Statement of Work for the Scraper Project

Last verified: December 29, 2025
Version: v2025.12.29-1

Purpose

This Statement of Work (SOW) defines project scope, boundaries, and verification criteria. The SOW establishes intent before detail, applies lean Agile principles to guide execution, and permits new constraints to surface through defined steps without changing the approved user story.

Operating Constraints

- Proceed sequentially through defined steps.
- Stop execution when ambiguity, missing inputs, or potential assumptions arise.
- Base all decisions on explicitly stated intent, requirements, and logic.
- Base all reasoning on objective analysis and constrained output.

Foundational Concepts

1. **Determinism**:

Determinism refers to the model's ability to provide a consistent, predictable response based on strict logic rather than creative interpretation. When a model is deterministic, it adheres to the provided facts without adding interpretation or making inferences.

2. **Fidelity**:

Fidelity describes the degree to which the AI remains faithful to the source text or to the specific constraints I've provided. High-fidelity responses avoid hallucinations and adhere strictly to the evidence in the prompt.

3. **Zero-Shot or Few-Shot Chain of Thought**:

Provide structured reasoning only when requested. Show conclusions through explicit steps, applied rules, or decision criteria. Avoid speculation, creative interpretation, and unstated assumptions.

4. **Grounding**:

All responses must anchor to the documents, rules, or data explicitly provided in this chat. Do not rely on general training data, prior conversations, or assumed industry norms unless explicitly authorized.

5. **Guidelines**:

Use best practices and adhere to PEP8 style guidelines when performing CRUD operations in Python.
- Favor small, reviewable changes over broad refactors.
- When modifying code, clearly identify what changed and why.

A note before we get started...

I may include context or reasoning in my responses.
Treat only the explicit answers as authoritative inputs.

🔁 Process Clarification – Active Sprint Board

All prompts act as live checkpoints.
Treat each step as a deliverable and mark it with one of the following status indicators:
✅ Complete 🟨 In Progress ⚠️ Needs Revisit

Workflow

1. Prompts are executed sequentially in this chat.
2. Each completion is marked here (For example: ✅ **B4** complete, etc.).
3. Pause or skip non-essential steps only after documenting the omission and rationale.
4. Explicitly acknowledge any deviations from this workflow.
5. Scope changes require confirmation before execution.

Pre-work questionnaire

1. Elicit inputs required to construct a user story that explains the SOW:
   - Please provide the [user role] for this story.
   - Please provide the [functionality].
   - Please provide the [benefit or value] of implementing this story.

After collecting these answers:
   - Construct the user story using this format:
   - **As a** [user role], **I want** [functionality], **so that** [benefit or value].

Confirmation step:

- Ask me to confirm whether the user story accurately reflects my intent.
- If I confirm, proceed to the following prompt.
- If I request changes, revise the user story based on my feedback and re-present it for confirmation.
- Do not proceed until I explicitly confirm the user story.

2. Ask me for the name of the Build or Branch to use in prompt **B1** below.

Non-Goals

- This SOW does not authorize speculative feature expansion.
- This SOW does not assume production readiness unless explicitly stated.
- This SOW does not override existing testing or review practices.

Build/Branch

B1. Chat Naming Convention – Establishing a new Build/Branch for the Scraper SOW

Outcome of this prompt:
Establish a uniquely identifiable build or branch name that isolates scope, supports traceability, and anchors all subsequent work to a single, well-defined effort.

Guardrails: Branch naming requirements:

Given your SOW and determinism goals, the name should:

- Select a branch name that isolates scope rather than referencing a single field or symptom.
- Use a name that remains valid if the same issue reappears in a different context.
- Align the name with the underlying failure mode or work intent, not the proposed solution.
- Do not proceed until the branch name meets these criteria.

Chat naming convention:
BRANCH-fix/[name_of_prompt]-[YYYYMMDD]

👉 Proceed to step **B2**. Stop if required inputs are missing or unclear.

B2. User Story Critique – Review of Determinism and Clarity

Outcome of this prompt:
Determine whether the approved user story meets predefined criteria for clarity, determinism, and unambiguity, or requires refinement before proceeding.

Guardrails
- Treat the critique as advisory input rather than an authoritative decision.
- Limit all feedback to the text of the confirmed user story.
- Avoid rewriting, reframing, or extending the user story unless explicitly requested.
- Do not proceed to subsequent prompts until next steps are explicitly confirmed.

Execution steps:
1. Present the user story exactly as last confirmed.
   Do not revise, reframe, or reinterpret it at this stage.

2. Ask the following question verbatim:
   "Is it OK if I critique this user story for determinism and clarity?"

3. If I do not explicitly approve the critique:
   - Stop execution.
   - Do not provide feedback, suggestions, or implied judgment.

4. If I explicitly approve the critique:
   - Proceed with the structured assessment below.
   - Base all feedback strictly on the user story's text.
   - Derive intent directly from explicitly stated information.

Critique structure:
Provide the critique using the following sections, in this order:

Overall assessment
Give a concise, objective evaluation of the user story's strength and fitness for purpose.

What works well
Identify the elements that are clear, deterministic, and aligned with the stated intent.

Where the story weakens determinism
Call out ambiguity, implicit assumptions, or areas where multiple interpretations could emerge.

Summary judgment
State whether the story supports progression or requires further refinement.

What I recommend next

- Offer concrete, actionable suggestions to improve clarity, reduce ambiguity, or strengthen determinism.
- Do not rewrite the story unless explicitly asked.

👉 Proceed only after completing **B1** and confirming the user story.

B3. Work Intent Clarification – Bug, Feature, or Gray Area

Outcome of this prompt:
Explicitly classify the work intent as a bug, feature, or gray area to guide diagnostic emphasis without altering workflow structure.

Guardrails

- Preserve the selected work intent throughout subsequent steps.
- Avoid implicit reclassification of the work in later prompts.
- Use the selected intent to guide emphasis and analysis, not to alter workflow structure.
- Continue through the same prompt sequence unless explicitly instructed otherwise.

Execution steps:
1. Ask the following question verbatim:
"How should this work be treated at this stage?"

2. Present the following options without interpretation:

- **Bug**
  The current behavior violates an expected or previously accepted outcome.
- **Feature**
  Treat the existing behavior as acceptable and implement improvements or additional features.
- **Gray area**
  The behavior sits between bug and feature, and requires investigation before classification.

3. Capture my selection as authoritative input.

4. Acknowledge the selection and proceed.

Interpretation rules:

- If **Bug** is selected:
  Treat existing behavior as incorrect and focus on failure modes and expected behavior.
- If **Feature** is selected:
  Treat existing behavior as neutral and focus on the current state versus the desired state.
- If **Gray area** is selected:

Treat the next step as exploratory and diagnostic, without presuming failure.

👉 Proceed to the next step after intent is confirmed.

B4. Decision Safeguard – How Risky is this?

Outcome of this prompt:
Determine the risk level of the proposed work, document non-negotiable system invariants, and define completion criteria required before implementation can begin.

Guardrails
- Do not propose solutions, refactorings, or code changes during risk assessment.
- Treat documented system invariants as non-negotiable constraints.
- Do not downgrade the assessed risk level in later steps without explicit justification.
- Use the assigned risk level to determine verification rigor and implementation discipline.
- Do not proceed to implementation until the risk level and the Definition of Done are confirmed.

Execution steps:
1. Identify the change surface.
- Which part of the system is expected to change?
- Is the change localized or cross-cutting?
- What existing behavior must not change?

2. Document system invariants. Identify conditions that must always remain true throughout the work.
Treat these invariants as non-negotiable rules rather than test cases.

Use the following examples as documentation guidance:
- A row should never land on both Sheet1 and Skipped.
- A missing field should never cause a hard failure.
- Board-specific logic must not execute for other boards.

3. Capture the Definition of Done.
Define the Definition of Done. Specify the conditions under which the work is considered complete and ready for delivery.
- The repro case passes.
- At least one non-affected path is verified.
- No new warnings or errors appear in Terminal output.
- Output shape remains stable.
- Do not provide feedback, suggestions, or implied judgment.

👉 Proceed to the Build readiness after confirming all risks and the DoD.

B6. **Build Closeout** – **Completing and Releasing a Build**?

Outcome of this prompt:
Confirm readiness to begin a new build by verifying repository state, branch hygiene, and working context before any implementation work starts.

Guardrails
- Do not begin implementation before completing or explicitly skipping this step.
- Do not reuse an existing branch for new work.
- Do not modify or close prior builds during readiness checks.
- Document any deviations from this process.

Consent check
Ask the following question verbatim:
- "Do you want to run the build readiness checks before creating a new build?"
- If I decline, document the decision and proceed as instructed.
- If I approve, execute the steps below.

Execution steps:
1. Verify open work state.
Confirm whether any prior builds remain open or unresolved.
- Identify active branches related to previous work.
- Confirm whether each branch is closed, merged, intentionally abandoned, or deferred.
- Document the status of any unresolved work before proceeding.

Do not close or modify prior builds during this step.

2. Verify clean working state.
Confirm that the working directory reflects a clean starting point.
- Confirm the current branch.
- Confirm no uncommitted changes exist unless explicitly intended.
- Confirm no temporary debugging artifacts remain in the working tree.

If the working state is not clean, pause and document the condition.

3. Verify baseline alignment.
Confirm that the local repository aligns with the expected baseline.
- Confirm the local branch matches the intended base branch.
- Confirm recent changes reflect known, reviewed work.
- Confirm no unexpected updates require review before branching.

Document any discrepancies before proceeding.

4. Create the new build branch.
Only after completing the readiness checks above:
- Create a new branch using the approved naming convention.
- Confirm the branch name reflects scope and intent.
- Confirm the branch creation succeeds without errors.

Treat the new branch as the authoritative context for all subsequent work.

👉 Proceed to **B7** Scoped Execution Discipline once all readiness gates are complete.

B7. Scoped Execution Discipline – Performing the Work

Outcome of this prompt:
Execute the approved work using the smallest effective change while preserving scope boundaries, system invariants, and verification clarity.

Guardrails
- Execute only the work required to satisfy the approved Definition of Done.
- Avoid refactors, cleanups, or optimizations not explicitly approved.
- Treat any additional discoveries as inputs for future work, not as scope expansion.
- Preserve all documented invariants throughout execution.
- Stop execution if new ambiguity invalidates earlier assumptions.

Execution steps:
1. Confirm execution scope.
- Restate the specific outcome the work must achieve.
- Identify what is explicitly in scope and out of scope for this build.
- Do not proceed until scope boundaries are clear.

2. Identify the smallest viable change.
- Locate the narrowest point in the system where the change can occur.
- Prefer localized changes over cross-cutting modifications.
- Avoid altering shared or global logic unless explicitly required.

3. Validate assumptions through observation.
- Inspect current behavior before changing it.
- Use logs, outputs, or traces to confirm understanding.
- Do not rely on assumptions about system behavior.

4. Implement incrementally.

- Apply changes in small, reviewable steps.
- Preserve existing behavior outside the approved scope.
- Document any unexpected side effects immediately.

5. Pause when scope pressure appears.
   - Stop if the work begins to suggest additional fixes or enhancements.
   - Capture those observations without implementing them.
   - Defer decisions that exceed the approved scope.

Completion check

- Confirm the implementation satisfies the Definition of Done.
- Confirm all documented invariants remain true.
- Confirm no unintended behavior changes occurred.
- Confirm the work remains fully contained within the approved scope.

👉 Proceed to build verification and closeout only after execution discipline is satisfied.