## 🧠 Personal Knowledge Management (PKM)

### Architecture

typescript

```typescript
interface PersonalNote {
  id: string;
  project_id?: string;
  title: string;
  content: string;
  tags: string[];
  type: 'insight' | 'decision' | 'learning' | 'todo' | 'reference';

  // Smart connections
  related_chats: string[];
  related_documents: string[];
  related_projects: string[];
  auto_generated_links: {
    target_id: string;
    target_type: 'note' | 'project' | 'document';
    similarity_score: number;
    reason: string;
  }[];

  // Embedding for search
  embedding?: number[];

  // Metadata
  created_at: string;
  updated_at: string;
  importance: 1 | 2 | 3;
  status: 'active' | 'archived';
}
```

### UI Integration

### In-Chat Quick Capture:

typescript

```
// During conversation, show floating button
<ChatMessage>
  <MessageContent>{message.content}</MessageContent>
  <QuickActions>
    <button onClick={() => captureInsight(message)}>
      💡 Save Insight
    </button>
    <button onClick={() => captureDecision(message)}>
      🎯 Log Decision
    </button>
  </QuickActions>
</ChatMessage>

// Capture modal
<InsightCaptureModal>
  <input
    defaultValue={extractTitle(message)}
    placeholder="Insight title"
  />
  <textarea
    defaultValue={message.content}
    placeholder="Details (auto-filled from message)"
  />
  <TagInput
    suggestions={suggestTags(message, project)}
    placeholder="Tags"
  />
  <ImportanceSelector />
  <button>Save to Knowledge Base</button>
</InsightCaptureModal>
```

**Knowledge Graph View:**

typescript

```
// New dashboard tab: "Knowledge"
<KnowledgeGraphView>
  <GraphVisualization>
    {/* D3.js force-directed graph */}
    {/* Nodes: notes, projects, documents */}
    {/* Edges: connections, similarity */}
  </GraphVisualization>

  <Sidebar>
    <SearchBox placeholder="Search your knowledge..." />
    <FilterBar>
      <Filter by="type" />
      <Filter by="project" />
      <Filter by="date" />
      <Filter by="importance" />
    </FilterBar>
    <NotesList />
  </Sidebar>
</KnowledgeGraphView>
```

### Automatic Link Suggestions

typescript

```typescript
// When creating a note, suggest related content
async function suggestRelatedContent(note: PersonalNote): Promise<Suggestion[]> {
  // 1. Generate embedding
  const embedding = await ollama.embed(note.content);

  // 2. Vector search in existing notes
  const similarNotes = await redis.ft.search(
    'idx:notes',
    '*=>[KNN 5 @embedding $vec]',
    { params: { vec: embedding } }
  );

  // 3. Find related projects (keyword matching)
  const relatedProjects = await findRelatedProjects(note.tags);

  // 4. Find related documents in libraries
  const relatedDocs = await findRelatedDocuments(embedding);

  return [...similarNotes, ...relatedProjects, ...relatedDocs];
}
```

---

## 📋 Decision Journal

### Schema

typescript

```typescript
interface Decision {
  id: string;
  project_id: string;
  question: string;
  chosen_option: string;
  alternatives_considered: Alternative[];
  reasoning: string[];
  trade_offs: TradeOff[];

  // Context
  context: {
    project_phase: string;
    constraints: string[];
    goals: string[];
  };

  // Review system
  outcome?: 'success' | 'neutral' | 'regret';
  outcome_notes?: string;
  retrospective?: string;
  would_change?: boolean;
  lessons_learned?: string[];

  // Timeline
  decided_at: string;
  review_scheduled_for?: string;
  reviewed_at?: string;
}

interface Alternative {
  option: string;
  pros: string[];
  cons: string[];
  estimated_effort: 'low' | 'medium' | 'high';
  estimated_risk: 'low' | 'medium' | 'high';
  why_not_chosen: string;
}

interface TradeOff {
  aspect: string;
  chosen_value: string;
```

```
  sacrificed_value: string;
}
```

### Decision Capture Workflow

typescript

```typescript
// services/orchestrator/src/decision-capture.ts
export class DecisionCaptureService {
  async captureDecision(context: DecisionContext): Promise<Decision> {
    // 1. Detect decision point in conversation
    const isDecision = await this.detectDecisionPoint(context.message);

    if (isDecision) {
      // 2. Extract decision details using LLM
      const extracted = await this.extractDecisionDetails(context);

      // 3. Ask Skeptic agent for alternatives analysis
      const alternatives = await this.analyzeAlternatives(extracted);

      // 4. Store decision
      const decision = await this.storeDecision({
        ...extracted,
        alternatives,
        project_id: context.project_id
      });

      // 5. Schedule review
      await this.scheduleReview(decision, '+3 months');

      return decision;
    }
  }

  private async detectDecisionPoint(message: string): Promise<boolean> {
    const decisionKeywords = [
      'decided to', 'chose', 'going with', 'selected',
      'will use', 'instead of', 'rather than', 'vs'
    ];

    return decisionKeywords.some(kw =>
      message.toLowerCase().includes(kw)
    );
  }
}
```

### Periodic Review System

typescript

```
// Scheduled task runs weekly
async function reviewDecisions() {
  const dueForReview = await redis.ft.search(
    'idx:decisions',
    '@review_scheduled_for:[0 ' + Date.now() + ']'
  );

  for (const decision of dueForReview) {
    // Create notification
    await notifications.create({
      type: 'decision_review',
      title: 'Time to review a decision',
      message: `3 months ago you decided: ${decision.question}`,
      actions: [
        { label: 'Review Now', url: `/decisions/${decision.id}/review` },
        { label: 'Remind Me Later', action: 'snooze' }
      ]
    });
  }
}
```

### Decision Dashboard

typescript

```
// New tab: "Decisions"
<DecisionDashboard>
  <MetricsBar>
    <Stat label="Decisions Made" value={totalDecisions} />
    <Stat label="Success Rate" value="87%" />
    <Stat label="Pending Review" value={pendingReviews} />
  </MetricsBar>

  <DecisionTimeline>
    {decisions.map(d => (
      <DecisionCard
        decision={d}
        onReview={() => openReviewModal(d)}
        showOutcome={d.reviewed_at !== null}
      />
    ))}
  </DecisionTimeline>

  <InsightsPanel>
    <Insight>Most successful decisions were made during planning phase</Insight>
    <Insight>Decisions made quickly had 12% lower success rate</Insight>
  </InsightsPanel>
</DecisionDashboard>
```

---

## 📚 Template Library

### Schema

typescript

```typescript
interface Template {
  id: string;
  name: string;
  description: string;
  category: 'project' | 'n8n_workflow' | 'analysis' | 'research';

  // Version control
  version: string;
  changelog: VersionChange[];

  // Template structure
  structure: TemplateStructure;
  variables: Variable[];

  // n8n specific
  n8n_workflow?: N8nWorkflowTemplate;

  // Usage tracking
  times_used: number;
  success_rate: number;
  avg_completion_time: number;
  last_used: string;

  // Metadata
  created_by: string;
  created_at: string;
  updated_at: string;
  is_public: boolean;
  tags: string[];
}

interface TemplateStructure {
  agents: AgentType[];
  execution_order: ExecutionStep[];
  default_models: Record<AgentType, string>;
  approval_points: string[];
  expected_artifacts: string[];
}

interface Variable {
  name: string;
  type: 'text' | 'choice' | 'number' | 'library_reference';
  description: string;
```

```
  default?: any;
  required: boolean;
  validation?: string; // Regex or validation rule
}

interface N8nWorkflowTemplate {
  workflow_json: any;
  webhook_params: string[];
  required_credentials: string[];
  expected_inputs: Variable[];
  expected_outputs: string[];
}

interface VersionChange {
  version: string;
  changes: string[];
  breaking_changes?: string[];
  date: string;
}
```

### Built-in Templates

typescript

```
// Pre-loaded templates
const BUILTIN_TEMPLATES = [
  {
    id: 'research-deep-dive',
    name: 'Deep Research Report',
    description: 'Comprehensive research with multiple sources and verification',
    category: 'research',
    structure: {
      agents: ['researcher', 'critic', 'rag'],
      execution_order: [
        { agent: 'researcher', step: 'Initial web research' },
        { agent: 'rag', step: 'Query document libraries' },
        { agent: 'researcher', step: 'Synthesize findings' },
        { agent: 'critic', step: 'Verify claims and sources' }
      ]
    },
    variables: [
      { name: 'research_topic', type: 'text', required: true },
      { name: 'depth', type: 'choice', choices: ['shallow', 'medium', 'deep'], default: 'medium' },
      { name: 'relevant_libraries', type: 'library_reference', required: false }
    ]
  },

  {
    id: 'code-feature-full',
    name: 'Full Feature Implementation',
    description: 'Design → Code → Test → Review → Deploy',
    category: 'project',
    structure: {
      agents: ['design', 'coder', 'critic', 'executor'],
      execution_order: [
        { agent: 'design', step: 'Create architecture diagram' },
        { agent: 'coder', step: 'Generate implementation' },
        { agent: 'critic', step: 'Security and quality review' },
        { agent: 'executor', step: 'Run tests' },
        { agent: 'coder', step: 'Create PR' }
      ],
      approval_points: ['after_design', 'before_deploy']
    },
    variables: [
      { name: 'feature_description', type: 'text', required: true },
      { name: 'repo_path', type: 'text', required: true },
      { name: 'target_branch', type: 'text', default: 'main' }
```

```
  ]
},

{
  id: 'business-validation',
  name: 'Business Idea Validation',
  description: 'Research → Analyze → Challenge assumptions',
  category: 'analysis',
  structure: {
    agents: ['researcher', 'analyst', 'skeptic'],
    execution_order: [
      { agent: 'researcher', step: 'Market research' },
      { agent: 'analyst', step: 'Competitive analysis' },
      { agent: 'skeptic', step: 'Challenge assumptions and find flaws' }
    ]
  },
  variables: [
    { name: 'business_idea', type: 'text', required: true },
    { name: 'target_market', type: 'text', required: true },
    { name: 'investment_amount', type: 'number', required: false }
  ]
},

{
  id: 'n8n-data-pipeline',
  name: 'Data Processing Pipeline',
  description: 'n8n workflow for ETL tasks',
  category: 'n8n_workflow',
  n8n_workflow: {
    workflow_json: {
      nodes: [
        { name: 'Trigger', type: 'webhook' },
        { name: 'Extract Data', type: 'code' },
        { name: 'Transform', type: 'agent', agent: 'analyst' },
        { name: 'Load to Redis', type: 'redis' }
      ]
    },
    webhook_params: ['data_source', 'output_format'],
    required_credentials: ['redis'],
    expected_inputs: [
      { name: 'data_source', type: 'text', description: 'URL or file path' }
    ]
  }
```

```
  }
];
```

### Template Marketplace UI

typescript

```
// New tab: "Templates"
<TemplateMarketplace>
  <SearchBar placeholder="Search templates..." />

  <CategoryTabs>
    <Tab>All</Tab>
    <Tab>Projects</Tab>
    <Tab>n8n Workflows</Tab>
    <Tab>Research</Tab>
    <Tab>Analysis</Tab>
  </CategoryTabs>

  <TemplateGrid>
    {templates.map(t => (
      <TemplateCard
        template={t}
        onUse={() => openTemplateWizard(t)}
        onPreview={() => showTemplatePreview(t)}
        onFork={() => forkTemplate(t)}
      >
        <TemplateHeader>
          <h3>{t.name}</h3>
          <VersionBadge>{t.version}</VersionBadge>
        </TemplateHeader>

        <TemplateStats>
          <Stat icon="🎯" label="Success Rate" value={t.success_rate + '%'} />
          <Stat icon="⚡" label="Avg Time" value={formatDuration(t.avg_completion_time)} />
          <Stat icon="📊" label="Used" value={t.times_used + ' times'} />
        </TemplateStats>

        <TemplateDescription>{t.description}</TemplateDescription>

        <AgentChain>
          {t.structure.agents.map(agent => (
            <AgentBadge>{agent}</AgentBadge>
          ))}
        </AgentChain>

        <Actions>
          <button>Use Template</button>
          <button>Preview</button>
          <IconButton icon="fork">Fork</IconButton>
```

```
      </Actions>
    </TemplateCard>
  ))}
 </TemplateGrid>
</TemplateMarketplace>
```

### One-Click Project Creation

typescript

```typescript
// services/orchestrator/src/template-executor.ts
export class TemplateExecutor {
  async createProjectFromTemplate(
    templateId: string,
    variables: Record<string, any>,
    userId: string
  ): Promise<Project> {
    const template = await this.loadTemplate(templateId);

    // 1. Validate variables
    this.validateVariables(template.variables, variables);

    // 2. Create project
    const project = await db.projects.create({
      name: this.interpolate(template.name, variables),
      type: template.category,
      status: 'active',
      template_id: templateId,
      template_version: template.version,
      user_id: userId
    });

    // 3. Generate execution plan
    const plan = this.generatePlanFromTemplate(template, variables);

    // 4. Create initial task
    const task = await db.tasks.create({
      project_id: project.id,
      query: this.interpolate(template.description, variables),
      execution_plan: plan,
      status: 'planning'
    });

    // 5. If n8n workflow, deploy it
    if (template.n8n_workflow) {
      const workflow = await this.deployN8nWorkflow(
        template.n8n_workflow,
        variables,
        project.id
      );

      await db.projects.update(project.id, {
        n8n_workflow_id: workflow.id
```

```
    });
  }

  // 6. Track template usage
  await this.trackTemplateUsage(templateId, userId);

  return project;
}

private generatePlanFromTemplate(
  template: Template,
  variables: Record<string, any>
): ExecutionPlan {
  return {
    steps: template.structure.execution_order.map((step, idx) => ({
      id: `step-${idx + 1}`,
      description: this.interpolate(step.step, variables),
      agent_type: step.agent,
      model: template.structure.default_models[step.agent],
      depends_on: idx > 0 ? [`step-${idx}`] : [],
      require_approval: template.structure.approval_points?.includes(step.step)
    }))
  };
}
}
```

### Template Versioning

typescript

```typescript
// Template version control
interface TemplateVersion {
  template_id: string;
  version: string;
  changes: string[];
  breaking: boolean;
  created_at: string;
  created_by: string;
}

// When template is updated
async function updateTemplate(templateId: string, updates: Partial<Template>) {
  const current = await db.templates.findById(templateId);

  // Determine if breaking change
  const isBreaking = this.isBreakingChange(current, updates);

  // Increment version
  const newVersion = isBreaking
    ? incrementMajor(current.version)  // 1.0.0 → 2.0.0
    : incrementMinor(current.version); // 1.0.0 → 1.1.0

  // Store version history
  await db.template_versions.create({
    template_id: templateId,
    version: current.version,
    snapshot: current,
    created_at: new Date().toISOString()
  });

  // Update template
  await db.templates.update(templateId, {
    ...updates,
    version: newVersion,
    updated_at: new Date().toISOString()
  });

  // Notify users who have used this template
  if (isBreaking) {
    await this.notifyTemplateUsers(templateId, newVersion);
  }
}
```

## ⏰ Scheduled & Recurring Tasks

### Schema

typescript

```typescript
interface ScheduledTask {
  id: string;
  name: string;
  description: string;

  // Schedule
  schedule: string; // Cron format: '0 8 * * 1' (Mon 8am)
  timezone: string;
  enabled: boolean;

  // Task definition
  template_id?: string;
  query?: string;
  agents: AgentType[];
  variables?: Record<string, any>;

  // Delivery & notifications
  output_destinations: OutputDestination[];
  notify_on: 'always' | 'completion' | 'changes_only' | 'failure';
  notification_channels: ('discord' | 'email' | 'dashboard')[];

  // Execution history
  last_run?: string;
  last_status?: 'success' | 'failure';
  next_run: string;
  run_count: number;

  // Results storage
  store_results_in_library?: string;
  retention_policy: 'keep_all' | 'keep_last_n' | 'keep_for_days';
  retention_value?: number;

  created_at: string;
  created_by: string;
}

interface OutputDestination {
  type: 'discord' | 'email' | 'library' | 'dashboard' | 'notion' | 'webhook';
  config: any;
}
```

### Scheduler Service

typescript

```
// services/scheduler/src/cron-manager.ts
import { CronJob } from 'cron';

export class SchedulerService {
  private jobs: Map<string, CronJob> = new Map();

  async start() {
    // Load all enabled scheduled tasks
    const tasks = await db.scheduled_tasks.findAll({ enabled: true });

    for (const task of tasks) {
      this.registerJob(task);
    }
  }

  private registerJob(task: ScheduledTask) {
    const job = new CronJob(
      task.schedule,
      async () => await this.executeScheduledTask(task),
      null,
      true,
      task.timezone
    );

    this.jobs.set(task.id, job);
    logger.info(`Registered scheduled task: ${task.name}`);
  }

  private async executeScheduledTask(task: ScheduledTask) {
    try {
      logger.info(`Executing scheduled task: ${task.name}`);

      // Update last_run
      await db.scheduled_tasks.update(task.id, {
        last_run: new Date().toISOString(),
        run_count: task.run_count + 1
      });

      // Execute task
      const result = await this.runTask(task);

      // Store result
      if (task.store_results_in_library) {
```

```
      await this.storeResult(task, result);
    }

    // Send notifications
    await this.sendNotifications(task, result);

    // Update status
    await db.scheduled_tasks.update(task.id, {
      last_status: 'success',
      next_run: this.calculateNextRun(task.schedule)
    });

  } catch (error) {
    logger.error(`Scheduled task failed: ${task.name}`, error);

    await db.scheduled_tasks.update(task.id, {
      last_status: 'failure'
    });

    // Always notify on failure
    await this.sendFailureNotification(task, error);
  }
}

private async runTask(task: ScheduledTask): Promise<TaskResult> {
  if (task.template_id) {
    // Use template
    const executor = new TemplateExecutor();
    return await executor.executeTemplate(
      task.template_id,
      task.variables || {}
    );
  } else {
    // Run custom task
    return await orchestrator.executeTask({
      query: task.query,
      agents: task.agents,
      user_id: task.created_by
    });
  }
}
}
```

### Scheduled Task UI

typescript

```
// New tab: "Scheduled"
<ScheduledTasksView>
  <Header>
    <h1>Scheduled Tasks</h1>
    <button onClick={() => openCreateModal()}>
      + New Scheduled Task
    </button>
  </Header>

  <TaskList>
    {scheduledTasks.map(task => (
      <ScheduledTaskCard
        task={task}
        onToggle={() => toggleTask(task.id)}
        onEdit={() => editTask(task.id)}
        onRunNow={() => runTaskNow(task.id)}
        onViewHistory={() => showHistory(task.id)}
      >
        <TaskHeader>
          <Toggle checked={task.enabled} />
          <h3>{task.name}</h3>
          <StatusBadge status={task.last_status} />
        </TaskHeader>

        <Schedule>
          <Clock/> {humanizeCron(task.schedule)}
          <span className="text-muted">
            Next run: {formatRelative(task.next_run)}
          </span>
        </Schedule>

        <TaskStats>
          <Stat label="Runs" value={task.run_count} />
          <Stat label="Last run" value={formatRelative(task.last_run)} />
          <Stat label="Success rate" value={calculateSuccessRate(task)} />
        </TaskStats>

        <DeliveryMethods>
          {task.output_destinations.map(dest => (
            <Badge icon={getIcon(dest.type)}>{dest.type}</Badge>
          ))}
        </DeliveryMethods>
      </ScheduledTaskCard>
```

```
    ))}
  </TaskList>
</ScheduledTasksView>
```

### Example Scheduled Tasks

typescript

```
const EXAMPLE_SCHEDULED_TASKS = [
  {
    name: 'Weekly AI News Digest',
    schedule: '0 8 * * 1', // Monday 8am
    template_id: 'research-deep-dive',
    variables: {
      research_topic: 'AI developments from past week',
      depth: 'medium'
    },
    output_destinations: [
      { type: 'discord', config: { channel_id: '...' } },
      { type: 'library', config: { library_id: 'ai-news' } }
    ],
    notify_on: 'completion'
  },

  {
    name: 'Daily Stock Portfolio Analysis',
    schedule: '0 9 * * 1-5', // Weekdays 9am
    agents: ['researcher', 'analyst'],
    query: 'Analyze my stock portfolio performance and market trends',
    output_destinations: [
      { type: 'email', config: { to: 'user@example.com' } }
    ],
    notify_on: 'changes_only'
  },

  {
    name: 'Monthly Code Health Check',
    schedule: '0 0 1 * *', // First of month
    template_id: 'code-health-audit',
    variables: {
      repo_path: '/path/to/repo'
    },
    output_destinations: [
      { type: 'dashboard', config: {} }
    ],
    notify_on: 'always'
  }
];
```

## Context Switching & Focus Modes

### Schema

typescript

```typescript
interface WorkspaceContext {
  user_id: string;

  // Active work
  active_projects: string[]; // Max 3 pinned
  current_focus: string | null; // Currently open project
  focus_mode: FocusMode;

  // Quick access
  pinned_searches: SavedSearch[];
  pinned_libraries: string[];
  recent_documents: RecentDocument[];
  recent_notes: string[];

  // Preferences per mode
  mode_preferences: Record<FocusMode, ModePreferences>;

  // Session tracking
  session_start: string;
  time_in_focus: number; // seconds
  switches_today: number;
}

type FocusMode =
  | 'deep_work'     // No interruptions, auto-approve low-risk
  | 'research'      // Multi-tasking, lots of searches
  | 'planning'      // High-level thinking, diagrams
  | 'review'        // Going through results, approvals
  | 'casual';       // Normal mode

interface ModePreferences {
  notifications: 'all' | 'critical_only' | 'off';
  auto_approve_low_risk: boolean;
  auto_approve_medium_risk: boolean;
  preferred_models: Record<AgentType, string>;
  sidebar_collapsed: boolean;
  show_active_tasks: boolean;
}
```

### Focus Mode Behaviors

typescript

```
const FOCUS_MODE_CONFIG: Record<FocusMode, ModePreferences> = {
  deep_work: {
    notifications: 'critical_only',
    auto_approve_low_risk: true,
    auto_approve_medium_risk: false,
    preferred_models: {
      // Use fastest models to minimize interruption
      researcher: 'gpt-oss:20b',
      coder: 'gpt-oss:20b',
      // ... others
    },
    sidebar_collapsed: true,
    show_active_tasks: false
  },

  research: {
    notifications: 'all',
    auto_approve_low_risk: true,
    auto_approve_medium_risk: true,
    preferred_models: {
      // Use best models for quality research
      researcher: 'qwen3-next:80b-cloud',
      // ... others
    },
    sidebar_collapsed: false,
    show_active_tasks: true
  },

  planning: {
    notifications: 'all',
    auto_approve_low_risk: false, // Want to see all decisions
    auto_approve_medium_risk: false,
    preferred_models: {
      // Use reasoning models
      design: 'gemini-3-flash-preview-cloud',
      // ... others
    },
    sidebar_collapsed: false,
    show_active_tasks: true
  },

  review: {
    notifications: 'all',
```

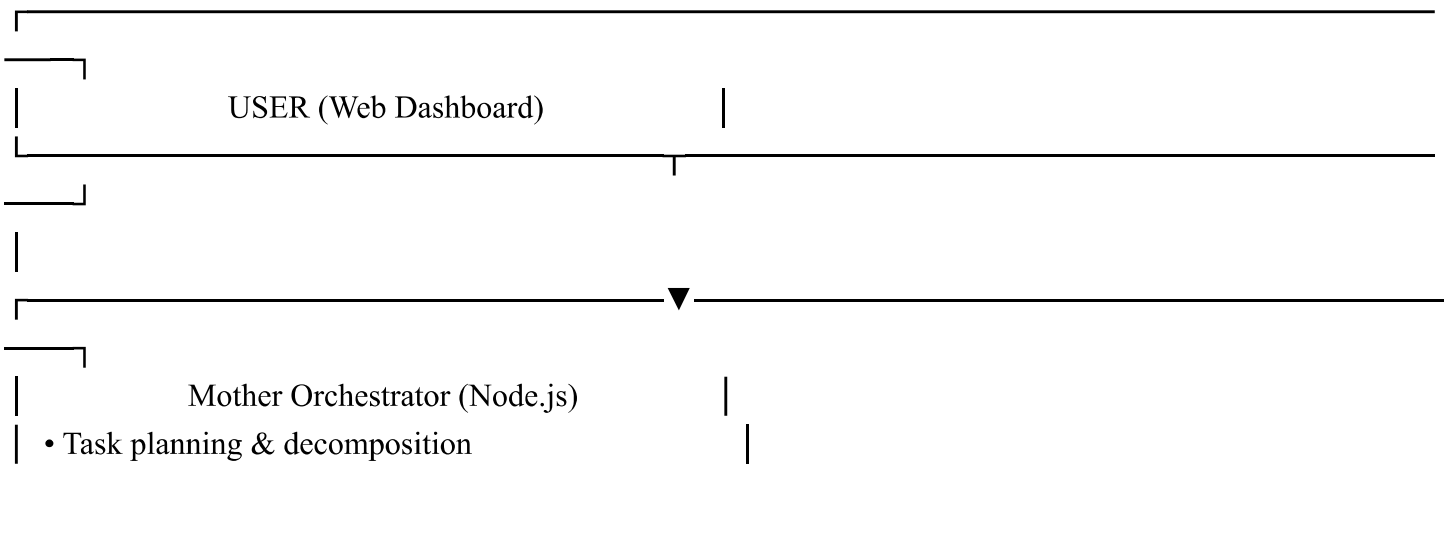
```
    auto_approve_low_risk: false,
    auto_approve_medium_risk: false,
    preferred_models: {
      // Standard models fine for review
      critic: 'deepseek-v3.1:671b-cloud',
      // ... others
    },
    sidebar_collapsed: false,
    show_active_tasks: true
  },

  casual: {
    notifications: 'all',
    auto_approve_low_risk: true,
    auto_approve_medium_risk: false,
    preferred_models: {}, // Use defaults
    sidebar_collapsed: false,
    show_active_tasks: true
  }
};
```

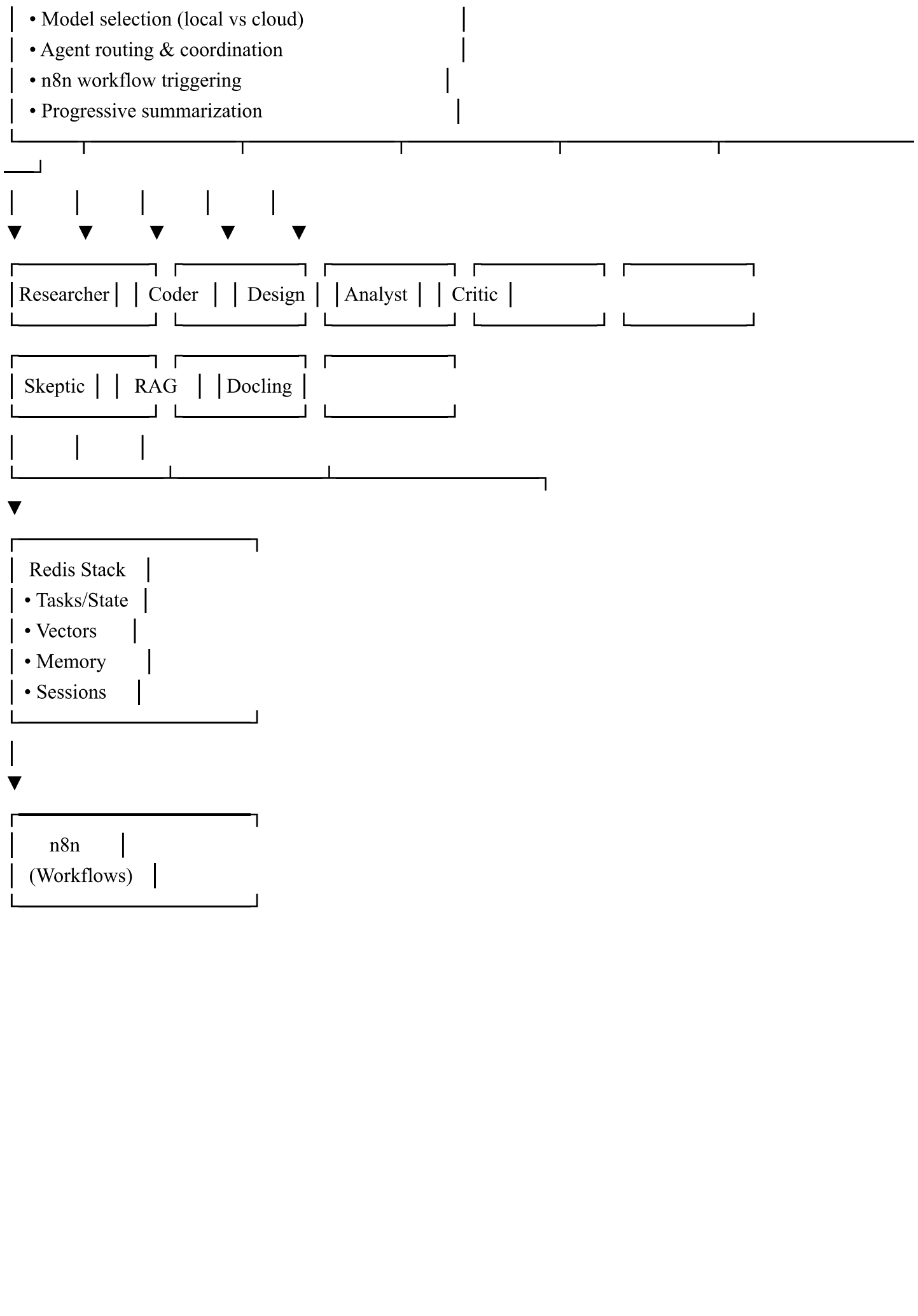
### Context Switch UI

typescript

```
// Top bar of dashboard
<ContextSwitcher>
  <FocusModeSelector>
    <ModeButton
      active={mode === 'deep_work'}
      onClick={() => setMode('deep_work')}
    >
      🎯 Deep Work
    </ModeButton>
    <ModeButton
      active={mode === 'research'}
      onClick={() => setMode('research')}
    >
      🔍 Research
    </ModeButton>
    <ModeButton
      active={mode === 'planning'}
      onClick={() => setMode('planning')}
    >
      📋 Planning
    </ModeButton>
    <ModeButton
      active={mode === 'review'}
      onClick={() => setMode('review')}
    >
      ✅ Review
    </ModeButton>
  </FocusModeSelector>

  <ProjectQuickSwitch>
    <Dropdown>
      <DropdownTrigger>
        <CurrentProject>{currentProject?.name}</CurrentProject>
        <ChevronDown />
      </DropdownTrigger>

      <DropdownMenu>
        <MenuSection label="Pinned Projects">
          {pinnedProjects.map(proj => (
            <MenuItem
              onClick={() => switchToProject(proj.id)}
              active={proj.id === currentProject?.id}
            >
```

```
          {proj.name}
          <Badge>{proj.status}</Badge>
        </MenuItem>
      ))}
    </MenuSection>

    <MenuSection label="Recent">
      {recentProjects.map(proj => (
        <MenuItem onClick={() => switchToProject(proj.id)}>
          {proj.name}
        </MenuItem>
      ))}
    </MenuSection># Mother-Harness: Complete Architecture Document

> **Unified multi-agent orchestration system with multimodal RAG, Redis Stack, Docling pipeline, and n8n integration**

---

## 🎯 System Overview

Mother-Harness is a three-tiered agent orchestration system designed for complex research, coding, analysis, and design tasks

- **Orchestrator (Mother)**: Plans, routes, and synthesizes
- **8 Specialist Agents**: Execute domain-specific tasks
- **Redis Stack**: Unified database (tasks, vectors, sessions, memory)
- **Docling**: Multimodal document ingestion
- **n8n**: Workflow automation + agent execution environment

---

## 🏗️ High-Level Architecture
```




• Model selection (local vs cloud)
• Agent routing & coordination
• n8n workflow triggering
• Progressive summarization
Researcher
Coder
Design
Analyst
Critic
Skeptic
RAG
Docling
Redis Stack
• Tasks/State
• Vectors
• Memory
• Sessions
n8n
(Workflows)

---

## 🖥️ Hardware Deployment (4-Node Homelab)

### **core1** (24 CPUs, 188GB RAM) - Primary Compute
**Containers:**
- Mother Orchestrator (Node.js)
- Redis Stack (full suite)
- Docling Service (async ingestion)
- Dashboard (Next.js)
- n8n (workflow engine)
- Agent Workers: Researcher, Coder, Design, Analyst, Critic, Skeptic, RAG

### **core2** (Ollama Host)
**Services:**
- Ollama server (local models)
- gpt-oss:20b (Q4/Q5 quantized) - default agent model

### **core3** (8 CPUs, 78.5GB RAM) - Overflow Compute
**Containers:**
- Additional agent worker replicas (if needed)
- Monitoring stack (Prometheus + Grafana)

### **core4** (Windows Server) - Storage
**Services:**
- File storage (document libraries)
- Media server
- Network shares (SMB/CIFS)

**File Structure:**

\core4\libraries
├── coding\ # Code repositories, tutorials ├── personal\ # Personal documents ├── financials\ # Finance docs ├── stocks\ # Stock research ├── media\ # Videos, audio (for media server) └── [custom]\ # User-defined libraries

---

## 🤖 Agent Roster & Model Assignments

| Agent | Purpose | Local Model | Cloud Model | Tools |
|---|---|---|---|---|
| **Orchestrator** | Planning, routing, synthesis | gpt-oss:20b | gpt-oss:120b-cloud | All agents, n8n |
| **Researcher** | Web research, documentation | gpt-oss:20b | qwen3-next:80b-cloud | Web search, RAG |
| **Coder** | Code generation, git ops | gpt-oss:20b | devstral-2:123b-cloud | Git, filesystem, n8n |
| **Design** | Architecture, UI/UX design | gpt-oss:20b | gemini-3-flash-preview-cloud | Mermaid, diagrams |
| **Analyst** | Data analysis, visualization | gpt-oss:20b | qwen3-coder:30b-cloud | Pandas, plotting |
| **Critic** | Verification, security review | gpt-oss:20b | deepseek-v3.1:671b-cloud | Static analysis |
| **Skeptic** | Business validation, devil's advocate | gpt-oss:20b | deepseek-v3.2-cloud | Research context |
| **RAG** | Document retrieval, synthesis | gpt-oss:20b | (local only) | Redis vectors, Docling |

**Model Selection Logic:**
1. **Default**: All agents start with local gpt-oss:20b (Q4/Q5)
2. **Upgrade triggers**: Task complexity, Critic flags low quality, user requests
3. **Cloud routing**: Orchestrator switches to cloud model for that specific task
4. **Cost awareness**: Track cloud token usage, warn user at thresholds

---

## 🧠 3-Tier Progressive Memory System

### **Tier 1: Recent Context (Verbatim)**
- **Storage**: Last 10 messages per project in Redis
- **Format**: Full conversation history with metadata
- **Retention**: Active session only
- **Access**: Always loaded for agent context

```json
{
  "project_id": "proj-123",
  "recent_messages": [
    {
      "id": "msg-1",
      "role": "user",
      "content": "Research vector databases",
      "timestamp": "2025-12-18T10:30:00Z"
    },
    {
      "id": "msg-2",
```

```
      "role": "assistant",
      "content": "I'll help research vector databases...",
      "agent_invoked": "researcher",
      "timestamp": "2025-12-18T10:30:15Z"
    }
  ]
}
```

### **Tier 2: Session Summaries (After Task Completion)**
- **Storage**: Redis JSON document
- **Trigger**: When task status = "completed" OR user closes project
- **Format**: Structured summary of findings, decisions, artifacts
```json
{
  "project_id": "proj-123",
  "sessions": [
    {
      "session_id": "sess-1",
      "started": "2025-12-18T10:30:00Z",
      "ended": "2025-12-18T11:45:00Z",
      "summary": "Researched vector database options. Found pgvector, Pinecone, Weaviate. Recommended pgvector for PostgreSQL integration.",
      "key_decisions": [
        "Use pgvector for embedding storage",
        "Implement hybrid search (vector + full-text)"
      ],
      "artifacts": ["report-vectordb.md"],
      "agents_used": ["researcher", "critic"]
    }
  ]
}
```

### **Tier 3: Long-Term Memory (Vector Embeddings)**
- **Storage**: Redis vector search (RediSearch + vector similarity)
- **Trigger**: Final report generation, important artifacts
- **Format**: Embedded chunks with metadata
- **Retrieval**: Semantic search when starting new related projects
```typescript
// Embedding storage
interface LongTermMemory {
  chunk_id: string;
  project_id: string;
```

```
  content: string;
  embedding: number[]; // 768-dim vector (gpt-oss:20b embeddings)
  metadata: {
    type: 'decision' | 'finding' | 'code' | 'artifact';
    session_id: string;
    timestamp: string;
    tags: string[];
  };
}
```

**Memory Flow:**

User Message
↓
[Tier 1] Recent 10 messages loaded
↓
Agent processes with full context
↓
[Tier 2] Session summary created on completion
↓
[Tier 3] Important findings embedded for future retrieval

---

## 📊 Redis Stack Schema

### **Database Structure**

#### **1. Tasks Collection**
```typescript
interface Task {
  id: string; // task-uuid
  project_id: string;
  user_id: string;
  type: 'research' | 'code' | 'design' | 'analysis' | 'mixed';
  query: string;
  status: 'planning' | 'executing' | 'approval_needed' | 'completed' | 'failed';

  // Planning
  todo_list: TodoItem[];
  execution_plan: ExecutionPlan;

  // Execution
  current_step: number;
  steps_completed: string[];

  // Results
  result?: TaskResult;
  artifacts: Artifact[];

  // Metadata
  created_at: string;
  updated_at: string;
  completed_at?: string;
}

interface TodoItem {
  id: string;
  description: string;
  agent: AgentType;
  status: 'pending' | 'in_progress' | 'completed';
  depends_on: string[];
  result?: any;
}
```

```

#### **2. Projects Collection**
```typescript
interface Project {
  id: string;
  name: string;
  type: 'research' | 'code' | 'design' | 'mixed';
  status: 'active' | 'completed' | 'archived';

  // Threads
  threads: string[]; // Array of task_ids

  // Memory
  recent_messages: Message[]; // Last 10
  session_summaries: SessionSummary[];

  // Metadata
  created_at: string;
  updated_at: string;
  last_activity: string;
}
```

#### **3. Chat Threads (Non-Project)**
```typescript
interface ChatThread {
  id: string;
  title: string;
  messages: Message[];
  summary?: string; // Generated when closed
  created_at: string;
  updated_at: string;
}
```

#### **4. Document Libraries (Vector Store)**
```typescript
interface DocumentChunk {
  id: string; // chunk-uuid
  library: string; // 'coding' | 'personal' | 'financials' | etc.
  document_id: string;
  document_name: string;
  file_path: string; // \\core4\libraries\coding\doc.pdf
```
```

```
  // Content
  content: string;
  embedding: number[]; // 768-dim vector

  // Multimodal
  images: ImageReference[];
  tables: TableReference[];

  // Metadata (from Docling)
  page_number?: number;
  section_title?: string;
  hierarchy: string[]; // ['Chapter 1', 'Section 1.1']
  chunk_type: 'text' | 'table' | 'figure' | 'code';

  // Timestamps
  indexed_at: string;
  source_modified_at: string;
}

interface ImageReference {
  id: string;
  file_path: string; // \\core4\libraries\.images\hash.png
  caption?: string;
  page_number?: number;
}
```

#### **5. Approvals Collection**
```typescript
interface Approval {
  id: string;
  task_id: string;
  project_id: string;
  step_id: string;

  type: 'file_write' | 'code_execution' | 'git_push' | 'workflow_creation' | 'api_call';
  description: string;
  risk_level: 'low' | 'medium' | 'high';

  // Preview
  preview: {
    files?: string[];
    commands?: string[];
```

```
  workflow?: any;
 };

 // Status
 status: 'pending' | 'approved' | 'rejected';
 response_notes?: string;

 created_at: string;
 responded_at?: string;
}
```

#### **6. Document Libraries Registry**
```typescript
interface Library {
 id: string;
 name: string; // Display name
 folder_path: string; // \\core4\libraries\coding
 description?: string;

 // Stats
 document_count: number;
 last_scanned: string;
 scan_status: 'idle' | 'scanning' | 'processing';

 // Config
 auto_scan: boolean; // File watcher enabled
 scan_schedule?: string; // Cron: '0 2 * * *' (2am daily)

 created_at: string;
 updated_at: string;
}
```

### **Redis Indexes**
```typescript
// Vector similarity search
FT.CREATE idx:documents
 ON JSON
 PREFIX 1 doc:
 SCHEMA
  $.embedding AS embedding VECTOR FLAT 6 DIM 768 DISTANCE_METRIC COSINE
  $.library AS library TAG
  $.document_id AS document_id TAG
```

```
    $.chunk_type AS chunk_type TAG
    $.section_title AS section_title TEXT
    $.content AS content TEXT

// Task search
FT.CREATE idx:tasks
  ON JSON
  PREFIX 1 task:
  SCHEMA
    $.project_id AS project_id TAG
    $.status AS status TAG
    $.type AS type TAG
    $.query AS query TEXT
    $.created_at AS created_at NUMERIC SORTABLE

// Project search
FT.CREATE idx:projects
  ON JSON
  PREFIX 1 project:
  SCHEMA
    $.name AS name TEXT
    $.status AS status TAG
    $.type AS type TAG
```

---

## 📁 Docling Pipeline

### **Service Architecture**

File Watcher (core1) ↓ Detects new/modified files in \core4\libraries* ↓ Publishes to Redis Stream: docling:jobs ↓ Docling Service (core1) ↓ Async Processing (polling pattern) ↓ Stores chunks in Redis + images in \core4\libraries.images
↓ Updates Active Tasks UI