How to build a prior authorization intelligence business without lighting money on fire: how to guide for bootstrappers

JAN 13, 2026

Abstract

This essay lays out a concrete business plan and a technical build guide for a pri authorization intelligence company designed to be bootstrapped, capital efficien boring in the best possible way. It is written for health tech investors and operat who have seen too many beautifully narrated decks die on contact with payer rea

Key ideas covered

• Why prior authorization is an information problem masquerading as an operat problem

• How to build a sellable product before touching machine learning, integration PHI

• A realistic path from a solo founder build to eight figure ARR

• What to actually ship in the first ninety days

• Where the real moat forms and where it absolutely does not

Table of Contents

Why prior authorization keeps getting treated like the wrong kind of problem

Why most prior authorization startups fail in the same boring ways

The wedge product that actually creates leverage

The business model that works without venture scale burn

What prior authorization rules actually look like when you read the source docu

Data sources that are public, fragmented, and good enough to build on

How public prior authorization policy behaves in the real world

A concrete example using imaging prior authorization

How a single CPT code turns into a revenue generating product

From policy PDFs to normalized rules without burning capital

The technical architecture that keeps this business cheap and boring

How prior authorization rules become RESTful JSON APIs

Delegated utilization management vendors and how they fit into the model

Building the rules engine before touching machine learning

Shipping a first version that people will actually pay for

Vibe coding the product without losing control of the system

How this becomes a large, durable business over time

The real problem hiding inside prior authorization

Prior authorization has developed a reputation as a bureaucratic nightmare, whi accurate but incomplete. The common framing treats it as a workflow problem. take too long. Portals are annoying. Staff spend hours on the phone. All true. Bu framing misses the more fundamental issue, which is that no one knows the rule enough confidence at the moment decisions are made. By the time a practice is submitting a prior authorization, the damage has already been done. The patient scheduled. The clinician has committed to a plan of care. The cost of being wror now real. That is why most prior authorization tooling ends up being a bandage than a cure. Speed helps, but it does not change the underlying economics.

From an investor's point of view, this distinction matters a lot. Workflow softwa tends to compete on features and price, and it bleeds margin as soon as humans required to keep things moving. Information infrastructure, on the other hand, compounds. If the product tells a practice up front that a prior authorization is required, or that the likelihood of approval is near zero, the entire downstream disappears. Fewer submissions, fewer denials, fewer appeals, fewer staff. That is where leverage lives. The uncomfortable truth is that most of the industry has b trying to optimize the wrong step.

Why most PA startups fail in predictab ways

The graveyard of prior authorization startups is already well populated, and it k growing. The failures are not mysterious. They follow a pattern. The first mistak trying to sell to health systems too early. Enterprise buyers want everything, integrations everywhere, guarantees, and pricing that assumes infinite patience. second mistake is leading with automation instead of insight. Automating a bad decision just makes the bad decision happen faster. The third mistake is reachin machine learning before understanding the data. Models trained on partial, bias outdated policy logic create a false sense of confidence that collapses under audi

There is also a subtler failure mode that investors sometimes miss. Many teams assume that payers are the ultimate customer. That sounds logical until one

remembers that payers already have the rules and have no incentive to make the easier to interpret. The buyers who feel the pain are providers and digital health operators who live downstream of opaque policy decisions. Any business that fo who is actually writing the check eventually wanders into a long sales cycle with exit velocity.

The wedge product that actually works

The wedge that works is unglamorous and narrow. It is a pre encounter prior authorization rules lookup. Not submission. Not tracking. Not appeals. Just a cl answer to a simple question before a patient is scheduled. Does this service, und this payer and plan, require prior authorization, and under what conditions. The product lives upstream of everything else and saves money by preventing work r than optimizing it.

This wedge is powerful because it does not require integrations to be useful. A practice administrator can run a lookup in a browser. A digital health company call an API during eligibility and benefits checks. A revenue cycle team can sani check a new service line before launch. None of this requires touching clinical d embedding inside an EHR. That keeps risk low and speed high.

Investors sometimes underestimate how much buyers will pay for a boring answ delivered early. The willingness to pay is not driven by delight. It is driven by av loss. If a tool prevents even a handful of unnecessary prior authorizations per we pays for itself almost immediately. That is the kind of ROI story that does not re a charismatic sales pitch.

The business model investors should n underestimate

This business works best as a subscription with optional usage based expansion. on, a flat monthly fee per practice or per organization keeps procurement simple hundred to fifteen hundred dollars per month is an easy yes for most specialty

practices if the value is clear. Digital health companies prefer usage based pricir which fits naturally once the core lookup logic is stable.

The path to scale is not about chasing thousands of tiny customers with heavy su needs. It is about standardizing the product enough that support disappears. A thousand practices paying an average of twelve hundred dollars per month is alr mid eight figure revenue business. A handful of enterprise API customers can ge there even faster. The margins are strong because the marginal cost of a lookup i close to zero once the data pipeline is built.

The technical architecture that keeps costs low

The technical architecture for this business is intentionally boring. That is a fea not a bug. At a high level, the system ingests publicly available policy documents normalizes them into structured rules, and evaluates those rules at query time. T is no need for real time integrations, streaming data, or complex orchestration. A simple batch pipeline and a deterministic rules engine go a long way.

A basic setup uses a crawler to fetch policy documents on a scheduled basis. Documents are hashed so changes can be detected without manual review. Parsi focuses on extracting the parts that matter, which are code lists, indications, exclusions, and effective dates. The normalized output is stored in a relational database where each rule is explicit and explainable. An API layer sits on top, exposing a single lookup endpoint. A lightweight user interface can be added fo technical users, but it is not the core product.

This architecture keeps cloud costs predictable and small. It also keeps debuggi sane. When a customer questions an answer, the system can point directly to the source policy and the logic applied. That transparency builds trust faster than a dashboard ever will.

Data sources that are public, messy, a sufficient

One of the reasons this business is attractive to bootstrap is that the raw inputs free. Major payers publish medical policy documents because they are required Delegated utilization management vendors publish their own criteria. Governm programs publish coverage determinations. None of this data is clean, consisten easy to work with, which is precisely why there is opportunity.

The key is not to boil the ocean. Most prior authorization volume is concentrate relatively small set of codes and specialties. Imaging, oncology, cardiology, and orthopedics cover a disproportionate share of the pain. Starting with one special a handful of payers dramatically reduces scope while still delivering meaningful The policies change slowly enough that weekly or even monthly updates are suff early on.

It is also important to accept that published policies describe intent, not reality. do not capture every edge case or operational quirk. That is fine. The goal is not predict every denial. The goal is to reduce uncertainty enough that bad bets are avoided. Over time, feedback from users can be used to refine the rules and flag policy language diverges from actual behavior.

How public prior authorization policy actually behaves in the wild

This section should start by deflating a common misconception. Public prior authorization policy is not hidden. It is just scattered, inconsistently written, an updated without ceremony. Large national insurers publish medical policy docu because they are required to. These documents usually live on provider portals c public facing policy libraries. They include lists of CPT and HCPCS codes, indications for coverage, exclusions, and language like “prior authorization is required for the following services when rendered in the outpatient setting.” Th language sounds definitive until one notices that the same payer delegates imagi

utilization management to a third party, whose clinical guidelines are published entirely different site, under a different update cadence.

The key point to drive home is that this mess is stable. It does not change daily. changes slowly and unevenly. That makes it perfect for a low capital business. T policies for imaging at a large national insurer might update quarterly. The dele vendor guidelines might update slightly more often, but still on the order of wee not hours. This means the data ingestion problem is batch oriented, not real tim also means correctness is about coverage, not perfection.

A concrete example using imaging pric authorization

This is where specificity matters. Pick one large insurer and one delegated utiliz management vendor. For imaging, this often looks like a national insurer delegat advanced imaging to a vendor that publishes modality specific guidelines for MI CT, and PET. The insurer's own policy document might say that MRI of the lum spine requires prior authorization for commercial plans. That document will list CPT codes, the general indications, and the effective date. On its own, that alrea answers a valuable question for a practice.

But now layer in the delegated vendor. That vendor publishes a guideline stating MRI of the lumbar spine is appropriate only after six weeks of conservative thera unless red flag symptoms are present. That guideline is public. It is written in cl language. It is not encoded in a machine readable way, but it is readable.

A small imaging center scheduling patients every day faces a decision. Schedule and fight later, or delay scheduling until authorization is secured. Both options money. A pre encounter lookup tool that says, in plain language, that this CPT u this insurer requires prior authorization, is governed by this delegated vendor, a will require documentation of conservative therapy unless certain diagnoses are present, is immediately useful. It prevents staff from guessing. It prevents unnec submissions. It prevents scheduling errors.

From a business perspective, this example shows why cash flow can start immed
The product does not need to predict approval outcomes. It does not need to int
with scheduling software. It simply needs to surface and normalize what is alrea
public. An imaging center paying a thousand dollars a month to reduce chaos ar
a high volume service is not doing charity. It is buying insurance against avoidab
waste.

From PDFs to production without burni capital

This section should walk through the pipeline slowly, almost tediously, because
investors who have built things appreciate boring clarity. Start with discovery.
Identify the handful of policy URLs that matter for the initial specialty and paye
combination. These URLs are stable. They can be monitored weekly. The docum
can be downloaded and stored with a hash so changes are detected automatically

Parsing does not need to be magical. Tables can be extracted. Headings can be
detected. Code lists can be pulled with regular expressions. Human review can b
used early to validate assumptions. This is not a place to over optimize. The
normalized output should be a simple internal representation of a rule. For exan
rule that says this CPT requires prior authorization for this payer and plan type,
a pointer to the governing document and its effective date.

Storage can be a relational database. There is no reason to introduce distributed
systems or exotic databases. Evaluation can be deterministic. Given an input of p
plan, CPT, diagnosis, and site of care, the engine evaluates applicable rules and
returns the result. The simplicity here is the moat early on. It keeps operating co
low and trust high.

Building the rules engine before the hy engine

There is a temptation to frame this problem as an artificial intelligence challeng Resist that temptation. The first version of the product should be entirely rules l Explicit logic beats opaque predictions when trust is on the line. A rule that say prior authorization is required because a code appears on a payer list is boring, l is defensible.

The rules engine itself does not need to be sophisticated. It evaluates a set of conditions against an input and returns a result. The complexity lives in the data modeling, not the execution. Each rule should be versioned, traceable, and testal When a policy changes, the diff should be visible. When a customer asks why an answer changed, the system should have a clear explanation.

Machine learning has a role later, particularly in estimating approval probability turnaround time. But those models are only useful once the underlying policy lo solid. Starting with ML too early just adds cost and confusion. Investors who hav lived through previous hype cycles will recognize this pattern immediately.

Shipping the first version without embarrassing yourself

The first version of this product should feel almost offensively simple. A single l form. A single API endpoint. A result that includes the answer, the source, and a confidence indicator. No fancy charts. No sprawling settings pages. The goal is t something into the hands of real users as fast as possible.

Pilots should be paid, even if the price is discounted. Free pilots attract the wro kind of feedback. Paying customers are motivated to tell you when something is wrong. Early adopters should be chosen carefully. A small specialty practice witl high volume of prior authorizations and a pragmatic administrator is worth mor a marquee logo that will never commit.

Trust building matters. Clear disclaimers should be present. This tool does not guarantee approval. It informs decisions. That honesty actually increases credibi Overpromising is what kills early trust in this space.

Go to market without pretending you a a platform

The go to market motion for this business is refreshingly direct. The buyer knov pain. The demo is short. The ROI story is obvious. Cold outbound works if it is targeted. Partnerships can wait. Content marketing can help, but it should be grounded in specifics, not generic complaints about the system being broken.

One of the biggest mistakes is trying to position this as a platform too early. Plat are expensive to build and hard to explain. This is a product. It does one thing w That is enough. Expansion can come later through APIs and embedded use cases the initial sale should be simple.

Investors should pay attention to sales efficiency here. This is not a company tha needs a large sales team to grow. A small number of disciplined sellers or even founder led sales can take it surprisingly far. That capital efficiency is part of th appeal.

Vibe coding, or how this actually gets built without a team of fifteen

This section should demystify the build process for non technical investors with insulting technical readers. Vibe coding in this context does not mean hacking something together recklessly. It means leaning into modern tooling that collaps distance between idea and implementation. A solo founder can describe the desi behavior of a policy ingestion script and generate a working draft in minutes. Th draft can be refined manually. The same applies to API scaffolding, basic user interfaces, and even test cases.

The important distinction is that vibe coding works best when the domain logic clear. Prior authorization rules are annoying but finite. Once the mental model i solid, the code follows. This allows a tiny team to move quickly without accumul massive technical debt. The architecture stays legible. Changes are understanda That matters when policy updates inevitably break assumptions.

Here is a concrete, end to end example of how publicly posted prior authorizatic rules get turned into a versioned ruleset and exposed as a RESTful JSON API tha developers can call.

The policy artifact being used in this example is UnitedHealthcare's "Radiology Notification and Prior Authorization CPT Code List for commercial and Individ Exchange plans," published on UHCprovider.com. In that document, CPT 73721 explicitly listed as a code that requires prior authorization for the covered plan t in scope.

How the rule gets developed from the primary source policy

A payer publishes a policy artifact that is effectively a machine extractable table code, description, modality, and the statement that the table contains the codes requiring prior authorization for the specified plan types. Engineers do not need infer anything complicated for the first pass. The rule is simply "if the member i the payer's commercial or Individual Exchange product lines, and the requested service code is on this list, then prior authorization is required." The document supplies the interpretation frame: it is not a clinical guideline; it is a requiremer intended for providers to determine whether prior authorization is required.

In practice, a rules company takes that artifact and translates it into a normalize internal representation that preserves three things the payer cares about and one thing customers care about. The payer cares about scope (which products the lis applies to), the effective period (when it is valid), and the procedural identifier (C or HCPCS). Customers care about provenance, meaning a linkable source and a version identifier, because nobody wants an argument about what the software "thought" the rule was. That provenance is your credibility moat early on.

A minimal normalized rule record for the CPT 73721 line item can look like this

Internal normalized rule object

```
{
“rule_id”: “uhc-comm-rad-pa-2025-11-04:cpt:73721”,
“payer”: “UnitedHealthcare”,
“line_of_business”: [“commercial”, “individual_exchange”]
“domain”: “radiology”,
“procedure_code_system”: “CPT”,
“procedure_code”: “73721”,
“procedure_description”: “MRI ANY JT LXTR C-MATRL”,
“requirement”: “prior_authorization_required”,
“modality”: “MR”,
“effective_start_date”: “2025-11-04”,
“effective_end_date”: null,
“source”: {
“type”: “pdf”,
“title”: “Radiology Notification and Prior Authorization Code List”,
“url”: “https://www.uhcprovider.com/content/dam/provider/docs/pu/prior-auth/radiology/COMM-Radiology-Prior-Notification-Authorization-CPT-Code-List.pdf”,
“publisher”: “UHCprovider.com”,
```

```
“extracted_at”: “2026-01-13”,

“evidence”: {

“match”: “73721 MRI ANY JT LXTR C-MATRL MR”

}

}

}
```

That specific “evidence” string is not just a nicety. It is how the product stays defensible when a customer disputes a result. The payer’s own document is the citation. In the UHC commercial radiology list, CPT 73721 appears as “73721 M ANY JT LXTR C-MATRL MR,” which is exactly the kind of line item a parser c reliably detect and normalize.

How the rule gets extracted and versioned in a low cost pipeline

The cheapest viable production approach is batch ingestion plus change detecti not continuous crawling.

Fetch. Download the PDF on a schedule and store it as an immutable blob. Com hash of the bytes. If the hash changes, treat it as a new version and run parsing a

Extract. Use a PDF table extractor or a text extraction pass that targets the tabul region. In this specific UHC artifact, the codes appear in a table where the CPT and a short description are adjacent, repeated across the page. The parser does n need to “understand” MRI. It needs to identify five digit CPT patterns and captu the adjacent description and modality.

Normalize. Convert each extracted row into a canonical rule record with consist fields across payers. Normalize payer identity, line of business, and domain. Stor

provenance.

Diff. Compare the newly extracted set of rules to the prior version for that same artifact. Produce a changelog: added codes, removed codes, description changes. changelog becomes a product feature later, but it also becomes internal QA now.

Test. Run a small suite of deterministic tests. For example: “if procedure_code is 73721 and payer is UnitedHealthcare and line_of_business is commercial, then prior_auth_required should be true.” This protects against silent parser regressi

The key business point is that none of this requires integrations, PHI, or a clinic model. It is document driven, deterministic, and explainable.

How the rules become a RESTful JSON API for app developers

App developers want a single call that returns an answer plus enough metadata t justify the answer.

A clean design is to separate “coverage requirement” from “clinical criteria.” Th example covers requirement only, because the UHC artifact is a code list stating which codes require prior authorization for the given scope.

Endpoint design

POST /v1/prior-authorization/requirements:lookup

```
Request JSON

{

“payer”: {

“name”: “UnitedHealthcare”,

“payer_id”: null
```

```
},
“member_context”: {
“line_of_business”: “commercial”,
“state”: “NY”
},
“service”: {
“procedure_code_system”: “CPT”,
“procedure_code”: “73721”
},
“rendering_context”: {
“place_of_service”: “outpatient”,
“site_of_care”: “freestanding_imaging_center”
}
}
Response JSON
{
“prior_authorization_required”: true,
“requirements”: [
{
```

```
“type”: “prior_authorization”,

“scope”: {

“payer”: “UnitedHealthcare”,

“line_of_business”: [“commercial”, “individual_exchange”]

“domain”: “radiology”

},

“service”: {

“procedure_code_system”: “CPT”,

“procedure_code”: “73721”,

“description”: “MRI ANY JT LXTR C-MATRL”,

“modality”: “MR”

},

“evidence”: {

“source_title”: “Radiology Notification and Prior
Authorization CPT Code List”,

“source_url”:
“https://www.uhcprovider.com/content/dam/provider/docs/pu
/prior-auth/radiology/COMM-Radiology-Prior-Notification-
Authorization-CPT-Code-List.pdf”,

“source_publisher”: “UHCprovider.com”,

“source_version_tag”: “PCA-7-25-02223-Clinical-QRG_110420
```

```
“matched_text”: “73721 MRI ANY JT LXTR C-MATRL MR”

},

“as_of_date”: “2026-01-13”,

“confidence”: 0.95

}

],

“explanations”: [

“CPT 73721 appears on the UnitedHealthcare commercial and
Individual Exchange radiology prior authorization CPT cod
list.”

]

}
```


Why this is developer friendly

It returns a boolean for workflow branching, but it also returns a structured evic object so downstream systems can display the rationale, log it, and survive comp reviews. The “source_version_tag” can be derived from the document header str that appears in the PDF, which is visible in the artifact text and helps with auditability.

How delegated utilization managemen vendors fit into the same API shape

Delegation usually means the payer’s code list tells you whether PA is required, the delegated vendor’s clinical guidelines tell you what documentation and indic

are expected for approval. Those become an adjacent endpoint, not a tangled one

For example, you might add

POST /v1/prior-authorization/criteria:lookup

That endpoint would return a documentation checklist and clinical criteria refe sourced from the delegated vendor's published guideline documents. eviCore, fo instance, publishes public clinical guidelines for categories like musculoskeletal spine imaging, which are explicitly intended for medical necessity review, and ca linked as evidence artifacts similar to the payer PDF.

In other words, the rule graph becomes two layers that the API exposes cleanly. layer answers "is PA required." The other answers "what must be true for appro

How this supports a low capital, cash flow from day one business model

The commercial trick is to launch before the second layer exists. A large portion buyer pain is simply uncertainty about whether PA is required and where to rou The UHC CPT list already enables a sellable feature: for a given payer scope, the software can deterministically answer that CPT 73721 requires prior authorizati and provide the payer's own artifact as evidence.

That can be monetized immediately as a web lookup tool for specialty practices imaging centers, with an API plan for digital health builders. The first customer paying for reduced scheduling chaos, fewer failed submissions, and fewer staff h wasted. The cost to serve is tiny because the pipeline is batch and the response i computed from cached rules.

How this can be vibe coded without becoming sloppy

Vibe coding works here because the domain can be constrained into small, testa units.

Start by vibe coding the ingestion script and the parser against a single known a like the UHC radiology list, then lock in a test fixture that asserts CPT 73721 is extracted and normalized correctly. That single test becomes a guardrail as more payers and documents are added. Because the core logic is deterministic, AI assi coding is mostly generating boilerplate and accelerating iteration, not inventing business logic.

A practical approach is to vibe code in layers.

One, generate the scaffolding for a document fetcher, storage, hashing, and a job runner.

Two, generate a first pass PDF extractor and immediately pin it to unit tests usi real snippets from the artifact, such as the exact “73721 MRI ANY JT LXTR C-MATRL MR” row.

Three, generate the API server and OpenAPI spec, then enforce contract tests so output schema stays stable as the ruleset grows.

Four, add a minimal admin UI for review and corrections, because the cheapest accuracy improvement early on is a human override tool rather than a clever mo

This is how a solo founder can ship something real quickly while still building t kind of evidence chain that sophisticated customers and partners require.

How this quietly grows into a large business

If the product works, growth comes from deepening rather than widening. Addi more payers. More specialties. More nuanced conditions. Over time, the dataset policy versions and user feedback becomes a strategic asset. Patterns emerge. Ce

rules are always challenged. Certain policies change frequently. Certain combina of payer and service are especially risky.

At that point, higher order products become possible. Alerts when policies chan Scenario modeling for new service lines. Approval likelihood estimates based on historical outcomes. These features increase value without fundamentally chang the architecture. They also increase switching costs.

From an investor's perspective, the most interesting thing about this business is boring it looks at the start and how sticky it becomes over time. There is no vira No network effect that fits neatly on a slide. Just a steady accumulation of trust a data in a corner of the system that everyone complains about and no one wants t

That is often where the best businesses hide.

3 Likes

← Previous

Nex

Discussion about this post