TL;DR

- We propose a pragmatic agentic AI architecture for a mobile app that emphasizes clear structure, explicit goals, feasibility, and staged implementation. The design doc uses a formal report structure to present purpose, options, evaluation, recommendations, and rollout in a way that enables quick scanning by senior engineers and decision-makers [S6][S1][S7]. Costs and schedules are addressed toward the end, with recommendations grounded in feasibility criteria and implementation considerations [S6].

Problem

- Users want to delegate multi-step, real-world tasks to an application that can plan, call tools, and present safe, reversible outcomes.
- Plain “chat-only” models struggle with multi-step planning, tool usage, and approvals; an agent loop introduces structure, orchestration, and guardrails to improve reliability.
- Success criteria
- Completes representative multi-step tasks with minimal user iterations.
- Presents action previews and supports easy approval/undo.
- Meets explicit latency, privacy, and cost targets while remaining maintainable by an iOS team.

Goals / Non-Goals

- Goals
- Design an agent loop with planning, tool execution, error handling, and reflection.
- Define a typed tool surface and data contracts for safe action execution.
- Support short-term scratchpad/context and local retrieval for user data.
- Provide clear failure handling, observability hooks, test strategy, and rollout steps.
- Non-Goals
- Not selecting a specific model/vendor.
- Not prescribing platform-specific APIs/frameworks.
- Not defining product UX beyond essential guardrails (preview/approve/undo).

Background / Constraints

- This document follows a formal report structure to aid scanning and decision-making: executive summary (TL;DR), clear introduction of purpose/problem, background and constraints, options/evaluation, recommendations/implementation, costs/schedule, and references [S6][S1][S5][S7]. Sections are explicitly titled and organized for readability; longer sections use subheadings and bullets [S6][S1][S7].
- Constraints to consider during design (conceptual, not platform-specific)
- Functional: task planning, tool use, autonomy limits, multimodal I/O, preview/approve.
- Non-functional: latency, privacy posture, offline behavior, energy/thermal, cost.
- Delivery: ability to phase features, collect structured feedback, and iterate safely.
- Cost, schedule, and personnel considerations should appear toward the end, with costs as accurate as possible and projections clearly labeled; simple tabular formatting is preferred when detailing costs [S6].

Proposed Design

- High-level architecture (modular)
- Orchestrator: Implements the agent loop (plan → act via tools → observe → reflect → next step/finish).
- Tooling Layer: A registry of typed actions with strict input/output schemas, least-privilege access, and explicit user gating for sensitive operations.

- Context & Memory: Short-term scratchpad plus a local index to fetch user-relevant data; supports providing grounded context to the planner step.
- Model Gateway: Abstraction for LLM/VLM calls and structured outputs; supports streaming and function-call style outputs.
- Safety & Guardrails: Action previews, diff/approve flows, undo, allowlists, and autonomy limits.
- Observability: Private-by-design traces of the loop (no sensitive payloads), tool outcomes, and timing.
- Options and evaluation approach
- Present alternative deployment topologies and orchestration strategies; evaluate by feasibility, cost, maintainability, and performance (criteria-based evaluation is recommended in formal reports) [S6].
- Recommendation
- Start with a conservative autonomy level (request → plan → propose actions → user approve → execute), focusing on core tools and robust error handling, then expand autonomy and tool surface in later phases.

Data / Interfaces
- Tool contracts
- Each tool/action has a canonical name, version, typed parameters, preconditions, and a safe, structured response.
- Example fields (conceptual)
- name: string
- version: semver
- params: typed object (required/optional, constraints)
- permissions: required grants/consent gates
- dryRun(): preview result/diff
- execute(): effectful call
- response: { status, result, error?, auditMeta }
- Orchestrator interfaces
- plan(input, context) → plan steps
- decideTool(planStep) → tool name + params
- runTool(tool, params) → result or error
- reflect(state) → adjust plan or finish
- Model gateway interfaces
- complete(prompt, options) → stream tokens
- functionCall(prompt, schema) → structured output or call request
- Context/memory interfaces
- embedAndIndex(items) → ids
- retrieve(query, k) → items
- scratchpad: ephemeral state of current loop
- Data handling notes
- Keep schemas stable and versioned to prevent mismatch errors and ease rollout.
- Use dry-run/preview paths before effectful execution for reversible flows.

Failure Modes & Edge Cases
- Planning errors: incoherent or looping plans; mitigation: step caps, reflection budget, and fallback to user clarification.
- Tool misuse: wrong tool or params; mitigation: schema validation, preconditions, and allowlists.
- Partial failures: one tool fails mid-sequence; mitigation: transactional grouping where possible, resume/rollback paths, and user-facing diffs.

• Ambiguous permissions: missing grants; mitigation: request-gate and degrade to read-only previews.
• Oversized context: truncation or loss of crucial info; mitigation: prioritize salient items and constrain scratchpad.
• Non-deterministic outputs: unstable structured responses; mitigation: strict schema validation and retries with minimal changes.
• Timeout/latency spikes: mitigation: budgets per step, progressive disclosure/streaming updates, cancellation.

Performance & Cost
• Performance principles
• Constrain token/context budgets per step and cap loop iterations.
• Prefer streaming UI for perceived responsiveness.
• Batch or coalesce tool calls when safe; set explicit timeouts and fallbacks.
• Cost principles
• Track per-task compute and tool-invocation counts.
• Cache stable sub-results and embeddings where safe.
• Stage feature rollout to observe real-world usage before widening access.
• Placement and scaling
• Abstract the model gateway to support multiple backends or tiers over time without API churn; defer binding decisions to rollout stages.

Security & Privacy
• Data minimization and least privilege on every tool.
• Sensitive operations require explicit user approval with clear, auditable previews.
• Redact/avoid logging sensitive payloads; store only necessary metadata for diagnostics.
• Use allowlists for tools/actions; deny-by-default for unknown operations.
• Provide undo and clear recovery paths for effectful actions.

Observability
• Private-by-design telemetry events
• agent_loop_started/finished
• plan_generated (metadata only)
• tool_invoked/tool_succeeded/tool_failed (names, timings, statuses; no PII)
• reflection_applied
• user_approved/user_rejected
• Tracing
• Correlate steps within a task; capture timing budgets and retries.
• Dashboards
• Task success rate, average steps per task, tool failure rates, cancel/undo frequency, latency percentiles.

Testing Plan
• Deterministic prompt tests for core scenarios with golden traces of tool calls and approvals.
• Schema conformance tests: fuzz parameters and ensure strict validation/clear errors.
• Fault injection: timeouts, partial failures, permission denials.
• Regression suite for representative user scenarios; ensure stability of plans and previews across versions.
• Performance tests: cap-to-cap measurements per loop step and per tool invocation.
• Safety tests: ensure deny-by-default behavior and approval gates cannot be bypassed.

Rollout Plan

- Phase 0: Spikes and internal harness; define schemas and contracts; build a minimal tool and loop with previews.
- Phase 1: Dogfood with a narrow tool surface and strict approval gates; collect traces and iterate.
- Phase 2: Limited beta; expand tools, add caching and resilience; track success/latency/cost.
- Phase 3: General availability; widen autonomy where safe; finalize dashboards and SLOs.
- Scheduling and feasibility should be documented with interim and final milestones to provide confidence and progress tracking, per formal report guidance [S6].

Alternatives Considered

- Orchestration strategies
- Single-pass “plan then execute” vs. iterative ReAct-style loops.
- Pure function-calling vs. schema+router hybrid.
- Context strategies
- Minimal scratchpad vs. scratchpad + retrieval index.
- Stateless per-turn vs. short session memory.
- Guardrails
- Always-on approvals vs. risk-based gating by tool class.
- Deployment topologies
- Client-only vs. client + remote gateway.
- Rationale
- Chosen approach emphasizes safety, reversibility, maintainability, and incremental complexity growth; options can be revisited with data during rollout.

Open Questions

- What autonomy level is acceptable at launch versus after proving reliability with tracing and scorecards?
- Which tool categories provide the highest initial ROI given approval/diff UX and safety needs?
- How should we prioritize context sources and memory expiration policies to balance relevance and footprint?
- What caching policies (prompt/results/embeddings) best balance storage, staleness, and privacy for our use cases?
- Which evaluation metrics most strongly correlate with user trust and perceived productivity gains in agentic flows?
- What thresholds (latency/timeouts/step caps) optimize task completion without frustrating users in mobile conditions?
- How should we structure audit logs to maximize debuggability while minimizing sensitive payload exposure?
- What rollback/undo semantics are feasible for multi-step tool sequences with partial side effects?
- How do we phase permissions and disclosures so users understand capabilities without alert fatigue?
- Which telemetry events are essential for first release versus later performance and reliability tuning?

References

- [S1] Report definition and structure fundamentals; executive summary and body organization guidance.
- [S5] Report types and standard sections; writing steps and recommendations emphasis.
- [S6] Formal report anatomy, section ordering, evaluation criteria, cost placement, schedule/feasibility, and audience-aware positioning.

- [S7] Effective report traits (concise, well-structured, scannable) and standard formats.
- [S2] 403 retrieval note (not content-bearing).
- [S3] Outline usage as drafting tool; evidence-backed claims and organization levels.
- [S4] Source contained only a video title/footer (not content-bearing).
- [S8] Outline clarity, thesis guidance, and structured planning steps.