

# StillSolutions Pilot Readiness Whitepaper

---

May 18, 2026

CONTENTS

StillSolutions Pilot Readiness Whitepaper .....

Executive Summary .....

Current Product State .....

Product Objective .....

Target Architecture .....

Backend Workstream .....

1. Canonical Data Model .....

2. CSV Ingestion and Normalization .....

3. Rules and Calculation Engine .....

4. Persistence and Multi-Tenant Data Boundaries .....

5. Export Service .....

6. Security, Privacy, and Compliance Readiness .....

Frontend Workstream .....

1. Guided Pilot Flow .....

2. Mapping Review Interface .....

3. Validation and Error States .....

4. Report UI .....

5. Export UI .....

6. Placeholder Cleanup .....

Dev Workflow and Automation .....  
Pilot Readiness Checklist .....  
Recommended Delivery Plan .....  
Phase 1: Make the Existing Demo Reliable .....  
Phase 2: Add Backend Structure .....  
Phase 3: Host the Pilot .....  
Phase 4: Controlled Supplier Pilot .....  
Success Metrics .....  
Risks and Mitigations .....  
Key Recommendations .....  
References .....

# StillSolutions Pilot Readiness Whitepaper

Backend and frontend work required before a supplier-facing pilot

Prepared for: Kisa and Daniel Wright

Meeting date: May 18, 2026

Source basis: Meeting transcript provided in the request, with current platform and healthcare privacy references checked where relevant.

## Executive Summary

StillSolutions is targeting a real operational pain in the DME POS supplier workflow: suppliers submit claims or order packets, discover too late that required documentation is missing, and then carry denial risk after inventory and service costs have already been incurred. The product direction discussed in the meeting is sound: take a supplier CSV, calculate risk and urgency, then produce a stoplight-style work queue showing which patients, prescribers, payers, or documentation gaps need attention first.

The current product is not ready for an external pilot yet. The visible UI is directionally useful and should not be heavily redesigned. The main issue is that the product still behaves like a local demo built around ideal sample data, not a hosted workflow that can survive messy supplier inputs, data privacy expectations, export requirements, and a non-technical pilot participant.

The next phase should focus on three things:

1. Build a reliable backend around CSV ingestion, normalization, report generation, persistence, and exports.
2. Turn the frontend into a guided pilot workflow: upload, map, validate, review, export, and share.
3. Put the app online with basic authentication, environment management, automated browser tests, and a repeatable development workflow using project-specific skills.

The most important nuance from the meeting is the CSV problem. A clean sample CSV is not enough. Every supplier will export data differently: headers vary, columns move, dates change format, payer names drift, patient IDs may be named differently, and documentation fields may be incomplete. If the product assumes a perfect CSV, the pilot may work once and fail with the next supplier. The immediate engineering goal is therefore not more UI polish. It is making the importer and backend tolerant enough to understand 25 or more realistic CSV variants that mean the same thing but look different.

## Current Product State

The meeting showed a dashboard-like interface with a stoplight report. The report appears to show patient or order records, urgency, status, priority, and recommended next actions. There is an import CSV action on the left and an export CSV action on the report page. Sample data could be imported, although it was not fully clear whether the UI refreshed reliably after import. The export action appeared not to work, and its intended downstream use was not yet fully defined.

The current UI has useful bones:

- The stoplight model is easy to understand.
- The report layout is simple enough for a supplier to scan.
- The product value is visible once sample data is loaded.
- Light mode / dark mode is not the key issue.
- The existing interface should mostly be retained for the first pilot.

But the current state also exposes pilot blockers:

- The product was not immediately ready to show on the call.
- The app appears to be run locally rather than hosted.
- There is no confirmed end-to-end flow from upload to validated report to export.
- The import flow relies too heavily on known sample data.
- The export flow does not appear wired.
- There is no clear supplier-system crosswalk yet.
- The UI contains placeholder content, including the bottom-left name reference, that should be removed before a pilot.
- There is no visible onboarding or data requirement screen for pilot users.

The conclusion is direct: StillSolutions has enough product shape to justify a pilot, but not enough operational reliability to put a supplier through the pilot without first hardening the back-end, data workflow, and testing loop.

## Product Objective

StillSolutions should become a hosted web application that helps DME POS suppliers identify documentation and authorization risk before revenue is lost.

The pilot version should support this simple promise:

A supplier uploads an order or claims-related CSV. StillSolutions normalizes the file, identifies missing or time-sensitive documentation, calculates urgency, and returns an actionable report showing which records need outreach first and why.

The pilot should not try to become a full practice management system. It should focus on one narrow workflow:

1. Upload supplier data.
2. Map and validate the file.
3. Calculate documentation risk.
4. Produce a spotlight report.
5. Export a work queue that can be used in the supplier's existing system.

That is enough to test whether StillSolutions saves time, reduces denial exposure, and helps suppliers act before deadlines are missed.

## Target Architecture

The meeting recommendation was to avoid packaging a desktop app. That is the right call. A desktop application creates avoidable packaging and support overhead across Windows, macOS, and Linux. A web app is the better path for a pilot because it gives one hosted surface, easier iteration, one login model, and one place to apply security controls.

The recommended pilot stack is:

- Frontend: Vercel or equivalent frontend hosting for the web UI.
- Backend: Railway or equivalent app hosting for ingestion, normalization, rules, and export services.
- Database and file storage: Supabase or equivalent managed Postgres plus controlled object storage.
- Authentication: Clerk or equivalent auth provider when the pilot moves beyond a controlled demo.

This stack should not be treated as permanent enterprise architecture. It is a pragmatic pilot stack: fast to ship, easy to inspect, and compatible with AI-assisted development workflows. Vercel documents support for common frontend frameworks and features such as routing, server-side rendering, static assets, framework support, and deployment surfaces. Railway documents deployment environment variables, Git-triggered metadata, and an MCP server that can be installed for OpenAI Codex and other coding tools. Supabase provides Postgres Row Level

Security guidance for browser-exposed data access and storage policies. Clerk Organizations provides a model for roles and permissions if StillSolutions needs supplier-level workspaces later.

For healthcare data, the stack choice must be validated before accepting real PHI. HHS states that the HIPAA Security Rule is about protecting electronic protected health information using administrative, physical, and technical safeguards. HHS also states that covered entities and business associates using a cloud service provider for ePHI need the appropriate business associate agreement and HIPAA compliance posture. This whitepaper is not legal advice. It means StillSolutions should not accept real production ePHI until security, contractual, and operational controls have been reviewed.

## Backend Workstream

### 1. CANONICAL DATA MODEL

Before improving the importer, define the canonical record that the backend needs. The sample CSV is not the system of record. It is just one possible input format.

The canonical order or patient-risk record should include, at minimum:

- Supplier account or organization ID.
- Upload batch ID.
- Source file ID.
- Patient identifier from supplier system.
- Optional patient display label for pilot/demo use.
- Device, item, SKU, HCPCS, or product category.
- Payer or plan.
- Prescriber or provider reference.
- Order date, supply generated date, dispense date, or service date.
- Documentation requirements known or inferred.
- Prior authorization status.
- Six-month visit or visit documentation status.
- Signed written order status.
- Other payer-specific required documentation status.
- Days left or days overdue.
- Computed risk status.

- Computed priority.
- Recommended action.
- Export status.
- Audit timestamps.

The canonical model should separate raw uploaded values from normalized values. This matters because users will need to understand what the system changed, inferred, or could not map.

## 2. CSV INGESTION AND NORMALIZATION

This is the highest-priority backend work.

The app should be tested against at least 25 mock CSV files. Each file should represent the same business meaning but use different column orders, header labels, date formats, optional columns, and messy values. Examples:

- `patient_id`, `Patient ID`, `Member ID`, `Acct #`, and `External Patient Ref` should be treated as possible patient identifiers.
- `payer`, `insurance`, `plan`, and `primary payor` should be mapped to payer.
- `device`, `item`, `product`, `DME`, `HCPCS`, and `equipment` may all describe the supplied item.
- Dates may appear as `MM/DD/YYYY`, `YYYY-MM-DD`, text dates, blank dates, or mixed formats.
- The file may contain extra columns that the app should ignore but preserve in raw data.
- Required columns may be missing and should trigger a validation screen instead of a silent failure.

The importer should have a deterministic path first, then an AI-assisted fallback only where needed.

Recommended flow:

1. Parse the CSV with a real CSV parser.
2. Detect headers and encodings.
3. Map headers to canonical fields using a synonym dictionary.
4. Validate required fields.
5. Normalize dates, payer names, device names, and identifiers.
6. Produce a mapping confidence score.

7. Ask the user to confirm uncertain mappings in the frontend.
8. Store both the raw upload and normalized records.
9. Generate the report from normalized records only.

AI can help infer ambiguous headers or suggest mappings, but core validation should be deterministic and testable. For pilot trust, the system should show what it inferred.

### 3. RULES AND CALCULATION ENGINE

The stoplight report needs explicit calculation logic. The UI should not merely display whatever status appeared in the CSV. As discussed in the meeting, status and priority should be calculated from the input facts.

The first rule engine can be simple:

- Red: documentation is missing, due now, expired, or close enough to deadline that immediate outreach is required.
- Yellow: documentation is incomplete or approaching deadline.
- Green: requirements appear complete or no urgent action is detected.
- Unknown: required inputs are missing, unmapped, or not trusted enough to calculate.

Each status should include a reason string. Example: `Red because signed written order is missing and the service date is inside the denial-risk window.`

The backend should also produce the recommended action:

- Contact patient.
- Contact prescriber.
- Request signed written order.
- Confirm six-month visit documentation.
- Verify prior authorization.
- Review payer-specific documentation.
- Exclude from automated scoring until required fields are mapped.

The calculation engine should be versioned. A report generated under rule version `v0.1` must remain explainable later, especially if a supplier challenges the output.

## 4. PERSISTENCE AND MULTI-TENANT DATA BOUNDARIES

For a real pilot, uploads, normalized rows, reports, and exports should be stored. A pure in-browser CSV reader is not enough because the product needs history, auditability, and repeatability.

Minimum backend tables:

- organizations
- users
- upload\_batches
- source\_files
- raw\_rows
- normalized\_records
- mapping\_decisions
- report\_runs
- report\_items
- export\_files
- audit\_events

If Supabase is used, Row Level Security must be enabled and designed deliberately. Supabase documentation says RLS should be enabled on exposed schemas, and service keys that bypass RLS should never be exposed in the browser. Storage access also needs policies at the bucket and object level. That means StillSolutions should avoid any shortcut where the frontend holds administrative keys or directly accesses cross-tenant data.

## 5. EXPORT SERVICE

The export button needs a clear purpose before the pilot.

Based on the meeting, the likely export target is a CSV work queue that a supplier can cross-walk into its own practice management system. That export should include:

- Supplier patient ID.
- Device or item.
- Payer.
- Prescriber reference where available.
- Status.
- Priority.

- Days left or overdue.
- Missing documentation reason.
- Recommended action.
- Notes field.
- Batch ID and report run ID.

The export should not invent patient phone numbers or sensitive details unless the supplier provided them and the system is cleared to handle them. For a safer pilot, keep the first export focused on IDs and work actions, allowing the supplier to join it back to its own system.

The export service should also support templates later:

- Generic work queue CSV.
- Prescriber outreach list.
- Patient outreach list.
- Payer follow-up list.
- Exceptions / unmapped records report.

## **6. SECURITY, PRIVACY, AND COMPLIANCE READINESS**

StillSolutions is operating in a healthcare-adjacent workflow. Even a pilot can involve sensitive data. The safest path is:

- Use synthetic data for internal testing.
- Use de-identified or limited data for first external demos where possible.
- Do not ingest real PHI until authentication, authorization, audit logging, encryption posture, retention rules, deletion rules, vendor agreements, and incident procedures are reviewed.
- Treat vendor BAA availability as a gating item before real ePHI is stored or processed.
- Keep admin/service keys off the frontend.
- Log access and export events.
- Add a data retention policy for pilot uploads.

The HHS guidance is the reason this belongs in the backend plan, not as late paperwork. Once real supplier data enters the system, infrastructure and process choices become compliance choices.

## Frontend Workstream

### 1. GUIDED PILOT FLOW

The frontend should be reorganized around a five-step flow:

1. Upload CSV.
2. Confirm field mapping.
3. Review validation issues.
4. Generate report.
5. Export work queue.

This can be done without changing the visual character of the existing dashboard. The current report UI is simple and useful. The missing piece is user guidance.

The upload page should answer:

- What file is expected?
- What fields are required?
- What fields are optional?
- What happens to the file?
- Is this demo data, synthetic data, or real supplier data?
- What should the user do if mappings are uncertain?

The app should not let a user jump straight from a questionable import to a confident report without seeing validation and mapping status.

### 2. MAPPING REVIEW INTERFACE

Because CSV variability is the core risk, the frontend needs a mapping screen. This is where the product can turn a messy supplier file into a trustworthy workflow.

The mapping screen should show:

- Original CSV header.
- Detected canonical field.
- Confidence level.
- Example values from the file.
- Required or optional status.
- User override dropdown.

For the first pilot, this does not need to be beautiful. It needs to be clear. A user should be able to see that `Acct #` has been mapped to `Patient ID`, or that `DME Type` has been mapped to `Device`.

### 3. VALIDATION AND ERROR STATES

The import button currently does not give enough visible confidence. The app needs strong states:

- Upload in progress.
- Import succeeded.
- Import succeeded with warnings.
- Import failed.
- Required fields missing.
- Unknown date format.
- Duplicate patient IDs.
- Empty file.
- Unsupported file type.
- Rows skipped.
- Rows needing manual review.

Every failure should tell the user what to do next. A pilot participant should never need to open developer tools or ask the founder what happened.

### 4. REPORT UI

The report UI should keep the stoplight model but add traceability.

Each row should include:

- Status color.
- Priority.
- Patient or supplier ID.
- Device / item.
- Payer.
- Date basis used for calculation.
- Days left or overdue.
- Missing or risky documentation.
- Recommended action.

- Reason code.

Add filters:

- Red only.
- Yellow only.
- Unknown / unmapped.
- By payer.
- By prescriber.
- By action type.

Add a detail drawer for each record showing:

- Raw uploaded values.
- Normalized values.
- Calculation reason.
- Mapping warnings.
- Export status.

This is important because a supplier will not trust a red/yellow/green output unless it can see why the system made that call.

## 5. EXPORT UI

The export button should become a short export dialog:

- Choose export type.
- Choose included fields.
- Confirm that IDs are intended for crosswalk into the supplier system.
- Download CSV.
- Store export event in audit log.

At minimum, the first export type should be `Work Queue CSV`.

## 6. PLACEHOLDER CLEANUP

Remove or hide placeholder identity and demo-specific content before any pilot. The bottom-left `S. Sullivan` style reference discussed in the meeting should not appear in a supplier pilot unless it is explicitly labeled as demo data. Placeholder names create confusion and reduce trust.

Demo mode should be explicit:

- Demo Supplier
- Synthetic Patient 001
- Sample payer
- Sample file generated for testing

That is cleaner than accidentally showing fake production-looking identities.

## Dev Workflow and Automation

The meeting exposed a repeatability problem: the app could not be quickly located, booted, and demonstrated. This is not just a personal workflow issue. It is a product-readiness issue because pilot reliability depends on repeatable setup.

Every stable workflow should become a project skill or script:

- Start frontend locally.
- Start backend locally.
- Seed sample data.
- Generate 25 test CSV variants.
- Run importer tests.
- Run export tests.
- Run browser pilot smoke test.
- Deploy preview.
- Verify production health.

The ideal smoke test should:

1. Boot the app.
2. Open the browser.
3. Upload a sample CSV.
4. Confirm mappings.
5. Generate the stoplight report.
6. Export the work queue.
7. Take screenshots.
8. Fail if expected values are missing.

This should run before any pilot meeting. The founder should not have to manually rediscover where the app is or how to run it.

## Pilot Readiness Checklist

StillSolutions should not schedule an external pilot user until these items are complete:

- App is hosted behind a stable URL.
- Demo login or controlled access is working.
- Synthetic sample data loads end to end.
- At least 25 CSV variants pass ingestion tests.
- Import flow shows mapping, validation, and warnings.
- Report generation calculates status and priority from backend logic.
- Each status has a reason and recommended action.
- Export CSV works and can be opened cleanly.
- Placeholder UI content is removed or clearly marked as synthetic.
- Browser smoke test passes.
- Data handling rules are documented.
- Real PHI is blocked until compliance and vendor requirements are reviewed.
- Pilot success metrics are written down.

## Recommended Delivery Plan

### PHASE 1: MAKE THE EXISTING DEMO RELIABLE

Timebox: 2 to 4 focused build sessions.

Deliverables:

- Open the project in an IDE and define repeatable start commands.
- Create a project skill or script to boot the app.
- Generate 25 mock CSV files.
- Build importer tests against those files.
- Fix import refresh behavior.
- Remove confusing placeholder UI.
- Wire a basic export CSV.

Exit criteria:

- A local browser smoke test can upload sample data, show the report, and export a CSV without manual debugging.

## **PHASE 2: ADD BACKEND STRUCTURE**

Timebox: 1 to 2 weeks depending on current codebase shape.

Deliverables:

- Backend service for upload, parse, normalize, score, and export.
- Canonical database schema.
- Upload batch persistence.
- Report run persistence.
- Rule versioning.
- Audit events.
- API endpoints used by the frontend.

Exit criteria:

- The frontend no longer relies on only local in-browser state for the core report workflow.

## **PHASE 3: HOST THE PILOT**

Timebox: 2 to 5 days after Phase 2 if the app is already structured cleanly.

Deliverables:

- Frontend deployed.
- Backend deployed.
- Database provisioned.
- Environment variables configured.
- Staging and production separation.
- Demo account or controlled access.
- Health checks.

Exit criteria:

- A pilot participant can use a URL, upload a permitted pilot file, review output, and export the work queue.

## PHASE 4: CONTROLLED SUPPLIER PILOT

Timebox: 2 to 4 weeks of observation.

Deliverables:

- Pilot guide.
- Data intake agreement and rules.
- Success metrics.
- Weekly review of false positives, false negatives, and unmapped files.
- Export usability feedback.
- Supplier-specific field mapping improvements.

Exit criteria:

- StillSolutions can show whether the product reduces manual review time, identifies documentation risk earlier, and produces a work queue suppliers will actually use.

## Success Metrics

The pilot should measure operational usefulness, not just whether the software runs.

Recommended metrics:

- Percentage of uploaded rows successfully mapped.
- Percentage of rows requiring manual mapping correction.
- Number of red/yellow/green/unknown outputs.
- Percentage of red/yellow outputs accepted as useful by the supplier.
- Number of recommended actions completed by staff.
- Time from upload to usable report.
- Time saved versus current manual review.
- Number of records that would have been missed without the tool.
- Supplier confidence score after report review.
- Export CSV usability score.

The product only wins if a supplier says: this tells me who to contact, why, and what to do next.

## Risks and Mitigations

CSV variability is the top technical risk. Mitigation: build the 25-file mock suite immediately and treat every importer bug as a product bug, not a data issue.

Compliance exposure is the top operational risk. Mitigation: use synthetic data until vendor agreements, auth, audit, retention, and privacy posture are reviewed.

False confidence is the top product risk. Mitigation: show reason codes, mapping confidence, and unknown states instead of forcing every record into red/yellow/green.

Export ambiguity is the top workflow risk. Mitigation: define the export as a work queue first, not as a full integration with every supplier system.

Founder workflow drag is the top execution risk. Mitigation: make skills and scripts for every repeatable action, including booting, testing, demoing, deploying, and generating sample data.

## Key Recommendations

Do not chase an external pilot yet. First, prove the full flow internally with synthetic data.

Do not redesign the whole UI. Keep the stoplight report and make the workflow around it clearer.

Do not depend on one sample CSV. Build 25 intentionally different CSVs and force the importer to survive them.

Do not package a desktop app. Use a hosted web app for speed, control, and lower support burden.

Do not ingest real PHI until data controls and vendor obligations are reviewed.

Do make every successful development workflow repeatable through scripts or skills. StillSolutions should never need to rediscover how to boot, test, or demo the product.

## References

- Meeting transcript: Kisa and Daniel Wright, May 18, 2026.
- HHS, [The HIPAA Security Rule](#).
- HHS, [Business Associates FAQ](#).
- Vercel, [Frontends on Vercel](#).

- Railway, [Railway MCP documentation](#).
- Railway, [Variables reference](#).
- Supabase, [Row Level Security](#).
- Supabase, [Storage Access Control](#).
- Clerk, [Organization roles and permissions](#).