): backend/app/routers/chat.py
- POST /api/chat/ai/command (execute slash command): backend/app/routers/chat.py
- GET /api/chat/ai/context (context preview): backend/app/routers/chat.py

Dependencies and integration points
- AI:
- basic: backend/app/services/ollama_service.py::generate_ai_response
- enhanced: backend/app/services/ollama_service.py::generate_chat_response
- Context:
- backend/app/services/user_context_service.py::get_full_user_context
- backend/app/services/user_context_service.py::format_context_for_ai
- Slash commands:
- backend/app/services/command_parser.py
- Storage:
- AI conversations stored in chat_history_collection

Concrete examples
1) Task question (enhanced)
- Input: "What are my overdue tasks?"
- Behavior: the system injects overdue/pending task context into the system prompt and answers with task names and deadlines.
2) Command execution
- Input: /task list
- Behavior: command is executed server-side; result is stored as an assistant message and returned.

3.4 Document Analysis Agent (Upload ? Extract ? Answer)

Official agent name: Document Analysis Assistant
Version: internal module

Primary function & responsibilities
- Extract text from user-uploaded documents (PDF/DOCX/text/code/images via OCR)
- Provide AI analysis or Q&A over extracted content

Technical specifications
- Extraction pipeline: backend/app/services/document_processor.py
- PDF: pypdf
- DOCX: python-docx
- Images: Pillow + pytesseract OCR
- Analysis:
- backend/app/services/ollama_service.py::analyze_document_with_ai
- Enhanced chat integrates extraction directly in backend/app/routers/chat.py

Input/Output interfaces
- Input:
- file upload to POST /api/chat/ai/enhanced (multipart form file)
- Output:
- AI response plus document_metadata (type, filename, character count)

Dependencies and integration points
- Settings:
- ai_max_document_size, ai_max_context_length from backend/app/config.py
- OCR:
- Windows Tesseract path detection in backend/app/services/document_processor.py

Concrete example
- User uploads a PDF lecture note and asks: "Summarize key points and give practice questions."
- System extracts PDF text, truncates if needed, injects into AI chat context, returns a structured explanation.

3.5 Resource Summarization & Auto-Tagging Agent (Resource Library)

Official agent name: Resource Summarizer Agent
Version: internal module (Week 1)

Primary function & responsibilities
- Generate concise summaries of notes/documents and extract key points (used as suggested tags/key

concepts).

Technical specifications
- LLM prompt with bounded preview (content[:3000])
- Best-effort JSON extraction + fallback summary

Input/Output interfaces
- Function: backend/app/routers/resources.py::generate_resource_summary(content, title)
- Usage:
- file uploads and note creation in backend/app/routers/resources.py

Dependencies and integration points
- AI: backend/app/services/ollama_service.py::generate_ai_response
- MongoDB: resources_collection

Concrete example
- Teacher uploads "Syllabus.pdf" ? system stores file + ai_summary + ai_key_points ? UI shows summary and suggested tags.

3.6 Flashcard Generation Agent (Resources + Chat Command)

Official agent name: Flashcard Generator Agent
Version: internal module (Week 1)

Primary function & responsibilities
- Generate 8-12 study flashcards from a note/document summary/content.

Technical specifications
- Uses JSON-constrained generation mode: backend/app/services/ollama_service.py::generate_json_response
- Validates card schema and filters invalid entries.

Input/Output interfaces
- Endpoint: POST /api/resources/{resource_id}/flashcards (backend/app/routers/resources.py)
- Core generator: backend/app/routers/resources.py::generate_flashcards_ai
- Chat command: /flashcard generate <topic> (implemented in backend/app/services/command_parser.py)

Dependencies and integration points
- AI: JSON output mode for higher success rate
- MongoDB: resources_collection
- Extraction fallback:
- flashcard endpoint tries to read existing PDFs/text files if stored content is missing

Concrete example
- Student clicks "Generate Flashcards" on a note; system generates and persists flashcards and renders them in the flashcard viewer.

3.7 Stress Recommendations Agent (Wellbeing)

Official agent name: Stress Recommendation Agent
Version: internal module (Week 1)

Primary function & responsibilities
- Provide actionable recommendations to reduce stress based on workload and deadlines.

Technical specifications
- Deterministic stress scoring (0-10) based on:
- upcoming deadline pressure
- average complexity
- deadline overlap
- historical pattern from recent logs
- LLM recommendations: attempts JSON array parsing, otherwise uses fallback recommendations

Input/Output interfaces
- Endpoint: GET /api/stress/current (backend/app/routers/stress.py)
- Persists results to stress_logs collection.

Dependencies and integration points
- MongoDB:
- reads tasks_collection
- reads/writes stress_logs
- AI: backend/app/services/ollama_service.py::generate_ai_response

Concrete example
- Student has multiple near-term deadlines ? objective stress increases ? agent recommends prioritization and timeboxing actions.

3.8 Extension Request Review Agent (Teacher workflow support)

Official agent name: Extension Review Agent
Version: internal module

Primary function & responsibilities
- Analyze a student's deadline extension request and recommend approve/deny/conditional with confidence and reasoning.

Technical specifications
- LLM prompt forces JSON response schema; parses JSON with fallback
- Confidence clamped to [0.0, 1.0]

Input/Output interfaces
- Function: backend/app/services/ai_extension_service.py::analyze_extension_request(...)
- Endpoint: POST /api/extensions/ (backend/app/routers/extensions.py) stores ai_recommendation, ai_reasoning, ai_confidence_score.

Dependencies and integration points
- MongoDB: extension_requests_collection, tasks_collection, notifications_collection
- WebSocket: notification broadcast to teacher via backend/app/websocket/broadcaster.py
- AI: backend/app/services/ollama_service.py::generate_ai_response

Concrete example
- Student requests +7 days due to overlapping deadlines ? agent suggests "conditional" with a suggested deadline ? teacher approves/denies via review endpoint.

3.9 AI Grading Explanation Agent (Implemented, limited integration)

Official agent name: AI Grading Assistant (Explanation)
Version: internal module (Week 2 feature)

Primary function & responsibilities
- Turn performance metrics into a teacher-friendly, constructive grading explanation:
- reasoning
- strengths, weaknesses
- improvement suggestions
- encouragement

Technical specifications
- LLM prompt asks for JSON response; parses JSON; fallback response if invalid
- Includes deterministic grade calculation helper: calculate_grade_from_performance

Input/Output interfaces
- Functions:
- backend/app/services/ai_grading_service.py::generate_grading_explanation(...)
- backend/app/services/ai_grading_service.py::calculate_grade_from_performance(...)
- Note: current teacher review endpoints in backend/app/routers/grading.py focus on task review and feedback submission; direct invocation of ai_grading_service is not shown in the current router.

Dependencies and integration points
- AI: backend/app/services/ollama_service.py::generate_ai_response

Concrete example
- Intended usage: compile submission metrics + call agent ? return structured feedback JSON suitable for a grading dashboard UI.

---

4) Implementation Details

4.1 Development timeline and version history

- Week 1: Student AI features (stress meter, focus mode, resource AI summary + flashcards)
- Week 2: Teacher efficiency tools (grading assistant module, class analytics, bulk tasks)
- Week 3: Smart study planner
- Week 4: Google Calendar integration
- v2.1 (Jan 2026): UI redesign + WebSocket features + USN support + resource library enhancements

4.2 Performance metrics and benchmarks

Business-level metrics available in-repo
- Teacher time savings and efficiency improvements are documented in README.md.

System-level benchmarks (current state)
- The backend does not include explicit latency/throughput benchmarking instrumentation.
- Performance drivers:
- LLM inference time (model size, hardware, prompt length)
- MongoDB query performance (indexes configured in backend/app/db_config.py)
- WebSocket fanout volume for broadcasts

Recommended benchmark plan for review
- p50/p95 latency per AI endpoint (/chat/ai/enhanced, /study-planner/generate, /resources/{id}/flashcards)
- Prompt size vs latency curve (document uploads + large contexts)
- Fallback rate (% JSON parsing failures leading to fallback behavior)
- Error rate categories (Ollama unreachable, OCR missing, malformed docs)

4.3 Error handling and recovery mechanisms

- Central AI error pattern: generate_ai_response returns strings beginning with "AI Error:" on failure.
- Structured output fallback: agents attempt to parse JSON by slicing {...} or [...]; on failure they return safe defaults.
- Task analysis defaults in backend/app/services/ai_task_service.py
- Scheduling fallback algorithm in backend/app/services/ai_scheduling_service.py
- Flashcards return structured errors and validate schema in backend/app/routers/resources.py
- Document extraction resilience: if optional libraries are missing, the extraction returns explicit error text; unsupported types are rejected by allow-list.

4.4 Security considerations

Authentication and authorization
- REST: Firebase token verification in routers (example: backend/app/routers/tasks.py)
- WebSocket: token required on connect and validated in backend/app/websocket/events.py

Secrets management
- Required env vars: MONGODB_URI, FIREBASE_CREDENTIALS_PATH, SECRET_KEY in backend/app/config.py
- Calendar token encryption key: optional but strongly recommended (CALENDAR_ENCRYPTION_KEY)

Sensitive token storage
- Google tokens are encrypted with Fernet before storage in MongoDB (backend/app/services/google_calendar_service.py).
- Operational caveat: if CALENDAR_ENCRYPTION_KEY is not set, an ephemeral runtime key is generated, which can make stored tokens undecryptable after restart. Workaround: set a stable CALENDAR_ENCRYPTION_KEY.

File upload safety
- AI chat document upload:
- extension allow-list via is_supported_file
- size limit via ai_max_document_size
- filename sanitization and safe storage under uploads/ai_docs/...
- Task attachment upload:
- allow-listed extensions + max size in backend/app/routers/tasks.py

---

5) Operational Documentation

5.1 Deployment configuration

Local/dev startup
- Windows starter script: start.bat
- Backend: uvicorn app.main:app --reload --host 0.0.0.0 --port 8000

- Frontend: npm run dev
- Combined REST + Socket.IO ASGI app is wired in backend/app/main.py.

Key runtime dependencies
- Ollama local runtime
- MongoDB instance
- Firebase service account JSON
- Optional: Google OAuth credentials for calendar sync

5.2 Monitoring and maintenance procedures

- Logging writes:
- logs/app.log (DEBUG+)
- logs/error.log (ERROR+)
- console INFO logs
- configured in backend/app/utils/logger.py
- Operational checks:
- GET /health endpoint in backend/app/main.py
- AI connectivity sanity check: backend/app/services/ollama_service.py::test_ollama_connection

5.3 Scaling capabilities

- API scaling: FastAPI can scale horizontally behind a reverse proxy (stateless requests).
- WebSocket scaling: Socket.IO generally requires sticky sessions or a shared broker to scale horizontally. Current configuration appears single-process (no broker configured).
- AI scaling: local Ollama runtime is typically the bottleneck; scale vertically (GPU/CPU/RAM) or isolate Ollama onto a dedicated host.

5.4 Known limitations and workarounds

- Calendar encryption key stability: set CALENDAR_ENCRYPTION_KEY to persist decryptability across restarts.
- Per-agent versioning: modules do not have explicit semantic versions; add explicit version tags if formal auditability is required.
- AI grading integration gap: grading explanation module exists but is not visibly invoked by grading endpoints; integrate into teacher review flow if required.
- Benchmark instrumentation absent: business metrics exist, but p95 latency/fallback rates/error budgets are not measured in code; add telemetry if production review requires SLAs.

Appendix A) Quick Reference: AI-Related Endpoints (Backend)

- Tasks (AI analysis + subtasks): POST /api/tasks/ (backend/app/routers/tasks.py)
- Study planner: POST /api/study-planner/generate (backend/app/routers/study_planner.py)
- Stress recommendations: GET /api/stress/current (backend/app/routers/stress.py)
- Resource summarization: note/file endpoints in backend/app/routers/resources.py
- Flashcards: POST /api/resources/{resource_id}/flashcards (backend/app/routers/resources.py)
- Extensions (AI recommendation): POST /api/extensions/ (backend/app/routers/extensions.py)
- AI chat:
- Basic: POST /api/chat/ai
- Enhanced: POST /api/chat/ai/enhanced (backend/app/routers/chat.py)