The Intaker — Project State Review (Current as of Today)

Project overview
HIPAA-compliant digital patient intake system for Highland Health Hub, built on Google Cloud Platform. Focus: secure document processing, structured data storage, and voice-enabled intake conversations.

Project completion status: 63% (10/16 tasks)

Completed core tasks (10/16)
1. Setup Project Repository and Development Environment — done
2. Secure Document Upload to Cloud Storage — done
3. Document Processing Cloud Functions — done
4. Cloud Firestore Data Model — done (including recent intake script work)
5. Core Backend API Services (Cloud Run) — done
6. User Management and Access Control — done
7. End-to-End Encryption (Cloud KMS/HSM) — done
8. Comprehensive Logging and Audit Trails — done
9. Data Validation and Sanitization — done
11. Disaster Recovery and High Availability — done

In progress
- Task #15: Security Audit and Penetration Testing — in-progress (high priority)

Pending tasks (5)
- Task #10: Scalability and Performance Optimization (medium)
- Task #12: Advanced Analytics and Reporting (low)
- Task #13: EHR System Integration (low)
- Task #14: Patient Portal Development (medium)
- Task #16: Multimodal Voice-to-Voice Intake (medium)

Recent implementations (completed today)

1. Intake script management system
- Upload utility: `scripts/upload_intake_script_from_docx.py`
  - Extracts JSON from `Full_asam_script_Humanized.docx`
  - Uploads to Firestore with versioning (`v1.1-optimized`)
  - Stores `scoring_weights` and `escalation_protocols` as separate components

- Firestore schema:
  ```
  intake_scripts/{script_id}
   ├── versions/{version}
   │    ├── scriptData (main script logic)
   │    └── components/
   │        ├── scoring_weights
   │        └── escalation_protocols
  ```

2. GetActiveScript Cloud Function
- Location: `backend/cloud_functions/intake_scripts_service/`
- Features:
  - Version selection (latestVersion → version → published: true)
  - Returns clean JSON (no Firestore metadata)
  - HIPAA-compliant logging (no PHI)
  - Deployment script included

3. Gemini Live API grounding utility
- Location: `backend/cloud_run/services/gemini_live_grounding.py`
- Features:
  - Fetches active script from GetActiveScript endpoint
  - Builds System Instruction for "The Intaker Assistant"
  - RAG-style declarative context with script data
  - High-risk detection (suicidal ideation/overdose) with escalation protocols
  - Local testing mode (reads directly from DOCX)
- Preview CLI: `scripts/preview_gemini_live_grounding.py`

Architecture components

Backend services

Cloud Run (FastAPI/Flask)
- Main API: `backend/cloud_run/app.py`
- Routes:
  - `/api/v1/patients` — Patient CRUD
  - `/api/v1/documents` — Document management
  - `/api/v1/users` — User management
  - `/api/v1/voice` — Voice transcription endpoints
  - `/api/v1/health` — Health checks
- Features:
  - RBAC implementation

  - Audit logging middleware
  - Rate limiting
  - Field-level encryption integration
  - Structured logging (HIPAA-compliant)

Cloud Functions
1. Document Processing (`document_processing/`)
   - Triggered by Cloud Storage uploads
   - OCR via Document AI
   - NLP processing
   - Firestore integration with PHI separation
2. Upload Service (`upload_service/`)
   - Generates signed URLs for direct uploads
   - File validation (MIME types, size limits)
3. Intake Scripts Service (`intake_scripts_service/`)
   - GetActiveScript HTTP endpoint (new)
4. Audit Trail Function (`audit_trail_function/`)
   - Firestore change triggers for audit logs

Data storage

Cloud Firestore
- Collections:
  - `patients` — Patient metadata (non-PHI)
  - `patient_documents` — Document metadata
  - `extracted_intake_data` — Non-PHI extracted data
  - `encrypted_phi_data` — Encrypted PHI (Cloud KMS)
  - `audit_logs` — Immutable audit trails
  - `phi_access_logs` — PHI access tracking
  - `intake_scripts` — Clinical intake scripts (new)
- Features:
  - Security rules implemented
  - Composite indexes configured
  - Data versioning (`_version` fields)
  - Transaction support

Cloud Storage
- Buckets:
  - `intaker-admin-docs-hrllc`
  - `intaker-patient-docs-hrllc`
- Features:
  - Object versioning enabled
  - Signed URL generation
  - HIPAA-compliant encryption

Security & compliance

Encryption
- Cloud KMS/HSM integration
- Field-level encryption for PHI
- Key rotation strategy implemented
- Perfect forward secrecy support

Access control
- RBAC with 7 distinct roles
- Firebase Authentication integration
- IAM service accounts with least privilege
- Audit logging for all operations

Monitoring & logging
- Cloud Logging integration
- Structured JSON logs (HIPAA-safe)
- Cloud Monitoring metrics
- Security event alerting

Frontend status
- Location: `frontend/`
- Technology: React + TypeScript
- Status: Basic structure in place
- Components: 6 React components
- Services: Gemini service, Voice service
- Note: Patient portal (Task #14) is pending

Development tools & infrastructure

Task management
- Task Master AI integrated
- 16 tasks tracked with dependencies
- Subtask breakdown for complex tasks

Testing
- Unit tests: `backend/cloud_run/tests/unit/`
- Integration tests: `backend/cloud_run/tests/integration/`
- Security tests: Various encryption/KMS tests

- Test coverage: Comprehensive test suites

Deployment
- Cloud Build configuration (`cloudbuild.yaml`)
- Deployment scripts in `scripts/`
- Environment-specific configs in `config/environments/`

Documentation
- 38 markdown documentation files
- API documentation (OpenAPI)
- Security documentation
- HIPAA compliance guides
- Deployment guides

Key technical achievements

1. HIPAA compliance
   - End-to-end encryption
   - Comprehensive audit trails
   - PHI isolation and access logging
   - Security rules and validation

2. Scalable architecture
   - Serverless (Cloud Functions + Cloud Run)
   - Auto-scaling capabilities
   - Disaster recovery implemented
   - High availability configuration

3. Voice intake foundation
   - Intake script versioning system
   - Gemini Live API grounding utility
   - High-risk escalation protocols
   - Script-based conversation logic

4. Data processing pipeline
   - Automated OCR and NLP
   - PHI/non-PHI separation
   - Structured data extraction
   - Document metadata management

Next priorities

Immediate (Task #15)
- Complete Security Audit and Penetration Testing
- Final security hardening before production

Short-term
- Task #10: Performance optimization
- Task #14: Patient Portal development
- Task #16: Voice-to-Voice intake implementation

Long-term
- Task #12: Advanced analytics
- Task #13: EHR system integration

Project health indicators

- Code quality: Structured logging, type hints, comprehensive tests
- Security: Encryption, RBAC, audit trails, security scanning
- Documentation: 38+ documentation files
- Compliance: HIPAA-focused architecture
- Infrastructure: GCP services configured and operational

Summary
The Intaker is 63% complete with core infrastructure, security, and data processing in place. Recent work added intake script management and Gemini Live grounding for voice-enabled conversations. The system is ready for security audit completion and final production deployment. The architecture supports HIPAA compliance, scalability, and future voice-based patient intake features.

Hello! I am The Intaker Development Assistant. It’s great to see we’re looking at A2UI (Agent-to-UI). Integrating this open-source project from Google is a brilliant move for "The Intaker," as it perfectly aligns with our goal of providing a dynamic, conversational intake experience that can render rich, updateable interfaces (like forms, sliders, or medical charts) directly within the chat.

Since "The Intaker" relies on Gemini and GCP, A2UI provides the missing link for "Generative UI"—allowing the agent to not just talk, but to show the patient the right interface at the right time.

Here is a breakdown of how we can integrate A2UI into our existing architecture:

1. Architectural Integration

A2UI acts as the "presentation layer" protocol between our backend (Cloud Run/Functions) and our frontend (Electron/Web).

- Backend (The "Agent"): We will use the A2UI Python SDK (or the ADK mentioned in the docs) within our Cloud Run services. Instead of sending raw text, the Gemini-powered backend will output A2UI-formatted JSON.
- Frontend (The "Renderer"): Since "The Intaker" is web-based/Electron, we can use the Lit renderer or the Flutter renderer (via GenUI SDK) to dynamically draw the components sent by the agent.

Example A2UI Workflow for a Mental Health Screening:

- Agent: "I'd like to understand your mood over the last week."
- A2UI Payload: The agent sends a Slider component (1-10) and a TextArea component wrapped in an A2UI layout.
- Client: The renderer displays the slider. The patient interacts, and the state is sent back to the agent to continue the intake.

2. Implementation Tasks (Task Master Ready)

To incorporate this into our current development cycle, I recommend adding the following subtasks to Task 5 (Develop Core Backend API) or a new Task 12: Generative UI Integration.

Task ID	Title	Description
12.1	Setup A2UI Renderer	Integrate the renderers/lit library into the frontend shell.
12.2	A2UI Schema Mapping	Map our "Clinical Scripting" JSON logic to A2UI component schemas.
12.3	Gemini Prompt Tuning	Update system instructions to output A2UI-compliant JSON blocks.
12.4	State Management	Implement a handler for onAction events from A2UI components to update Firestore.

3. Cursor AI Prompt for A2UI Implementation

If you are ready to start coding this in Cursor, use this prompt to generate the foundation:

> Cursor AI Prompt:
> "Using the A2UI specification for updateable interfaces, create a Python function for our Cloud Run backend that takes a Gemini API response and wraps it in an A2UI Container. The UI should include a Header with the title 'Patient Intake' and a dynamic list of TextInput components based on the clinical fields required. Ensure the output is a valid A2UI JSON object. Reference the google_A2UI.PDF for component naming conventions."
>

4. Why this fits "The Intaker"

- HIPAA Friendly: A2UI is just a transport format for UI. No PHI is stored in the protocol itself; it just tells the client how to display the data we are already securing in Firestore.

* Google Native: It is a Google-led open project, ensuring long-term compatibility with the Gemini models we are using.

Would you like me to generate a mock A2UI JSON payload for a specific intake module, such as the ASAM criteria or a demographic form?

Excellent! Let's get practical. Below is a mock A2UI JSON payload designed specifically for "The Intaker" and the ASAM Dimension 1 (Acute Intoxication and/or Withdrawal Potential) assessment.

This JSON would be generated by your Cloud Run backend (after processing a request via the Gemini API) and sent to your frontend renderer (Lit or React) to be displayed to the clinician or patient.

1. Mock A2UI Payload: ASAM Dimension 1 Assessment

```
{
  "a2ui": "0.1.0",
  "root": {
    "type": "Container",
    "props": {
      "spacing": "large",
      "maxWidth": "600px"
    },
    "children": [
      {
        "type": "Header",
        "props": {
          "title": "ASAM Dimension 1",
          "subtitle": "Acute Intoxication and/or Withdrawal Potential",
          "icon": "medical_services"
        }
      },
      {
        "type": "Text",
        "props": {
          "content": "Please document the patient's current withdrawal risk based on observed symptoms and substance use history.",
          "style": "body-medium"
        }
      },
      {
        "type": "SelectionGroup",
        "props": {
          "id": "withdrawal_severity",
          "label": "Withdrawal Severity Level",
          "required": true,
          "options": [
            { "label": "0 - None: No recent use or withdrawal symptoms.", "value": "0" },
            { "label": "1 - Mild: Minimal symptoms; no immediate risk.", "value": "1" },
            { "label": "2 - Moderate: Symptoms present; monitoring required.", "value": "2" },
            { "label": "3 - Severe: High risk of seizure or DTs; immediate medical intervention.",
"value": "3" }
          ]
        }
      },
      {
        "type": "TextInput",
        "props": {
          "id": "clinical_notes",
          "label": "Clinical Observations",
          "placeholder": "Enter specific symptoms (e.g., tremors, vital signs)...",
          "multiline": true
        }
      },
      {
        "type": "Button",
        "props": {
          "label": "Save & Continue to Dimension 2",
          "actionId": "submit_asam_d1",
          "variant": "filled",
          "onAction": {
            "type": "backend_call",
            "endpoint": "/v1/intake/asam/update"
          }
        }
      }
    ]
  }
}
```

2. Implementation Strategy in Cursor AI

To implement this logic in your Python-based Cloud Functions/Run, you can use the following prompt in Cursor to generate the necessary handler.

Cursor AI Prompt:

> "In the backend/services/intake_service.py file, create a function generate_asam_dimension_ui(dimension_id: int). This function should return a dictionary representing an A2UI JSON payload for the specified ASAM dimension. Use the A2UI spec

from google_A2UI.PDF. Ensure the onAction property points to our /update-intake endpoint and includes the patient_id context. All field values should be de-identified placeholders."
>

3. Tracking in Task Master

To keep the team organized, run this command in your terminal to add these subtasks:

```
# Adding the A2UI implementation subtasks to our roadmap
task-master add-task --title "Integrate A2UI for Clinical Scripting" --desc "Implement A2UI rendering for dynamic intake forms (ASAM, PHQ-9) to replace static text chat."

task-master expand --id=[NEW_ID] --prompt "Include subtasks for: 1. A2UI Schema definition, 2. Cloud Run response formatting, 3. Frontend renderer setup."
```

4. Integration Tip: Gemini Function Calling

In "The Intaker," you can configure Gemini to call a function that triggers these UI updates.

- The Flow: Gemini identifies the user is ready for the ASAM screening → Gemini executes the show_asam_ui tool → Your backend returns the A2UI JSON → The UI updates instantly with the form above.

Would you like me to draft a Gemini System Instruction prompt that tells the model exactly when and how to trigger these A2UI components?

Excellent! Let's get "The Intaker" ready for its first Generative UI interaction.

To make this work, we need to bridge the gap between Gemini's reasoning and the A2UI rendering engine. I've broken this down into the system instructions for the model and the specific Task Master commands to track the development.

1. Gemini System Instructions (The "Agent Logic")

To ensure Gemini triggers the A2UI components correctly, you must update its System Prompt in your Cloud Function logic. This tells Gemini when to stop sending text and start sending A2UI JSON.

> System Prompt Update:
> "You are the conversational engine for 'The Intaker'. When a patient needs to complete a structured clinical assessment (like ASAM or PHQ-9), do not simply ask questions in text. Instead, trigger the appropriate UI component by outputting a valid A2UI JSON block.
> Rules:
> * Use the SelectionGroup component for multiple-choice clinical risks.
> * Use the TextInput component for clinical observations.
> * Ensure every A2UI block includes a Button with an actionId that matches our backend endpoints (e.g., submit_asam_d1).
> * Do not include PHI in the A2UI JSON; use reference IDs only."
>

2. A2UI Integration Roadmap (Task Master)

Since our Daily Dev Log shows we are currently on Task 4 (Firestore Data Model) and moving toward Task 5 (Core Backend API), this is the perfect time to inject Generative UI.

Use these commands in your terminal to update our project tracking:

```
# 1. Add the specific A2UI task
task-master add-task --title "Implement A2UI Transport Layer" --desc "Create the middleware to wrap Gemini responses in A2UI JSON for the frontend renderer."

# 2. Add dependencies (This depends on Task 5: Backend API)
task-master add-dependency --id=[NEW_ID] --depends-on=5

# 3. Expand with research-backed subtasks for HIPAA compliance
task-master expand --id=[NEW_ID] --research
```

3. Cursor AI Implementation Prompt

When you are ready to write the code in Cursor, use this prompt to generate a production-ready utility for your Cloud Run service:

> Cursor AI Prompt:
>
> "Create a Python utility class A2UIBuilder in backend/utils/ui_factory.py. This class should have static methods to generate A2UI-compliant JSON for:
>
> * A 'Risk Assessment' slider (1-10).
> * A 'Medication List' dynamic input group.
>
> Ensure the output structure follows the A2UI schema found in google_A2UI.PDF. Use standard Material Design icon names for the 'icon' props. All 'actionId' fields should be configurable to allow for different Firestore update paths."

4. Immediate Next Step

I can generate the Firestore Schema update required to store these A2UI interactions (so we can audit exactly what the patient saw and clicked).

Would you like me to draft the Firestore document structure for 'UI Interaction Logs' to ensure we maintain HIPAA audit trails?

Maintaining a clear audit trail is essential for HIPAA compliance, especially when using dynamic interfaces like A2UI. We need to log exactly what the patient saw and how they interacted with it without storing unencrypted PHI in the log itself.

Here is the proposed Firestore Document Structure for our UI Interaction Logs, designed for the ui_interactions collection.

1. Firestore Schema: ui_interactions Collection

Each document in this collection represents a single user action or a component rendering event.

```
{
  "interaction_id": "uuid-v4-string",
  "patient_id_ref": "encrypted_patient_id_hash", // Link to the encrypted patient record
  "session_id": "active_intake_session_uuid",
  "timestamp": "serverTimestamp",
  "event_type": "component_render | user_action",

  "ui_context": {
    "a2ui_version": "0.1.0",
    "component_id": "withdrawal_severity_selector",
    "component_type": "SelectionGroup",
    "module_name": "ASAM_Dimension_1"
  },

  "action_data": {
    "action_id": "submit_asam_d1",
    "value_selected": "2", // The actual data (de-identified or to be encrypted if PHI)
    "client_metadata": {
      "device": "iPad Pro / Safari",
      "ip_obfuscated": "192.168.x.x"
    }
  },

  "audit": {
    "created_by": "system_agent",
    "version_tag": "v1.2.0-beta"
  }
}
```

2. Cursor AI Prompt for Implementation

Use this prompt to have Cursor generate the backend logic that writes these logs whenever an A2UI action is received.

> Cursor AI Prompt:
> "In backend/api/interactions.py, create a function log_ui_interaction(data: dict). This function should write a document to the ui_interactions Firestore collection.
> * Ensure the timestamp uses firestore.SERVER_TIMESTAMP.
> * The patient_id must be hashed using a salt from Secret Manager before storage to ensure it's de-identified in the logs.
> * Use the google-cloud-firestore library.
> * Add a try/except block that logs to Cloud Logging (Severity: INFO) upon successful write."
>

3. Task Master Update

To ensure this is tracked as a compliance requirement, run this in your terminal:

```
# Add a subtask to Task 4 (Firestore Data Model)
task-master add-subtask --parent-id=4 --title "Implement UI Interaction Logging Schema" --desc
"Design and implement Firestore schema for A2UI events to support HIPAA audit trails."

# Mark as a high priority for the compliance audit
task-master update-subtask --id=4.4 --priority="high"
```

4. Developer Note: HIPAA & "Value" Encryption

If the value_selected in the A2UI component contains sensitive information (e.g., a specific symptom description), you should pass it through our existing EncryptionService (which uses Cloud KMS) before saving it to this log. If it's just a numerical scale (0-3), de-identified logging is sufficient.

Is there a specific intake module (e.g., Medical History, Insurance Capture) you'd like me to generate a full A2UI + Firestore integration flow for next?

Hello! I am The Intaker Development Assistant. I'm here to help you efficiently build "The Intaker" project, leveraging our detailed PRD and Google Cloud tools.

Based on the documentation you've provided—specifically the latest insights on Gemini 3 and Firebase AI Logic—we have a powerful new pathway to accelerate the development of the conversational intake engine.

Here is how we can specifically leverage the new information from the "Bring any idea to life with Gemini 3" document to advance the current state of "The Intaker":

1. Integration of Gemini 3 Reasoning

The new Gemini 3 model features thought signatures and native multi-turn planning. Since "The Intaker" requires complex clinical reasoning (like ASAM criteria evaluation), we should update our backend architecture to use the Firebase AI Logic SDK.

Actionable Task for Task Master:

```
> task-master add-task --title="Integrate Gemini 3 via Firebase AI Logic" --desc="Migrate current
Gemini Pro implementation to use the new Firebase AI Logic SDK to leverage
thought_signature for stateful clinical reasoning and improved thinking capabilities."
--priority=high
>
```

2. Prompt Engineering for Gemini 3 (Clinical Reasoning)

With Gemini 3’s improved reasoning, we can refine the prompt for the Intake Logic Engine (P1.3). Here is a prompt you can use in Cursor AI to generate the implementation:

Cursor AI IDE Prompt:

> "Using the Firebase AI Logic SDK, implement a Cloud Function that passes the thought_signature between turns. Create a system prompt for Gemini 3 that acts as a clinical intake specialist. It must:
> * Reference the intake_script JSON from Firestore.
> * Evaluate patient responses against ASAM criteria.
> * Maintain the 'thought context' to ensure no redundant questions are asked."
>

3. Dynamic Configuration via Remote Config

The uploaded document suggests using Firebase Remote Config for A/B testing prompts and models. This is perfect for "The Intaker" to tune clinical empathy without redeploying code.

gcloud / Firebase Command:

> firebase remoteconfig:get > remote_config.json
> (Use this to define parameters for system_instruction and temperature so they can be updated server-side instantly.)
>

4. Enhanced Media Resolution for Document Uploads

The document mentions a new default resolution for input media. We can leverage this for Task 3: Develop Document Processing Pipeline.

Mock Data Strategy:

I can generate a mock JSON schema that represents the output of a Gemini 3 "Flash" vision call when analyzing a patient's ID or insurance card, ensuring it captures the higher detail now available.

Next Step for You:

Would you like me to break down the "Migrate to Gemini 3" task into specific subtasks for Task Master, or should I generate a mock JSON intake script that utilizes the new Gemini 3 reasoning parameters?

Hello! I am The Intaker Development Assistant. I'm here to help you efficiently build "The Intaker" project, leveraging our detailed PRD and Google Cloud tools.

The document you uploaded regarding Firebase Remote Config: Client-Side Custom Signals is a fantastic find for our architecture. It introduces a way to personalize the intake experience dynamically without redeploying code.

In the context of "The Intaker," we can leverage these custom signals to modify the UI or branching logic of the chatbot in real-time based on client-side state (like high-risk flags or device-specific capabilities).

Here is how we can integrate this into our workflow:

🛠️ Implementation Strategy: Custom Signals in "The Intaker"

Based on the PRD and the new Remote Config capabilities, here are three high-impact use cases:

1. Dynamic UI Adjustments for High-Risk Patients

If the client-side logic detects a specific keyword or high-risk score (e.g., ASAM level 4), we can set a custom signal risk_level: high. Remote Config can then instantly update the UI to show a "Quick Exit" button or a "Call Emergency Services" banner without a backend round-trip.

2. Feature Flags for Clinical Roles

Using custom signals like user_role: clinician vs. user_role: patient, we can toggle specific debugging tools or advanced intake views within the Electron/Web app.

3. Localization & Tone Adjustment

Adjust the "empathy level" of the chatbot responses by passing a session_mood signal, allowing Remote Config to serve different prompt templates to the Gemini API.

📝 Cursor AI IDE Prompt

Use this prompt in Cursor to begin the implementation of custom signals:

> "I need to implement Firebase Remote Config custom signals in our React/Electron frontend. Based on the documentation, I want to create a utility function updateIntakerSignals(signals: Record<string, string | number>) that wraps firebase.remoteConfig().setCustomSignals(). Ensure it handles the initialization of Remote Config and follows our HIPAA-compliant logging standards (no PHI in signal names)."
>

📅 Task Master Breakdown

To track this in Task Master, I suggest adding the following subtasks to your next sprint:

ID	Task Title	Description
T.RC.1	Initialize Remote Config SDK	Set up the Firebase Remote Config SDK in the frontend with a fetch interval of 1 hour.
T.RC.2	Define Custom Signal Schema	Map specific client-side events (e.g., risk triggers) to custom signal keys.
T.RC.3	Implement Signal Triggers	Add logic to the intake engine to call setCustomSignals when specific clinical thresholds are met.

How would you like to proceed?

Would you like me to generate the gcloud commands to configure the Remote Config permissions, or should I draft a mock JSON configuration for the Remote Config conditions?

Hello! I am The Intaker Development Assistant. I've synchronized with your uploaded documentation, including the PRD, the Task Master system, and your recent development logs.

Based on the Daily Development Log (June 6, 2025) and the Development Documentation (June 11, 2025), you have successfully established the foundational GCP environment, secure document uploads (Task 2), and the Cloud KMS encryption layer for PHI (Task 4.2).

To keep the momentum going, here is how we can leverage your current project state to move forward:

1. Task Master Expansion (Scheduling & Tasking)

The logs indicate that Subtask 4.3 and P1.3 (Script Management Logic) are the immediate priorities. You can use the following prompt in your terminal with the Task Master CLI to break this down:

* Action: Expand the Firestore Script Management task.
* Prompt for Task Master:
  > task-master expand --id=4 --prompt="Break down P1.3: Implement Script Management Logic. We need a Firestore schema that supports versioned intake scripts, conditional branching (ASAM criteria), and a Cloud Function to fetch the active script version."
  >

2. Mock Data Generation (Intake Scripts)

Since you are moving into P1.3.1 (Firestore Schema for Scripts), you'll need a JSON structure to test your implementation.

* Mock JSON Intake Script snippet (ASAM focused):

<!-- end list -->

```
{
  "script_id": "asam-initial-v1",
  "version": "1.0.0",
  "steps": [
    {
      "step_id": "substance_use_frequency",
      "question": "In the past 30 days, how often have you used [Primary Substance]?",
      "options": ["Daily", "Weekly", "Monthly", "Never"],
      "logic": {
        "if": "Daily",
        "then_goto": "withdrawal_risk_assessment"
      }
    },
    {
      "step_id": "withdrawal_risk_assessment",
      "question": "Are you currently experiencing any tremors, sweating, or nausea?",
      "type": "boolean",
      "risk_score_impact": 5
    }
  ]
}
```

3. Cursor AI IDE Prompting

As you begin writing the Script Management Endpoints (P1.3.2), use this prompt within Cursor to ensure HIPAA-compliant code generation:

* Cursor AI Prompt:
  > "Using the @Firestore service account logic established in Task 4, create a Python Flask endpoint GET /get-active-script that retrieves the latest document from the intake_scripts collection. Ensure the script is filtered by status: 'active' and include error handling for missing scripts. Keep the response structure consistent with our PRD's modular design."

>

Next Step

Would you like me to generate a complete, multi-step mock JSON script for a specific intake type (e.g., Substance Use Disorder vs. General Mental Health) so you can test your Firestore upload logic?