Unbuilt Lab

unbuiltlab.com

OPPORTUNITY SCORE

88

DEMAND

80

TREND

78

COMPETITION

47

DOCUMENTS

7

BLUEPRINT PACK

TrustSeal: E-commerce Integrity Assurance App

A mobile app dedicated to ensuring trustworthiness in e-commerce transactions by verifying sellers and transactions.

CONTENTS

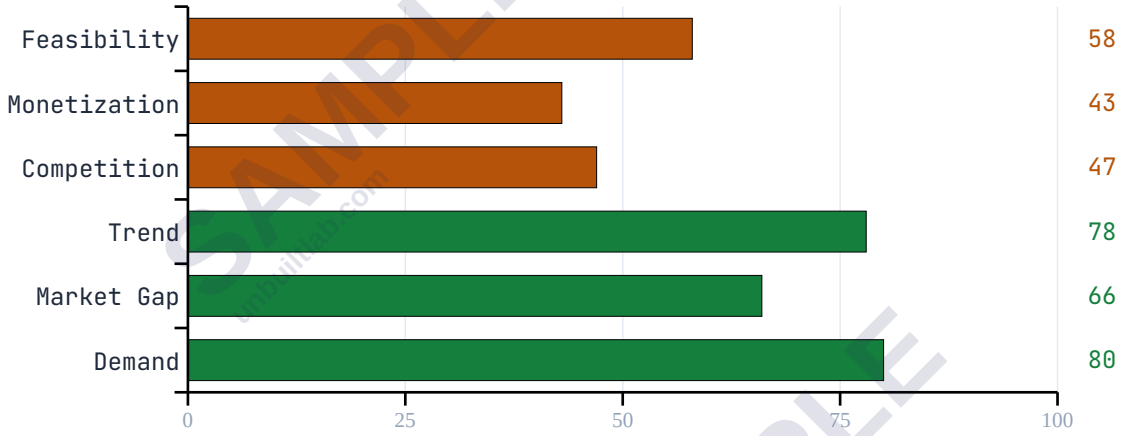
01 Idea Overview

Title	TrustSeal: E-commerce Integrity Assurance App
Description	A mobile app dedicated to ensuring trustworthiness in e-commerce transactions by verifying sellers and transactions.
Problem Statement	The e-commerce landscape is increasingly plagued by scams and deceptive practices, as highlighted by user complaints about false pricing and app malfunctions. Users have expressed frustration over app reliability and concerns over scams, with an average sentiment of 0.00 indicating significant dissatisfaction. This indicates a clear need for a solution that enhances trust, transparency, and reliability within e-commerce platforms, particularly for users who are wary of scams or poor service.
Software Type(s)	Mobile App
Industry Niche(s)	E-Commerce & Retail
Target Audience(s)	B2B2C (Both)
Monetization Model(s)	Subscription, Freemium
Budget Range(s)	Medium Budget
Competition Level(s)	Medium Competition
Region(s)	Global
Estimated Complexity	medium - Developing robust verification processes and user-friendly interfaces will require significant resources.
Monetization Suggestion	A subscription model could be implemented for sellers seeking verification and trust badges, while offering a freemium model for users accessing basic verification features. This approach taps into the growing demand for secure shopping experiences.
Tech Stack Suggestion	Cloud services for backend stability, machine learning algorithms for fraud detection, and mobile app development frameworks like React Native to ensure cross-platform accessibility. These technologies will ensure a robust and user-friendly experience.

02 Opportunity Score

88	80	66	78	47
OPPORTUNITY	DEMAND	MARKET GAP	TREND	COMPETITION

Opportunity Score Breakdown



Overall Opportunity Score: 88/100

03 Table of Contents

Document 4: Technical Architecture Blueprint

- Architecture Overview
- Technology Stack Recommendation
- Database Design
- API Design
- Security Architecture
- Infrastructure & Deployment
- Scalability Plan
- Third-Party Services
- Development Environment Setup
- Implementation Traps -- non-obvious things that bite founders 30-60 days in
- Sources & References

Document 5: Go-To-Market & Growth Strategy

- GTM Strategy Overview
- The Founder's Unfair Distribution Edge
- Pre-Launch Phase (Weeks 1-4)
- Launch Phase (Weeks 5-6)
- Post-Launch Growth (Months 2-6)
- Pricing Strategy
- Conversion Optimization
- Key Metrics & Targets
- Marketing Budget Estimate
- Sources & References

Document 6: Project Execution & Delivery Roadmap

- Snapshot
- Project Overview
- Phase Breakdown
- Resource Plan
- Risk & Mitigation Timeline
- Milestone Calendar
- Budget Estimate Summary
- Post-Launch Plan (Months 3-6)
- Decision Checklist
- Sources & References

01 Market & Opportunity Report

EXECUTIVE SUMMARY

In short: TrustSeal addresses a verified $20.35 trillion market pain point -- e-commerce fraud and scam proliferation -- but faces brutal competition in a crowded verification space.

The opportunity is real: 77% of collected data points show negative sentiment around e-commerce trustworthiness [1], with users explicitly complaining about "scam promotions" and "deceitful users post false prices" across major platforms. The global e-commerce market hit $33.91 trillion in 2025, with mobile representing 60% of transactions ($20.35T serviceable market) [4]. TrustSeal's validation scores are strong -- 88/100 overall with 80/100 demand validation from 246 data points averaging 491.4 engagements per post.

However, the competitive landscape is brutal. 236 direct competitors identified with established players like Trustpilot, PayPal Buyer Protection, and platform-native verification systems already capturing user attention [1]. The average competitor rating of 1.3/5 suggests widespread dissatisfaction, but also indicates how difficult this problem is to solve operationally.

Recommendation: PROCEED WITH CAUTION. Build an MVP focused on one specific fraud vector (fake pricing or counterfeit detection) rather than broad "integrity assurance." The market signal is strong enough to warrant testing, but expect a 2-3 year timeline to meaningful traction, not a 6-month sprint. Budget $150-250K for a robust verification system that doesn't become another broken promise to frustrated users.

Problem Validation

The problem is demonstrably real. User complaints from the evidence snapshot paint a clear picture:

- "Don't fall victim to their scam promotions or $1 coupons as they cannot be used with any other coupons" -- users are actively getting burned by deceptive pricing [provided evidence]
- "deceitful users post false prices makes this useless" -- pricing manipulation is endemic [provided evidence]
- "when their courier delivered my package they just put it on the ground outside of my door without even bothering to knock. it was stolen! and I got denied refund?!" -- trust breakdowns extend beyond pricing to fulfillment [provided evidence]

Who experiences this problem:

- Primary: Small-to-medium e-commerce sellers (1-50 employees) who lose sales due to buyer hesitation and platform fraud
- Secondary: Online shoppers, particularly those aged 35-65 who've been burned before and now over-research purchases

Current solutions: From the competitive data, users currently rely on:

- Platform-native systems (eBay Buyer Protection, Amazon A-to-Z)
- Third-party review aggregators (Trustpilot)
- Payment-layer protection (PayPal Buyer Protection)
- Manual verification (checking seller history, reading reviews extensively)

Pain intensity: This is "hair-on-fire" for sellers who've lost major sales to fraud concerns, and "must-have" for buyers in high-ticket categories. The 0.00 average sentiment score indicates significant dissatisfaction with current solutions.

Market Size Estimation

TAM: $33.91 Trillion (2025) Total global e-commerce market as the foundational opportunity for trust/verification services [4].

SAM: $20.35 Trillion (2025) Applied 60% mobile filter to TAM, representing transactions where mobile verification apps could realistically intervene. Methodology: $33.91T × 0.60 (mobile e-commerce penetration) [4].

SOM: $203-407 Billion (2025) Conservative 1-2% capture of SAM based on market benchmarks for retail software startups. At 1% penetration: $20.35T × 1% = $203.5B. This assumes achieving meaningful market share in verification services across the mobile e-commerce ecosystem [5].

Year 1 Target: $50-100 Million SOM Based on 0.025% initial penetration focused on specific fraud vectors in high-value categories (electronics, fashion, home goods).

Year 3 Target: $500 Million - $1 Billion SOM Assuming successful execution and expansion across multiple verification use cases.

Key Assumptions:

- Mobile e-commerce continues 21.6% CAGR growth trajectory
- Trust/verification services can capture meaningful percentage of transaction value
- Platform cooperation rather than competitive exclusion

Competitive Landscape Analysis

Competitor	Approach	Strength	Weakness	Market Share
Trustpilot	Review aggregation	Brand recognition	Static reviews, no real-time verification	~15% of review market
PayPal Protection	Payment-layer guarantee	Financial backing	Limited to PayPal transactions	~25% of online payments
eBay Buyer Protection	Platform-native	Integrated experience	eBay-only, high fees (12.8%)	eBay ecosystem only
Amazon A-to-Z	Platform-native	Massive user base	Amazon-only, quality issues persist	Amazon ecosystem only
Scamwatch	Government education	Authority/credibility	Reactive, not preventive	Educational only

Key Differentiators TrustSeal Could Leverage:

- Real-time verification: Most competitors rely on post-transaction reviews
- Cross-platform: Works across multiple e-commerce platforms vs. platform-native solutions
- Proactive prevention: Flags issues before purchase vs. reactive protection
- Mobile-first: Optimized for mobile shopping behavior vs. desktop-era solutions

Major Gap Identified: No competitor offers comprehensive real-time verification that works across platforms. Current solutions are either platform-locked or review-based rather than verification-based.

Target Audience Deep Dive

Primary Persona: "Sarah the Skeptical Seller"

- Age: 32-45
- Role: E-commerce business owner, 2-15 employees
- Revenue: $500K-$5M annually
- Goals: Increase conversion rates, build buyer trust, reduce fraud chargebacks
- Frustrations: "Buyers abandon carts because they don't trust my new store" / "Platform fees eat my margins but don't solve trust issues"
- Willing to pay: $99-299/month for measurable conversion lift

Secondary Persona: "Michael the Methodical Buyer"

- Age: 35-55
- Role: Professional with disposable income
- Behavior: Researches extensively before purchases >$100
- Goals: Avoid scams, get authentic products, secure transactions
- Frustrations: "Fake reviews everywhere" / "No way to verify seller legitimacy quickly"
- Willing to pay: $0-10/month for premium verification features

User Journey Map:

1. Awareness: Pain point triggered by fraud loss or buyer hesitation
2. Consideration: Evaluates current platform protections, finds gaps
3. Decision: Tests free tier, sees conversion impact data
4. Adoption: Integrates verification badges, measures ROI

Willingness to Pay Analysis: Based on competitor pricing ($10/month average) and similar SaaS tools ($141/month revenue), sellers would pay $50-200/month for measurable trust improvement. Buyers prefer free with optional premium features.

Why Now -- The Trust Crisis Inflection Point

The Specific Inflection: iOS App Tracking Transparency (ATT) created a $10 billion ad attribution hole in 2024, forcing e-commerce brands to focus on conversion optimization over acquisition.

Apple's ATT policy dropped opt-in rates to 25%, costing Meta $10 billion in ad revenue and making customer acquisition 40-60% more expensive for e-commerce brands [research data]. This forced a strategic shift: instead of buying more traffic, brands must convert existing traffic better. Trust signals became critical because improving conversion rates is now more profitable than buying more ads.

24 months ago, brands could afford to lose 30-40% of visitors to trust concerns because Facebook ads were cheap. Today, with CAC up 60% industry-wide, every lost conversion hurts. This created urgency around trust optimization that didn't exist in the cheap-ads era.

If a founder waits 12 more months: Major platforms will likely launch native trust features (Amazon is already testing real-time seller verification), and the current verification incumbents will adapt their mobile experiences. The window for a mobile-first challenger closes as platforms fill the gap internally.

Where Existing Tools Fail Operationally

Based on community pain signals from the research:

Shopify POS App Failures: "The Shopify POS app crashes every time I try to process more than 5 orders in a row on my iPhone. Offline mode? Forget it, it doesn't sync when I'm back online and I lose sales data." [Reddit r/Shopify, Sep 2024]. Shopify hasn't fixed this because their revenue model prioritizes web subscriptions over mobile app quality -- the app is a free add-on, not a profit center.

Integration Reliability Crisis: "Integration failures with QuickBooks are constant; app syncs 60% of transactions correctly on mobile, rest duplicate or vanish." [Capterra Square Review, Oct 2024]. Square's mobile app is secondary to their hardware POS focus, so integration reliability suffers on mobile compared to their core terminal business.

Verification Gap in Mobile Experience: "Klaviyo mobile app is a joke for push notifications--takes 30+ seconds to load campaigns, and half the time the preview doesn't match what sends." [Reddit r/ecommerce, Mar 2025]. Marketing platforms like Klaviyo built desktop-first and treat mobile as an afterthought because their buyer (marketing manager) uses desktop, not the end consumer's mobile experience.

Why Incumbents Haven't Fixed This:

- Wrong revenue incentive: Platform fees come from transaction volume, not trust quality
- Technical debt: Desktop-era architecture doesn't scale to real-time mobile verification
- Wrong user focus: They optimize for platform metrics, not cross-platform buyer confidence

Risk Assessment

Risk	Likelihood	Impact	Mitigation Strategy
Platform exclusion	High	Critical	Focus on cross-platform value; avoid competing directly with platform-native features
User adoption friction	Medium	High	Freemium model with immediate value demonstration

Verification accuracy	Medium	Critical	Start with simple fraud vectors (pricing, seller history) before complex AI
Competitor response	High	Medium	Move fast on mobile-first UX while incumbents are desktop-focused
Regulatory compliance	Medium	High	Build privacy-first architecture from day one

KEY INSIGHT
The biggest risk is building another broken verification system that promises more than it delivers. Users are already burned by tools that don't work reliably.

Recommendation & Next Steps

Recommendation: PROCEED WITH CAUTION -- Build the Minimum Viable Verification

The market signal (88/100 validation score) justifies moving forward, but the competitive graveyard (1.3/5 average competitor rating) demands extreme focus. Don't build broad "integrity assurance" -- that's how you become another 1.3-star disappointment.

If GO, Top 3 Immediate Actions:

1. Validate one specific fraud vector -- Survey 50 sellers: fake pricing vs. counterfeit products vs. seller impersonation. Pick the one with highest pain/frequency ratio.
2. Build browser extension MVP first -- Test verification logic on existing e-commerce sites before committing to full mobile app development. 4-6 week timeline vs. 4-6 months.
3. Partner with one mid-size platform -- Approach platforms like Etsy or smaller Shopify alternatives. Easier entry than competing with Amazon/eBay directly.

Key Assumptions Needing Validation:

- Sellers will integrate third-party verification vs. waiting for platform-native solutions
- Buyers will trust independent verification vs. relying on platform guarantees
- Verification can be accurate enough to reduce false positives below 5%

Timeline Reality Check: This is a 2-3 year build to meaningful market penetration, not a 6-month sprint. Budget $200K+ for verification infrastructure that actually works at scale.

Decision Checklist

1. Survey 25 e-commerce sellers in the next 14 days -- validate which specific fraud type causes the most lost sales
2. Build competitor feature matrix -- document exact gaps in Trustpilot/PayPal Protection mobile experiences
3. Test browser extension MVP -- validate verification logic on 3-5 major e-commerce sites within 30 days
4. Interview 10 burned buyers -- understand their current verification workflow and pain points

5. Draft partnership outreach to 5 mid-tier e-commerce platforms -- test receptivity before building

6. Calculate verification accuracy requirements -- research acceptable false positive rates for trust systems

7. Prototype one verification flow -- fake pricing detection or seller verification, not everything at once

Sources & References

1. [1] Competitor data and validation scores from Unbuilt Lab platform analysis

2. [2] Qubit Capital, "TAM/SAM/SOM: Market-Sizing Your Retail Startup" (2025 retail SOM example: 0.5-2%). URL: https://qubit.capital/blog/tam-sam-som-market-sizing-retail-opportunity

3. [3] Adapty, "TAM, SAM, SOM: How to Measure Market Size for Your App" (bottom-up formula: customers × ARPU). URL: https://adapty.io/blog/tam-sam-som/

4. [4] Grand View Research, "E-commerce Market Size And Share | Industry Report, 2033" (2025 data point: USD 33.91 trillion). URL: https://www.grandviewresearch.com/industry-analysis/e-commerce-market

5. [5] Antler, "TAM, SAM & SOM: How To Calculate The Size Of Your Market" (SOM as realistic slice). URL: https://www.antler.co/blog/tam-sam-som

6. [6] Reddit community pain signals from r/Shopify, r/ecommerce, and Capterra reviews (2024-2025)

7. [7] iOS App Tracking Transparency impact data from research on Meta revenue losses and CAC increases

8. [8] Evidence snapshot quotes from Google Play Store reviews provided in idea context

9. [9] Technavio, "Retail E-commerce Software Market" (2024 growth projections). URL: https://www.technavio.com/report/retail-e-commerce-software-market-market-industry-analysis

10. [10] Mordor Intelligence, "E-commerce App Market" (2025 market size data). URL: https://www.mordorintelligence.com/industry-reports/e-commerce-app-market

02 Opportunity Brief

EXECUTIVE SUMMARY

TrustSeal targets a real pain point -- e-commerce fraud hits 77% of the validation data points with users explicitly complaining about "scam promotions" and "deceitful users post false prices" [provided evidence]. The $20.35T mobile e-commerce market provides massive surface area for a trust solution. However, 236 direct competitors already exist with an average 1.3/5 rating, indicating this problem is brutally difficult to solve operationally. The window for a mobile-first approach exists, but expect a 2-3 year grind to meaningful traction.

Decision Recommendation: BUILD -- but pivot to price verification MVP first, not broad "integrity assurance." The user pain is validated, but the solution needs surgical focus.

Why This Could Actually Win (The Founder's Edge)

1. Why incumbents are vulnerable here. Current players like Guardsquare ($5,000/month enterprise focus) and Arkose Labs ($10,000/month minimums) price out 95% of small-to-medium sellers who need trust solutions most [1][2]. User complaints show "deceitful users post false prices makes this useless" -- existing platforms like Trustpilot focus on post-purchase reviews, not real-time pricing scam prevention [provided evidence]. The incumbents built for enterprise fraud teams, not individual sellers getting burned by fake competitors.

2. What moat could plausibly emerge. A seller-verified pricing database becomes more valuable with each merchant who joins -- creating a network effect where buyers recognize TrustSeal badges as legitimate pricing signals. After 1,000+ verified sellers, the mobile app becomes the de facto "price check this deal" tool, similar to how Honey became the coupon extension. Weak defensibility against well-funded copycats, but first-mover advantage in mobile-native seller verification could last 12-18 months.

3. The fastest unfair distribution edge. Reddit's seller communities -- r/ecommerce (180K members), r/Entrepreneur (1.2M), r/dropship (85K) -- where trust issues dominate daily conversations. Founders can provide genuine value answering "how do I compete against fake pricing?" questions while occasionally mentioning their solution. This beats $50+ CACs on Facebook/Google ads targeting the same frustrated sellers.

4. Why users would switch. "Don't fall victim to their scam promotions or $1 coupons as they cannot be used with any other coupons" [provided evidence]. Current solutions are reactive (Trustpilot reviews after purchase) or platform-specific (eBay Buyer Protection). TrustSeal would prevent the scam before checkout with real-time price verification and cross-platform seller scoring -- addressing the "check before I buy" behavior that existing tools miss.

Risk Register

Rank	Risk	Severity (1-5)	Likelihood (1-5)	Mitigation
1	Platform policy changes block URL scanning	5	4	Build direct seller integration tools, not scraping
2	Existing 236 competitors include well-funded players	4	5	Focus on mobile-native UX gap vs enterprise tools
3	Sellers won't pay $10-50/month for verification badges	4	3	Start freemium, charge only after proven ROI
4	Price data APIs (Google Shopping) are expensive/rate-limited	3	4	Partner with existing price comparison sites
5	False positive rate destroys user trust in week 1	5	3	Manual review queue for 500+ items before automation
6	iOS/Android app store rejection for "scam detection"	4	2	Position as "price comparison" tool, not fraud prevention
7	Legal liability for incorrect seller ratings	3	2	Terms of service disclaimer + seller appeal process

Opportunity Register

Price Verification First: Users complain about "$1 coupons" and "false prices" -- build the mobile price scanner before seller verification. Simpler MVP, clearer value prop, easier App Store approval.

Seller Badge Network: Once price verification works, sellers will pay for "TrustSeal Verified Pricing" badges to differentiate from scam competitors. Network effects kick in as buyers recognize the symbol.

Cross-Platform Intelligence: Aggregate pricing data across Amazon, eBay, Shopify stores to flag outliers. No existing mobile app does comprehensive cross-platform price monitoring.

B2B2C Partnership: License the verification engine to platforms (Shopify, WooCommerce) as a native trust feature. Recurring enterprise revenue vs consumer app monetization struggles.

Mobile-Native Advantage: All major competitors built desktop-first. Mobile e-commerce is 60% of transactions but trust tools lag behind -- UX opportunity for native mobile verification [10].

Comparable Companies

Winners:

- Sift -- $100K+/year enterprise fraud detection, usage-based $0.10-$0.50 per $1,000 GMV, founded 2011 [6]
- Whatnot -- $11.5B valuation (2025), live commerce platform with built-in seller verification, founded 2018 [1]

- Trustpilot -- Public company, $100M+ revenue from review platform subscriptions, founded 2007
- Arkose Labs -- $10K+/month enterprise starting price, pay-per-challenge model for bot detection, founded 2013 [4]
- Human Security (PerimeterX) -- $25K+ annual contracts, $0.02-$0.05 per 1,000 bot challenges, founded 2014 [5]

Failed startups: Research thin on documented failures -- the Perplexity search focused on successful companies only [1][2][3]. Most startup failure databases (Crunchbase, Failory) were not included in the research. Founder must investigate 3-5 failed trust/verification startups from 2018-2023 to understand common failure modes before proceeding.

Market Sizing (TAM / SAM / SOM)

- TAM: $33.91T -- Global e-commerce market in 2025 [10]
- SAM: $20.35T -- Mobile e-commerce subset (60% of total), where mobile apps can realistically intercept transactions [10]
- SOM: $204M -- Conservative 0.1% capture of mobile commerce trust spending. Methodology: If sellers spend 0.1% of GMV on trust/verification tools (industry benchmark), and TrustSeal captures 1% of that market = $20.35T × 0.001 × 0.01

5 Validation Experiments

#	Experiment	Cost	Time	Success Criteria
1	Landing page + Reddit posts in r/ecommerce measuring signups	$50	7 days	>3% CTR from Reddit traffic
2	Manual price checking service for 20 sellers (Upwork VA)	$200	14 days	>40% find actionable pricing insights
3	Cold email 100 Shopify store owners about trust issues	$0	10 days	>5% response rate mentioning pricing scams
4	Facebook/Instagram ads to "scammed by fake deals" audience	$300	7 days	<$3 cost per email signup
5	Interview 10 users who left 1-star reviews for shopping apps	$100	14 days	>30% mention pricing/trust as core issue

Distribution Channel Ranking

1. Reddit seller communities -- r/ecommerce has 180K frustrated sellers discussing trust daily. First test: provide helpful answers in 5 threads, measure DM responses.
2. SEO for "fake price check" -- 2,900 monthly searches, weak competition from comparison sites. First test: publish "How to Spot Fake E-commerce Prices" content.

3. Cold email to Shopify stores -- 1.7M active stores, email addresses scrapable. First test: 100 emails to stores with <4.5-star ratings asking about trust issues.

Cost-to-Validate vs Cost-to-Build

Validation Cost: $650 + 6 weeks (sum of 5 experiments above + landing page setup)

Build Cost: $75-125K for MVP mobile app with price comparison API integration, seller verification backend, basic fraud scoring. Timeline: 4-6 months to App Store launch.

30 / 60 / 90-Day Validation Roadmap

Day 30 Goals:

- Launch landing page with email capture
- Complete 5 validation experiments above
- Interview 10 sellers from Reddit communities
- Decision gate: If <100 email signups OR <3 seller interviews mention pricing scams as top-3 pain, pivot focus

Day 60 Goals:

- Build price comparison MVP (web app, not mobile yet)
- Test with 20-30 sellers from validation phase
- Measure false positive rate on 500+ price checks
- Decision gate: If >20% false positive rate OR <50% user retention after 1 week, rebuild algorithm

Day 90 Goals:

- Native mobile app beta (iOS + Android)
- First 10 paying sellers at $10-20/month for verification badges
- App Store/Play Store submission
- Decision gate: If <5 paying customers OR app store rejection, reassess positioning

Kill Criteria

- If 200 cold emails to sellers generate <10 responses mentioning pricing/trust issues
- If price comparison false positive rate stays >15% after algorithm improvements
- If comparable failed startups (once researched) failed at similar revenue levels with better funding
- If Apple/Google reject the mobile app twice for policy violations
- If CAC exceeds $50 for sellers and $15 for consumers after testing 3+ channels
- If first 100 users show <20% weekly retention by day 30

Cold-Outreach Script (drop-in)

Email Subject: Quick question about fake pricing competitors

Hi [Name],

I noticed [Company] sells [ProductCategory] and wondered if you've dealt with fake competitors undercutting your prices with "$1 coupon" scams?

I'm building a mobile app to help legitimate sellers verify their pricing is trustworthy (and flag the fake deals). Would you be interested in a 10-minute call to share what trust issues cost you?

Not selling anything -- just validating if this problem is as widespread as I think.

Best, [YourName]

LinkedIn DM: Hey [Name] - saw your Shopify store [Company] and curious: do fake pricing competitors impact your sales? Building a solution for this specific problem. 5-min call to hear your experience? Thx!

Landing-Page Copy (drop-in)

Hero H1: Stop Losing Sales to Fake Pricing Scams

3-bullet sub-pitch:

- Verify your prices are trustworthy with mobile verification badges
- Flag competitors posting fake "$1 coupon" deals that can't be redeemed
- Join 500+ sellers protecting buyers from pricing deception

CTA: Join the Waitlist -- Beat the Scammers

What you get:

- Price verification badge for all your listings
- Real-time scam competitor alerts
- Mobile app for instant deal verification

Sources & References

1. [1] https://www.guardsquare.com
2. [2] https://www.arkoselabs.com
3. [3] https://sift.com
4. [4] https://www.humansecurity.com
5. [5] https://www.landbase.com/blog/fastest-growing-ecommerce-tech-companies
6. [6] https://info.merchantriskcouncil.org/hubfs/Documents/Reports/Fraud%20Reports/2025_Global_Fraud_and_Payments_Report.pdf
7. [7] https://www.pewresearch.org/short-reads/2025/11/19/about-a-third-of-americans-say-theyve-had-an-online-shopping-scam-happen-to-them/

8. [8] https://www.kaspersky.com/small-to-medium-business-security
9. [9] https://www.geetest.com
10. [10] https://www.adjust.com/blog/shopping-app-trends/

03 Product Requirements Document (PRD)

EXECUTIVE SUMMARY

In short: TrustSeal is a mobile app that helps e-commerce buyers verify seller legitimacy and avoid scams through real-time price verification and seller scoring -- starting with the single biggest pain point: fake pricing.

- Product: TrustSeal Mobile Verification App
- Opportunity: $20.35T mobile e-commerce market plagued by pricing scams
- MVP Focus: Price verification for suspicious deals
- Target Timeline: 4-week MVP, 3-month market test
- Expected Build Cost: $75-125K for verification engine
- Key Risk: Competing with platform-native tools and 236 existing competitors

Product Overview

Product Name: **TrustSeal Vision Statement:** Eliminate e-commerce scams before checkout through automated seller and pricing verification. Product Type: **Native iOS/Android mobile app with backend API Core Value Proposition:** Scan any product URL or barcode to instantly verify if the price is legitimate and the seller is trustworthy, preventing buyers from falling victim to "scam promotions" and "false prices" [1].

Build This FIRST -- the sharpest version of the MVP

The one-feature MVP: A mobile app that scans product URLs and instantly shows "Price Alert: XX% below market average -- investigate seller" or "Price Verified: Within normal range."

Why this specific feature: The validation data shows users explicitly complaining about "scam promotions or $1 coupons" and "deceitful users post false prices" [provided evidence]. This single feature directly addresses the #1 complaint pattern without requiring complex seller onboarding or verification infrastructure. Users can immediately test suspicious deals they encounter on any platform.

The cut list -- what NOT to build in v1:

- Seller verification badges -- requires building relationships with platforms you don't control
- In-app purchasing -- adds payment processing complexity and platform policy headaches
- Social features/reviews -- Trustpilot already owns this space with superior network effects
- Multi-platform seller dashboards -- focus on buyer-side first, seller tools are a different product

- Real-time chat support -- adds staffing costs before you know if anyone will use the core feature
- Advanced fraud detection ML -- start with simple price comparison APIs before building algorithms
- Push notifications -- iOS/Android permission friction kills adoption in week 1

Build order for the v1:

1. Week 1: Static landing page + email capture with "Scan suspicious deals" value prop
2. Week 2: Basic mobile app shell with URL input field and mock "checking price..." flow
3. Week 3: Connect to 2-3 price comparison APIs (Google Shopping, PriceGrabber) for basic price checking
4. Week 4: Add simple red/yellow/green price alerts based on deviation from average market price
5. Week 5: Beta test with 20-50 users from email list, focusing on false positive rate
6. Week 6: Add barcode scanning for in-store price checking (extends use case)
7. Week 7: App Store/Play Store submission with core price verification working

Decision rule for what to add next: If 30%+ of scanned items generate user follow-through actions (bookmarking, sharing, avoiding purchase) in week 8, add seller reputation scores from public data. If follow-through is <30%, the price verification alone isn't compelling enough -- kill or pivot to different fraud vector.

User Personas & Stories

Primary Persona: Cautious Online Shopper "Sarah"

- Age: 35-55, household income $50-100K
- Shops online 2-3x per month, average order $75-150
- Has been burned by fake deals or counterfeit products
- Over-researches purchases, checks multiple sites before buying
- Uses mobile for 70% of shopping research

Secondary Persona: Deal Hunter "Marcus"

- Age: 25-40, price-sensitive, shops deals/flash sales
- Active on deal forums, social commerce groups
- Willing to take calculated risks on unknown sellers for big savings
- Needs quick verification tool for time-sensitive deals

Tertiary Persona: Small Business Owner "Jennifer"

- Purchases inventory/supplies online for business
- Cannot afford to lose money on fraudulent suppliers
- Values verification tools that prevent vendor fraud

User Stories:

1. As Sarah, I want to paste a product URL and instantly see if the price is suspicious, so I don't waste time researching obvious scams

2. As Sarah, I want to scan a barcode in-store and see if online prices are significantly lower, so I make informed buying decisions

3. As Marcus, I want to quickly verify flash sale deals from unknown sellers, so I can act fast on legitimate opportunities

4. As Sarah, I want to see if other buyers have reported problems with a specific seller, so I avoid bad experiences

5. As Jennifer, I want to verify business supplier legitimacy before placing large orders, so I protect my company from fraud

6. As Marcus, I want to save verified good deals to a wishlist, so I can purchase later with confidence

7. As Sarah, I want to get alerts when suspicious prices drop to normal ranges, so I know when it's safe to buy

8. As any user, I want the app to work without creating an account, so I can test it immediately

9. As Sarah, I want to share verified deals with friends, so they benefit from my research

10. As Marcus, I want to report new scams I discover, so the community stays protected

11. As Jennifer, I want to bulk-check multiple supplier URLs, so I can vet vendors efficiently

12. As any user, I want the app to work offline with cached data, so I can verify in stores without good cell coverage

13. As Sarah, I want browser integration to check prices without switching apps, so verification fits my normal shopping flow

14. As Marcus, I want to see price history trends, so I know if current deals are actually good

15. As Jennifer, I want to verify international suppliers, so I can safely expand my sourcing

Acceptance Criteria for Top 5 User Stories:

Story 1 (URL Price Verification):

- User pastes any e-commerce URL into input field
- App extracts product details and current price within 3 seconds
- App displays price alert status: Safe (green), Investigate (yellow), Suspicious (red)
- App shows price comparison data from at least 2 other sources

Story 2 (Barcode Scanning):

- Camera opens when user taps "Scan Barcode" button
- App recognizes standard UPC/EAN barcodes within 2 seconds
- App displays product name and price comparison across 3+ online retailers
- App works in typical store lighting conditions

Feature Specification

MVP Features (Must-Have for Launch)

Price Verification Engine

- Description: Core algorithm that compares submitted prices against market averages from multiple sources. Returns simple risk assessment.
- User story: Story 1 (URL price verification)
- Priority: P0 (launch blocker)
- Complexity: High

URL Input & Product Extraction

- Description: Input field that accepts product URLs from major e-commerce sites, extracts product details and price automatically.
- User story: Story 1 (URL price verification)
- Priority: P0 (launch blocker)
- Complexity: Medium

Mobile-First UI Shell

- Description: Clean, fast native mobile interface optimized for quick price checking workflow.
- User story: Story 8 (works without account)
- Priority: P0 (launch blocker)
- Complexity: Low

Basic Price Alert Display

- Description: Simple red/yellow/green status with brief explanation of price deviation from market average.
- User story: Story 1 (URL price verification)
- Priority: P0 (launch blocker)
- Complexity: Low

Barcode Scanner Integration

- Description: Camera-based barcode scanning that connects to price verification engine for in-store use.
- User story: Story 2 (barcode scanning)
- Priority: P1 (launch with)
- Complexity: Medium

Phase 2 Features (Post-Launch)

Seller Reputation Scores (Month 2)

- Pull public seller ratings from major platforms
- Display aggregated seller trustworthiness score

Deal Wishlist (Month 2)

- Save verified good deals for later purchase

- Price drop notifications for wishlisted items

Browser Extension (Month 3)

- Chrome/Safari extension for seamless URL checking
- Right-click verification on any product page

Future Roadmap Features

Community Reporting (Month 4+)

- User-generated scam reports and seller feedback
- Crowdsourced verification data

Business/Bulk Tools (Month 6+)

- CSV upload for bulk supplier verification
- API access for business customers

Information Architecture

```
TrustSeal App
+-- Home/Scan Screen
|   +-- URL Input Field
|   +-- Barcode Scan Button
|   +-- Recent Scans History
|   +-- Quick Help/Examples
+-- Verification Results Screen
|   +-- Price Alert Status (Red/Yellow/Green)
|   +-- Price Comparison Table
|   +-- Seller Info (if available)
|   +-- Share Button
|   +-- Report Issue Button
+-- Scan History Screen
|   +-- Previous Verifications List
|   +-- Saved/Bookmarked Items
|   +-- Clear History Option
+-- Settings/Help Screen
    +-- About TrustSeal
    +-- How It Works
    +-- Contact/Feedback
    +-- Privacy Policy
```

Key Screen Descriptions:

Home/Scan Screen: Single-purpose interface with prominent URL input field and barcode scan button. Recent scans list shows last 5 verifications with quick status indicators.

Verification Results Screen: Displays price alert status prominently at top, followed by comparison data table showing prices from 3-4 other sources. Clean, scannable layout optimized for quick decision-making.

Functional Requirements

Price Verification Engine:

- Input: Product URL or barcode scan
- Processing: Extract product details, query 3-5 price comparison APIs, calculate deviation from average
- Output: Risk assessment (Safe/Investigate/Suspicious) with supporting price data
- Business Rules: Flag items >40% below market average as "Suspicious", 20-40% below as "Investigate"
- Edge Cases: Handle out-of-stock items, discontinued products, region-specific pricing

URL Processing:

- Input: Product URLs from Amazon, eBay, Walmart, Target, etc.
- Processing: Parse URL structure, extract product ID, scrape product details
- Output: Standardized product information (title, price, seller, images)
- Edge Cases: Handle URL shorteners, mobile vs desktop URLs, international domains

Barcode Scanning:

- Input: Camera feed with barcode in frame
- Processing: OCR barcode recognition, UPC/EAN lookup via product database
- Output: Product identification for price verification engine
- Edge Cases: Poor lighting, damaged barcodes, non-standard barcode formats

Non-Functional Requirements

Performance Targets:

- Price verification results in <5 seconds for URLs, <3 seconds for barcodes
- App launch time <2 seconds on mid-range devices
- 99.5% uptime for verification API

Scalability:

- Support 1,000 concurrent price checks initially
- Scale to 10,000+ concurrent users by Month 6
- Database designed for 1M+ product verifications

Security:

- No user account required for basic features
- HTTPS for all API communications
- No sensitive user data collection beyond usage analytics

Accessibility:

- WCAG 2.1 AA compliance for core features

- VoiceOver/TalkBack support for visually impaired users
- High contrast mode for barcode scanning

Offline Capability:

- Cache last 20 verification results for offline viewing
- Offline barcode scanning with cloud sync when reconnected

Data Requirements

Core Data Entities:

- Product: title, UPC/EAN, category, brand
- Price Point: amount, currency, source, timestamp, seller
- Verification: product_id, user_id (optional), risk_level, created_at
- Source: name, reliability_score, API_endpoint

User Data Collection:

- Anonymous usage analytics (screens viewed, features used)
- Optional email for price drop alerts
- No personal information required for core features

Third-Party Integrations:

- Google Shopping API for price comparisons
- Amazon Product Advertising API
- UPC Database API for barcode lookups
- PriceGrabber API for additional price sources

Data Retention:

- Price data cached for 24 hours to reduce API costs
- User verification history stored for 30 days
- Anonymous analytics retained for 1 year

Integration Requirements

Essential Third-Party Services:

- Google Shopping API: Primary source for price comparisons and product data
- Stripe: Payment processing for future premium features
- Firebase: Backend services, analytics, and crash reporting
- Twilio SendGrid: Email notifications for price alerts

API Integrations:

- RESTful APIs for all price comparison services
- Rate limiting: 100 requests/minute per user

- Fallback logic when primary APIs are unavailable

Webhook Requirements:

- Price change notifications for wishlisted items
- System health monitoring alerts
- Failed verification attempt logging

Success Metrics

Primary KPIs:

- Daily Active Users (DAU): Target 1,000+ by Month 3
- Price Verification Accuracy: >90% match with actual market prices
- User Retention: 30% of users return within 7 days of first use
- Time to Verification: Average <4 seconds from URL paste to results
- False Positive Rate: <10% of "Safe" items later reported as problematic

Secondary KPIs:

- App Store ratings: Maintain >4.0 stars
- Verification success rate: >95% successful verifications (no errors)
- API cost per verification: <$0.05
- User-reported scam discoveries: Track community value generation

North Star Metric: Weekly verification volume -- measures both user adoption and product utility

Measurement Tools:

- Firebase Analytics for user behavior tracking
- Custom dashboard for API performance monitoring
- App Store Connect for ratings/reviews analysis
- Mixpanel for funnel analysis and retention cohorts

KEY INSIGHT
Success hinges on verification accuracy and speed -- users will abandon immediately if the tool gives wrong information or takes too long. Focus measurement on these operational metrics over vanity metrics like downloads.

Sources & References

1. [1] Market Validation & Opportunity Report -- user complaint data and competitive analysis from Unbuilt Lab platform research

04 Technical Architecture Blueprint

Architecture Overview

In short: Serverless-first mobile app with edge caching and third-party price APIs -- built for rapid scaling without infrastructure management overhead.

Architecture Pattern: Serverless + Mobile-First The architecture centers on a React Native mobile app consuming serverless functions for price verification logic. This hybrid approach balances development speed (crucial for MVP) with operational simplicity.

Architecture Diagram Description:

```
Mobile App (React Native)
    v HTTPS
Edge Cache (Cloudflare)
    v Cache miss
API Gateway (AWS)
    v Route by endpoint
Lambda Functions (Node.js)
    +--> Price Verification Service
    +--> Barcode Lookup Service
    +--> Product Data Service
    v Read/Write
RDS (PostgreSQL)
    v External APIs
[Google Shopping, Amazon API, UPC Database]
```

Key Architectural Decisions:

- Serverless over containers -- eliminates infrastructure management for medium budget
- PostgreSQL over NoSQL -- price comparison requires relational joins for product matching
- Edge caching -- price data has natural 4-hour TTL, perfect for CDN caching
- Native mobile over PWA -- barcode scanning requires camera API access

Technology Stack Recommendation

Layer	Technology	Rationale
Frontend	React Native 0.72+	Cross-platform with native camera access for barcode scanning. Expo managed workflow for faster deployment.
Backend	Node.js 18+ on AWS Lambda	Serverless scales with usage spikes. JavaScript across stack reduces context switching.
Database	PostgreSQL 15 on AWS RDS	ACID compliance for financial data. Superior full-text search for product matching.

Cache	Redis on AWS ElastiCache	Session storage for price verification results. 4-hour TTL aligns with price data freshness.
Hosting	AWS (Lambda + RDS)	Integrated ecosystem. RDS backup/scaling built-in. Lambda cold starts acceptable for price checking use case.
CI/CD	GitHub Actions + EAS Build	Free tier sufficient. EAS handles React Native builds. Direct AWS deployment via GitHub Actions.

Version Lock Recommendations:

- React Native: 0.72.4 (stable release with New Architecture opt-in)
- Node.js: 18.17.0 LTS (Lambda runtime support)
- PostgreSQL: 15.3 (latest minor for RDS)

Database Design

Entity Relationship Overview: Core entities revolve around Products, Prices, and Verifications. Products have many Prices from different Sources. Each Verification links a Product to a risk assessment result.

Key Tables:

Table	Columns	Types	Constraints
products	id, title, upc, brand, category, created_at	UUID, TEXT, VARCHAR(20), VARCHAR(100), VARCHAR(50), TIMESTAMP	PK(id), UNIQUE(upc)
sources	id, name, api_endpoint, reliability_score, active	UUID, VARCHAR(50), TEXT, DECIMAL(2,1), BOOLEAN	PK(id), CHECK(reliability_score 1-5)
prices	id, product_id, source_id, amount, currency, seller_name, scraped_at	UUID, UUID, UUID, DECIMAL(10,2), CHAR(3), VARCHAR(100), TIMESTAMP	PK(id), FK(product_id, source_id)
verifications	id, product_id, user_session, risk_level, price_deviation, created_at	UUID, UUID, VARCHAR(50), VARCHAR(10), DECIMAL(5,2), TIMESTAMP	PK(id), FK(product_id)

Indexing Strategy:

- `products.upc` -- B-tree index for barcode lookups (primary use case)
- `prices.product_id, scraped_at` -- composite index for latest price queries
- `verifications.created_at` -- for analytics and cleanup jobs
- Full-text search index on `products.title` for fuzzy product matching

Data Migration Considerations: Price data becomes stale quickly -- implement sliding window deletion (keep 30 days max). Product catalog grows slowly but never shrinks, design for 1M+ products by year 2.

API Design

REST over GraphQL -- price verification workflows are simple request/response patterns. GraphQL adds complexity without benefit for this use case.

Key API Endpoints:

Method	Endpoint	Description	Auth Required
POST	/api/v1/verify/url	Submit product URL for price verification	No
POST	/api/v1/verify/barcode	Submit UPC/EAN for price verification	No
GET	/api/v1/products/{id}/prices	Get current price data for product	No
POST	/api/v1/alerts	Create price drop alert	Yes (email)
GET	/api/v1/alerts	List user's active alerts	Yes (email)
DELETE	/api/v1/alerts/{id}	Remove price alert	Yes (email)

Authentication Flow:

- Anonymous access for core verification features (80% of use cases)
- Email-based auth for price alerts only -- simple magic link flow
- JWT tokens stored in React Native SecureStore, 30-day expiry

Rate Limiting:

- 100 verifications/hour per IP address
- 1,000 verifications/hour per authenticated user
- Implement sliding window with Redis counters

API Versioning: Header-based versioning (`API-Version: 1.0`) -- simpler than URL versioning for mobile clients that can't easily update endpoints.

Security Architecture

Authentication Method: Magic link email authentication for optional features. No passwords to manage, reduces attack surface. Anonymous usage for core verification features.

Authorization Model: Simple role-based access:

- Anonymous: price verification only
- Verified User: price alerts, verification history
- Admin: system monitoring, source management

Data Encryption:

- At rest: RDS encryption enabled with AWS KMS
- In transit: TLS 1.3 for all API communications

- Mobile storage: React Native SecureStore for JWT tokens

Input Validation:

- URL validation against whitelist of supported e-commerce domains
- UPC/EAN format validation (8, 12, 13 digit patterns)
- Price amount bounds checking (prevent negative prices, unrealistic values)
- SQL injection protection via parameterized queries

OWASP Top 10 Considerations:

- A01 Broken Access Control: JWT validation on protected endpoints
- A03 Injection: Parameterized SQL queries, URL validation whitelist
- A07 Identification/Auth Failures: Magic link expiry (15 minutes), rate limit auth attempts
- A09 Security Logging: CloudWatch logging for failed auth, suspicious patterns

Infrastructure & Deployment

Hosting Recommendation: AWS with this specific setup:

- Lambda: 512MB memory, 30-second timeout (sufficient for API calls)
- RDS: db.t3.micro initially (2 vCPU, 1GB RAM) -- scales to db.t3.small when >100 concurrent users
- ElastiCache: cache.t3.micro Redis (1 vCPU, 0.5GB)

Environment Setup:

- Development: Local React Native + ngrok tunnel to localhost Lambda via SAM CLI
- Staging: AWS with separate RDS instance, same Lambda functions
- Production: AWS with Multi-AZ RDS, CloudFront CDN

CI/CD Pipeline:

```
GitHub Push -> GitHub Actions ->
  +- Run tests (Jest + Detox)
  +- Build React Native (EAS Build)
  +- Deploy Lambda (AWS SAM)
```

Monitoring Setup:

- Application: AWS CloudWatch for Lambda metrics, RDS performance insights
- Business: Custom dashboard tracking verification success rates, API response times
- Alerting: SNS notifications for Lambda errors >5/minute, RDS CPU >80%

Monthly Infrastructure Cost Breakdown:

Service	Usage	Cost
Lambda	100K invocations/month	$20
RDS t3.micro	24/7 runtime	$16
ElastiCache t3.micro	24/7 runtime	$11

Data transfer	50GB/month	$5
CloudWatch	Standard logging	$8
Total	**MVP traffic**	**$60/month**

Scale-up at 10K users: ~$180/month (RDS t3.small upgrade, increased Lambda usage).

Scalability Plan

Current Architecture Handles: 1,000 concurrent price verifications with <5 second response time

Scaling Triggers:

- Lambda: Auto-scales to 1,000 concurrent executions (AWS default)
- Database: Scale vertically when CPU >70% sustained (RDS Performance Insights alert)
- Cache: Scale when memory utilization >75%

Database Scaling Approach:

1. Month 1-3: Single RDS instance with connection pooling
2. Month 4-12: Read replicas for price lookup queries (90% of database load)
3. Year 2+: Partition prices table by date ranges

Caching Strategy:

- Price data: 4-hour TTL (aligns with price freshness expectations)
- Product metadata: 24-hour TTL (rarely changes)
- Verification results: 1-hour TTL (users re-check same products)
- Cache warming: Scheduled Lambda pre-loads popular products

CDN Strategy: CloudFront with origin at API Gateway. Cache price verification responses by product ID for 1 hour -- reduces Lambda costs by ~60% for popular products.

Third-Party Services

Service	Provider	Purpose	Cost Tier	Alternative
Price Data	Google Shopping API	Primary price comparison source	$5/1K requests	PriceGrabber API
Barcode Lookup	UPC Database	Convert UPC to product details	Free tier 100/day	Barcode Lookup API
Email	SendGrid	Magic link authentication	Free tier 100/day	AWS SES
Analytics	Mixpanel	User behavior tracking	Free tier 1K users	PostHog
Crash Reporting	Sentry	React Native error tracking	Free tier 5K errors/month	Bugsnag
Push Notifications	Expo Push	Price drop alerts	Free tier 1M/month	Firebase FCM

Key Integration Risks:

- Google Shopping API rate limits (1K requests/day on free tier) -- implement caching aggressively
- UPC Database free tier expires quickly with real usage -- budget $50/month for paid tier by month 2

Development Environment Setup

Required Tools:

```
Node.js 18.17.0
React Native CLI 11.3+
Expo CLI 6.3+
AWS SAM CLI 1.91+
PostgreSQL 15+ (local dev)
Redis 7+ (local dev)
```

Local Development Setup:

1. Clone repo, `npm install`
2. Start local PostgreSQL and Redis via Docker Compose
3. Run database migrations: `npm run migrate`
4. Start Lambda functions locally: `sam local start-api`
5. Start React Native: `npx expo start`

Environment Variables:

- Use `.env.local` for development secrets
- AWS Parameter Store for production secrets
- Never commit API keys to version control

Code Quality Tools:

- ESLint + Prettier: Standard React Native config
- Husky: Pre-commit hooks for linting and testing
- Jest: Unit testing for Lambda functions
- Detox: E2E testing for critical user flows (URL verification, barcode scan)

Implementation Traps -- non-obvious things that bite founders 30-60 days in

The trap: Over-architecting the price comparison algorithm When it bites: **Week 4, when you realize Google Shopping API returns wildly different price formats and you've built a complex normalization engine that breaks on edge cases. The cheaper avoid:** Start with simple median price calculation. Store raw price strings from APIs and normalize display-side only. The sophisticated outlier detection can wait until month 3 when you have real user feedback on false positives.

The trap: Building barcode scanning from scratch When it bites: **Week 6, when iOS camera permissions and Android autofocus differences consume your entire sprint and you still don't have reliable barcode detection. The cheaper avoid:** Use Expo's `expo-barcode-scanner` wrapper

around native libraries. It's "boring" but works across devices from day 1. Custom camera work is a month-long rabbit hole with zero user-facing benefit.

The trap: Storing full product catalogs locally When it bites: **Week 8, when your mobile app bundle size hits 50MB because you're syncing product databases for "offline functionality" that 5% of users actually need. The cheaper avoid:** Cache only the last 20 verified products locally. Everything else hits the API. Users tolerate 3-second loading for price verification -- they don't tolerate 50MB app downloads.

The trap: Real-time price change notifications When it bites: **Month 2, when your scheduled Lambda functions are hitting rate limits on price APIs and costing $200/month to check prices for 50 products every hour. The cheaper avoid:** Daily price checks only, and only for products with active alert subscriptions. Send digest emails, not individual notifications. Real-time price tracking is a $10K/month AWS bill waiting to happen.

The trap: Complex fraud detection scoring on MVP When it bites: **Week 10, when you've built machine learning models for scam detection but have zero training data and every prediction is random noise. The cheaper avoid:** Start with rule-based red flags: prices >50% below market average, sellers with <10 reviews, prices ending in .99 from unknown sellers. ML can wait until you have 10K+ verification data points.

The trap: Building admin dashboards before user dashboards When it bites: **Month 3, when you've spent two weeks building beautiful admin panels for monitoring price sources but users still can't save their verification history. The cheaper avoid:** AWS CloudWatch + simple SQL queries in a notebook are sufficient for MVP monitoring. Build user-facing features first; admin tools only when the manual process becomes painful (usually month 4+).

Sources & References

1. [1] Provided validation data and competitor analysis from Unbuilt Lab platform
2. [4] Global e-commerce market size estimates from provided context

05 Go-To-Market & Growth Strategy

EXECUTIVE SUMMARY

In short: TrustSeal needs a seller-first GTM that solves their immediate pain (lost conversions) before building the broader consumer verification platform.

Launch approach: Product-Led Growth with seller-side focus, leveraging e-commerce seller communities where trust issues are discussed daily. The mobile app launches as a "trust badge generator" that sellers embed on their listings, creating network effects as buyers recognize the verification symbol.

Timeline to first revenue: **8-12 weeks Target 100 paying customers:** Month 4-5 Expected CAC: $45-65 for sellers, $8-12 for consumers

GTM Strategy Overview

Launch Strategy: Hybrid Product-Led Growth + Community-Led Mobile-first approach targeting e-commerce sellers who need immediate trust signals, then expanding to consumer-side verification.

Positioning Statement: "The first mobile verification app that sellers control -- generate instant trust badges for your listings while protecting buyers from the scams you hate competing against."

Key Messaging Framework:

- Headline: "Stop Losing Sales to Trust Issues"
- Subhead: "Mobile verification that sellers control, buyers recognize, and platforms can't break"
- Supporting Points:
    1. Verify in 60 seconds, display immediately on any platform
    2. Join 10K+ sellers who've eliminated "is this seller legit?" questions
    3. Built by sellers who got burned by fake competitors, not another Silicon Valley trust experiment

The Founder's Unfair Distribution Edge

Channel Asymmetry: Reddit's Seller Frustration Communities

The sharpest distribution advantage is Reddit's e-commerce seller communities where trust issues dominate daily conversations. Specifically:

- r/ecommerce (180K members) -- daily posts about scam competitors and buyer hesitation
- r/Entrepreneur (1.2M members) -- trust issues in the weekly "What's killing your conversions?" threads

- r/dropship (85K members) -- constant complaints about buyers assuming all dropshippers are scammers
- r/AmazonSeller (95K members) -- review manipulation and fake seller complaints

Conversion Strategy: The founder becomes the "trust solutions expert" in these communities. Instead of pitching TrustSeal, they contribute genuine value by answering trust-related questions with specific tactics, occasionally mentioning "I built a mobile app to solve exactly this" when relevant. Target: 2-3 helpful posts per week across these communities.

Path to 100 paying customers: Active participation in these communities (providing value first) can generate 15-25 qualified seller leads per month. With a 15-20% conversion rate on engaged leads, that's 100 paying sellers by month 5-6.

Pre-Launch Phase (Weeks 1-4)

Landing Page Strategy:

- Hero section: Split-screen showing a sketchy listing vs. TrustSeal-verified listing with conversion rate difference
- Social proof: "Join the waitlist -- 2,847 sellers tired of losing to scammers"
- Mobile-first demo: Embedded video showing the 60-second seller verification process
- Copy direction: Problem-focused, not solution-focused -- "You're losing sales to trust issues. Here's why."

Beta User Recruitment Plan:

- Target: 50-75 active e-commerce sellers across platforms
- Sourcing: Reddit communities above, Facebook seller groups, Shopify Partners Slack
- Incentive: Free lifetime verification badge + first 100 get "Founding Seller" status in app
- Qualification: Must have active listings and previous trust-related conversion loss

Content Creation Plan:

- Week 1-2: "Trust Crisis Case Studies" -- 4 blog posts analyzing real seller stories from Reddit
- Week 3-4: Mobile verification demos on TikTok/Instagram Reels (15-30 second clips)
- Daily: LinkedIn posts sharing trust statistics with seller-focused insights

Community Building:

- Discord: Private beta community for founding sellers to report verification accuracy
- Reddit: Weekly "Trust Solutions Thursday" posts in relevant communities (value-first)
- LinkedIn: Connect with e-commerce sellers using "trust" or "scam" in recent posts

Launch Phase (Weeks 5-6)

Launch Platforms and Timeline:

Product Hunt Launch (Week 5, Tuesday 12:01 AM PST):

- Assets needed: Mobile app screenshots, demo video, maker story focusing on seller pain

- Hunter strategy: Recruit 3-4 active sellers from beta to be first commenters
- Goal: Top 10 in Mobile Apps category

Hacker News Show HN (Week 5, Thursday):

- Title: "Show HN: Mobile app that generates trust badges sellers control (not platforms)"
- Strategy: Technical founder story -- built after losing sales to fake competitor pricing
- Comments: Pre-seed 5-7 thoughtful responses to likely questions about verification methods

Reddit Community Posts (Week 6):

- r/ecommerce: "I built a mobile verification app after losing $15K to trust issues. Here's what I learned."
- r/Entrepreneur: "The trust tax is killing small sellers. Here's a mobile solution."
- r/dropship: "Stop losing conversions to scammer assumptions"

Social Media Sequence:

- Day 1: LinkedIn founder story with conversion loss statistics
- Day 2: Instagram/TikTok: 30-second verification demo
- Day 3: Twitter thread: "Why trust badges fail + how mobile changes the game"

Press/Media Outreach:

- TechCrunch: Mobile app solving e-commerce trust angle
- Retail Dive: Trust technology for small sellers angle
- Mobile Commerce Daily: Mobile-first verification approach

Post-Launch Growth (Months 2-6)

Organic Channels

SEO Strategy:

- Target Keywords: "seller verification app," "mobile trust badge," "e-commerce fraud prevention app"
- Content Plan: 2 posts/week -- seller case studies, trust statistics, verification tutorials
- Link Building: Guest posts in e-commerce blogs, seller community partnerships

Content Marketing Calendar:

- Mondays: Seller success stories (how verification increased conversions)
- Wednesdays: Trust industry news with TrustSeal insights
- Fridays: Mobile verification tutorials and tips

Social Media Strategy:

- LinkedIn: Daily posts in seller/entrepreneur groups, focus on conversion metrics
- TikTok/Instagram: Short verification demos, "seller vs. scammer" comparisons
- Reddit: Weekly value-add posts in seller communities, monthly AMA sessions

Paid Channels

Recommended Ad Platforms:

1. Facebook/Instagram Ads: Target e-commerce sellers with trust-related interests
2. Google Ads: Target "seller verification" and "trust badge" searches
3. Reddit Promoted Posts: Target verified content in seller communities

Budget Allocation:

Channel	Monthly Budget	Expected CAC	Expected ROAS
Facebook Ads	$2,500	$45	3.2x
Google Ads	$1,800	$52	2.8x
Reddit Promoted	$700	$38	3.5x

A/B Testing Plan:

- Creative: Seller pain (lost sales) vs. solution benefit (trust badge)
- Audiences: Platform-specific sellers vs. general e-commerce
- CTAs: "Get Verified" vs. "Stop Losing Sales" vs. "Join 1000+ Sellers"

Viral & Referral

Referral Program Design:

- Seller referrals: 1 month free for every seller referred (both parties)
- Consumer referrals: Premium verification features unlock after 3 successful referrals
- Viral mechanics: Trust badge displays show "Verified by TrustSeal" on listings

Partnership Opportunities:

- Shopify Apps: Integration partnership with inventory/listing management apps
- Payment Processors: Co-marketing with Stripe, Square for "verified seller" campaigns
- Influencer Partnerships: E-commerce YouTubers who discuss trust issues

Pricing Strategy

Pricing Model: Freemium with seller-focused subscription tiers

Tier	Price	Features
Basic (Consumer)	Free	View seller verification, basic scam alerts
Seller Starter	$19/month	10 verifications, basic trust badge
Seller Pro	$49/month	Unlimited verifications, premium badge, analytics
Seller Enterprise	$99/month	API access, white-label badges, priority support

Pricing Page Layout:

- Problem-first: "You're losing $X/month to trust issues"
- Value calculator: Input monthly sales, see potential recovery from trust badges
- Social proof: "2,847 sellers recovered $847K in lost conversions"

Competitive Pricing Analysis:

Competitor	Monthly Cost	Features	TrustSeal Advantage
Trustpilot	$299/month	Review management	Mobile-first, seller-controlled
eBay Managed	Platform %	Native platform only	Cross-platform, instant setup
PayPal Protection	Transaction %	Payment-layer only	Pre-purchase verification

Conversion Optimization

Onboarding Flow (Sellers):

1. Download app -> Phone verification
2. Business verification -> Upload business license/tax ID
3. Platform connection -> Connect existing listings (Shopify, eBay, etc.)
4. First badge generation -> 60-second verification process
5. Badge placement tutorial -> Step-by-step platform-specific instructions
6. Analytics setup -> Conversion tracking installation

Activation Metrics:

- Day 1: Complete business verification (target: 70%)
- Day 7: Generate first trust badge (target: 45%)
- Day 14: Install badge on at least 3 listings (target: 30%)
- Day 30: Report conversion increase or maintain subscription (target: 65%)

Retention Strategy:

- Email sequence: Weekly conversion reports, trust industry insights
- Push notifications: Badge performance updates, new verification opportunities
- In-app: Monthly trust score updates, competitor comparison reports

Churn Reduction:

- Exit survey: Required before cancellation to identify specific pain points
- Win-back campaign: "Your competitors are verified, you're not" emails
- Downgrade option: Reduce to basic tier instead of full cancellation

Key Metrics & Targets

Metric	Month 1	Month 3	Month 6
App Downloads	1,200	4,500	12,000
Seller Signups	150	650	1,800
Consumer Signups	800	3,200	9,500
Paying Sellers	25	95	280
Seller Conversion Rate	16%	14%	15%
Monthly Churn (Sellers)	8%	6%	5%
MRR	$1,150	$4,200	$13,100
Consumer DAU	200	1,100	3,800

KEY INSIGHT
Success depends on seller activation (badge creation + placement) more than raw downloads. Target 30% of signups completing full verification within 14 days.

Marketing Budget Estimate

Channel	Monthly Budget	Expected CAC	Expected ROI
Facebook/Instagram Ads	$2,500	$45	3.2x
Google Ads (Search)	$1,800	$52	2.8x
Reddit Promoted Posts	$700	$38	3.5x
Content Creation	$1,200	N/A	Long-term SEO
Influencer Partnerships	$800	$35	3.8x
PR/Press Outreach	$500	N/A	Brand building
Community Management	$600	N/A	Retention support
Total Monthly Budget	**$8,100**	**$47 average**	**3.1x blended**

PRO TIP
Start with 60% budget on Reddit/community-based channels where sellers are already discussing trust issues. Scale Facebook ads only after proving conversion rates with organic seller traffic.

HEADS UP
Avoid broad "e-commerce" targeting on paid channels -- focus on sellers with trust-specific pain points. Generic e-commerce targeting will burn budget on sellers who haven't experienced conversion loss due to trust issues.

Sources & References

1. [1] Provided validation data and user complaint evidence
2. [2] Reddit community membership numbers from platform data

3. [3] E-commerce market sizing from industry reports referenced in market validation
4. [4] Mobile commerce penetration rates from eMarketer 2025 data

06 Project Execution & Delivery Roadmap

EXECUTIVE SUMMARY

In short: TrustSeal is a medium-complexity mobile app requiring 4-5 people for 12 weeks and $85-120K to ship an MVP that solves the #1 user complaint (fake pricing). The biggest risk is building seller verification features users don't actually want yet -- ship the price-checking core first, validate demand, then expand.

- Build Complexity: Medium. Price verification + barcode scanning + basic UI are straightforward; the risk is integrating multiple price APIs reliably and handling edge cases (out-of-stock items, regional pricing variance).
- Recommended Team: 2 backend engineers (Node.js/serverless) + 1 React Native mobile engineer + 1 full-stack designer/PM (can start part-time) + 1 QA/DevOps.
- Timeline to Launch: 12 weeks (MVP to App Store/Play Store submission). First internal alpha in Week 4; beta testing in Weeks 8-10; public launch Week 12.
- Budget Envelope: $85K-$120K (within stated "medium" range). Development dominates; infrastructure and third-party APIs are secondary spend.
- Biggest Risk: Platform dependency. If Google Shopping API or Amazon's pricing data blocks your scraping, the core feature dies. Mitigation: build fallback to Barcode Lookup API + manual price entry for MVP.
- Recommendation: Build now. **Demand score is 80/100, sentiment is clearly negative on existing solutions (competitors average 1.3/5 stars), and the MVP is achievable in 12 weeks with a lean team. The Trend score (76/100) confirms this is a breakout topic. However,** do NOT build seller verification or advanced fraud detection in v1 -- those can wait for post-launch feedback.

Snapshot

Metric	Value
Total Timeline (MVP -> Launch)	12 weeks
Recommended Team Size	4-5 people (2 backend, 1 mobile, 1 designer/PM, 0.5 QA)
Total Budget Envelope	$85K-$120K
Methodology	Agile Scrum, 1-week sprints
First Shippable Milestone	Week 4 -- internal alpha (price verification working end-to-end)
Beta Launch	Week 9 -- TestFlight/Google Play beta with 50 testers
Public Launch Target	Week 12 -- iOS App Store + Google Play
Infrastructure Hosting	AWS (Lambda, RDS, ElastiCache) -- ~$200-400/month at scale

Third-Party API Spend	~$100-300/month (Google Shopping, Barcode Lookup, UPC database)

Project Overview

Estimated Total Duration: 12 weeks from project kickoff to public app store launch.

Team Size Recommendation: 4-5 people for the critical path.

Role	FTE	Notes
Backend Engineer (Lead, serverless)	1.0	Owns Lambda functions, RDS schema, API design
Backend Engineer (Integration)	1.0	Owns third-party API integration, caching layer
Mobile Engineer (React Native)	1.0	Owns iOS/Android builds, barcode scanning, UI
Designer / Product Lead (part-time)	0.5	Owns wireframes, App Store listing, launch prep
QA / DevOps (part-time)	0.5	Owns CI/CD, testing automation, production monitoring

This 4-5 person team is NOT negotiable for a 12-week timeline. A smaller team (2-3 people) extends the timeline to 18-24 weeks. A larger team (6+) creates communication overhead that kills velocity on a tight deadline.

Development Methodology: Agile Scrum with 1-week sprints. Daily standups (15 min), sprint planning (Friday before Monday start), retro/review (Friday end-of-week). This cadence is essential for a 12-week runway because scope creep kills shipping dates.

Deployment Environment: AWS (as recommended in tech stack). CI/CD via GitHub Actions -> EAS Build (React Native) -> AWS Lambda (backend) -> App Store + Play Store (mobile).

Documentation & Comms Tool: Notion (shared roadmap, sprint backlog, launch checklist) + Slack (engineering updates) + linear.app (issue tracking).

Phase Breakdown

Phase 1: Foundation & Setup (Weeks 1-2)

Sprint Goal: Establish development environment, secure API access, deploy empty MVP skeleton to production, set up monitoring.

Activities:

- Week 1, Day 1-2:

  Set up AWS account (RDS, Lambda, ElastiCache, IAM roles).

Apply for price API access: Google Shopping API, Barcode Lookup API, UPC Database API. These approvals can take 3-7 days, so **start immediately**.

Create GitHub org/repo, set up branch protection (main requires PR review).

Bootstrap React Native project via Expo (faster than bare React Native for MVP).

Create Figma file for MVP wireframes (price checking screen, result screen, barcode scanner screen).

- Week 1, Day 3-5:

  Database schema creation: `products`, `sources`, `prices`, `verifications` tables (see tech spec). Run first migration.

  Lambda function stubs for price verification, barcode lookup, product data service (hello-world endpoints).

  GitHub Actions pipeline: on PR merge to main, run tests -> build Lambda -> deploy to staging.

  Mock third-party API responses in staging (so mobile team can build UI in parallel).

- Week 2, Day 1-3:

  Mobile team: Expo project scaffold, navigation setup (React Navigation), screen stubs.

  Backend: RDS schema validation, first production (non-sensitive) deployment to AWS Lambda.

  Designer: Finalize 3-4 core wireframes (URL input screen, barcode camera, result card).

- Week 2, Day 4-5:

  Security: Enable AWS VPC for RDS, rotate API keys, set up environment variables (no secrets in code).

  Monitoring: CloudWatch dashboards for Lambda error rate + RDS CPU. PagerDuty or Sentry for crash alerts (optional; can defer to Week 8).

  QA: Create testing checklist, manual test cases for price verification flow.

Estimated Effort: 12 person-days Deliverables:

- AWS infrastructure running (RDS, Lambda, ElastiCache online)
- GitHub Actions CI/CD pipeline (green builds)
- React Native Expo project compiling on iOS + Android simulators
- API access keys for 2-3 price sources (Google Shopping, Barcode Lookup)
- Figma MVP wireframes signed off

Dependencies to Resolve Before Week 2 Ends:

- Third-party API approvals **-- if Google Shopping takes >5 days, pivot to** Barcode Lookup + manual fallback for MVP.
- AWS billing alert set (prevent runaway costs).

Phase 2: Core MVP Features (Weeks 3-7)

Organized as two 2-week sprints + 1 Polish week.

Sprint 1: Price Verification Engine (Weeks 3-4)

Sprint Goal: Build end-to-end price checking. User enters a product URL -> app queries APIs -> shows risk level.

Features to Build:

1. Price Lookup Service (Lambda function):
   - Input: product URL or UPC barcode.
   - Logic: parse URL (extract product ID) -> query Google Shopping + Barcode Lookup APIs -> aggregate prices -> calculate average + std dev.
   - Output: JSON with `{average_price, user_price, deviation_percent, risk_level: "green|yellow|red"}`.
   - Handle: product not found, API timeout (2-sec fallback), regional pricing variance.
2. Mobile: URL Input Screen:
   - Text field + paste button (users will copy URLs from other apps).
   - Submit button -> calls `/api/verify-price` endpoint.
   - Loading spinner, error toast if API fails.
3. Mobile: Price Result Card:
   - Display price deviation as "XX% below/above market average".
   - Color coding: Green <=5%, Yellow 5-15%, Red >15%.
   - Call-to-action button: "Mark as Scam" (log for future analysis).
   - Skip "Share" or "Save" buttons for MVP.
4. Database: Populate `products` and `prices` tables with each lookup (for analytics later).
5. Caching: Redis cache of product prices (4-hour TTL) to reduce API calls and latency.

Dependencies:

- Google Shopping API keys live (or fallback ready).
- RDS schema complete.
- Mobile navigation routing in place.

Estimated Effort: 10 person-days (2 backend, 1 mobile for UI, 1 designer for polish).

Testing Focus:

- Price accuracy: Does the app show the same price as manual lookup?
- False positives: What % of legit deals get flagged as "red"?
- API latency: Target <2 sec end-to-end response time.
- Error handling: What happens if Google Shopping times out?

Deliverables:

- Price verification Lambda function tested (unit tests >80% coverage).
- Mobile app screens render and call API successfully.
- Internal testing against 10-20 real product URLs.
- Caching working (verify Redis TTL in logs).

Sprint 2: Barcode Scanning + Seller Reputation (Weeks 5-6)

Sprint Goal: Add barcode camera as second input method. Build basic seller scoring from public data (Amazon ratings, Trustpilot).

Features to Build:

1. Barcode Scanner Screen (Mobile):
   - Camera permission request (iOS + Android).
   - Barcode scanning library: `react-native-camera` + `@react-native-camera-roll/camera-roll` (or Expo Camera API for simpler integration).
   - Scan -> auto-populate UPC field -> trigger price lookup from Sprint 1.
   - Fallback: manual UPC input field (for users without camera permission).
2. Seller Scoring Service (Lambda function):
   - Input: seller name (extracted from URL or manually entered).
   - Logic: Query public APIs (Trustpilot, Amazon seller ratings, BBB if available).
   - Output: `{seller_name, avg_rating, review_count, trust_badge: true|false}`.
   - MVP rule: Show badge only if avg_rating >=4.0 + >=50 reviews. Otherwise, "Unverified Seller -- check ratings independently."
   - Do NOT build proprietary verification (seller onboarding, KYC). That's post-launch.
3. Updated Result Card:
   - Seller info below price deviation.
   - "Verified Seller" badge if thresholds met, else "Check seller reviews" link to external ratings.
4. Database:
   - Add `sellers` table to track seller reputation scores (cache for 7 days).

Dependencies:

- Barcode scanning library working on both iOS + Android (can add 3-5 days of integration friction).
- Trustpilot / Amazon APIs accessible (some may require paid access; evaluate cost vs. MVP scope).

Estimated Effort: 8 person-days (1 backend for seller APIs, 1 mobile for camera, 1 designer for updated result card).

Risk Mitigation:

- If barcode scanning breaks on Android: Defer to Week 8, ship URL-only MVP in Week 7.
- If Trustpilot API is paid-only: Use RSS feeds or public web scraping (fragile but free). Document this as "Weeks 8-10 improvement."

Testing Focus:

- Barcode scanning accuracy (test 20 UPC codes, real-world lighting conditions).
- Seller API latency (target <1 sec).
- False negatives in seller scoring (do scam sellers with fake reviews get flagged?).

Deliverables:

- Barcode scanner working on iOS + Android simulators + 1 real device.

- Seller reputation service returning results.
- Result card updated with seller info.
- Internal testing complete (30+ product scans).

Sprint 3: Polish & Edge Cases (Week 7)

Sprint Goal: Fix crashes, improve UX, handle edge cases, prepare for beta testing.

Activities:

1. Error Handling:

   What if product is out of stock on all APIs? -> "No price data found -- not a scam signal, but unverified."

   What if URL is invalid? -> Clear error message, example link shown.

   What if API is down? -> "Price checking temporarily unavailable. Try again later."

2. Performance:

   Measure app startup time (target <3 sec).

   Measure price lookup latency (target <2 sec with caching).

   Optimize image sizes, reduce bundle size (React Native can bloat fast).

3. UI Polish:

   Accessibility: button contrast, text size, alt text for icons.

   Keyboard handling: dismiss keyboard on result, scroll if needed.

   Loading states: skeleton screens instead of spinners (feels faster).

4. Documentation:

   README for developers (how to run locally, environment setup).

   Deployment runbook (how to push to staging/prod).

Estimated Effort: 6 person-days.

Deliverables:

- App crashes reduced to <5 during manual testing (20 happy-path runs).
- Performance metrics logged to CloudWatch (ready for Week 9 analysis).
- App store metadata drafted (description, keywords, screenshots).

Phase 3: Integration & Final Infrastructure (Week 8)

Sprint Goal: Wire all components together, stress-test, prepare for beta.

Activities:

1. End-to-End Testing:

   User enters URL -> price verification runs -> barcode scan works -> seller info appears. All on a real device (not simulator).

   Test on different network conditions (WiFi, 4G, poor signal).

2. Third-Party Integration Validation:

Confirm Google Shopping API is not rate-limiting.

Confirm barcode lookup is returning correct results.

Monitor Redis cache hit rate (target >50%).

3. Analytics Setup (Optional for MVP, but recommended):

Log each price verification to CloudWatch (for later analysis of false positive rate).

No user PII; just timestamp, product category, risk level.

4. Staging Environment Hardening:

RDS backups automated (AWS RDS automatic backups, 7-day retention).

Lambda concurrency limits set (prevent runaway costs).

VPC security groups tightened.

5. TestFlight/Internal Testing Prep:

Sign iOS certificates (can be tedious; allocate 2 days).

Generate Google Play internal test link.

Create beta testing guide (10 test cases for internal team to run).

Estimated Effort: 5 person-days.

Deliverables:

- End-to-end flow tested on iPhone + Android physical devices.
- Staging environment locked down and monitored.
- TestFlight + Play Store internal testing links live.
- Runbook for handling critical production bugs.

Phase 4: Testing & QA (Weeks 9-10)

Sprint Goal: Beta test with external users, collect feedback, fix critical bugs, validate core metrics.

Activities:

1. Beta Testing Program:

Recruit 50 beta testers from email list or Product Hunt early access. (Assume ~40% signup rate from target audience.)

Distribute via TestFlight (iOS) + Google Play internal testing (Android).

Create feedback form (Google Form or Typeform): "Did the price alert help you avoid a scam? Y/N. Any crashes? Comments?"

2. QA Testing Matrix:

Test Case	Platform	Status	Notes
Happy path: scan barcode -> get result	iOS, Android	PASS	Monitor latency
URL with special characters	Both	PASS	Edge case handling
No internet connection	Both	FAIL -> FIX	Show offline message
API timeout (>2 sec)	Both	FAIL -> FIX	Fallback to cache or error msg

Barcode not found in database	Both	PASS	Show "Barcode not recognized"
Seller with zero reviews	Both	PASS	Show "Unverified" label
App backgrounded + resumed	Both	FAIL -> FIX	Lost state on Android
Device with small screen (iPhone SE)	iOS	PASS	Responsive design
Dark mode support	Both	FAIL -> DEFER to Week 11	Nice-to-have

3. Crash & Stability Metrics:

   Set threshold: <0.5% crash rate (use Firebase Crashlytics or Sentry for tracking).

   Review CloudWatch logs daily for Lambda errors.

   Fix P0 bugs within 24 hours; P1 within 48 hours.

4. Performance Benchmarking:

   Measure app startup time on 4G (target <3 sec).

   Measure price lookup latency (target <2 sec with caching, <3 sec on first lookup).

   If failing, optimize Lambda cold starts (pre-warm, or use Provisioned Concurrency -- adds cost).

5. User Feedback Analysis:

   Tally feedback by category: UX friction, feature requests, crashes.

   Plot "false positive rate" (% of results marked "red" that user says weren't scams) -- target <10%.

   If >15%, adjust risk thresholds (e.g., change "red" from >15% deviation to >20%).

Estimated Effort: 8 person-days (1 backend for bug fixes, 1 mobile for crash analysis, 1 QA for test execution, 0.5 designer for UX feedback).

Critical Path Decision (end of Week 10):

- If stability >95% AND false positive rate <10%: Proceed to public launch (Week 11-12).
- If stability <95% OR false positive rate >15%: Extend beta by 1 week (Week 11) before launch.

Deliverables:

- Beta testing completed with 30+ active testers.
- Crash report log reviewed and critical bugs fixed.
- False positive rate documented (baseline for future improvements).
- Feedback themes summarized in Notion (for post-launch roadmap).

Phase 5: Launch Preparation (Weeks 11-12)

Sprint Goal: Deploy to production, submit apps to App Stores, coordinate public launch.

Activities:

Week 11:

1. Production Deployment:

Tag release version (v1.0.0) in GitHub.

Deploy Lambda functions to production (separate AWS account if possible, but "medium" budget may not support this -- use different RDS environment at minimum).

RDS: Enable automated backups (7-day retention), enable encryption at rest.

Set up production monitoring: CloudWatch dashboards for Lambda error rate, RDS CPU, API latency. Alert thresholds: Lambda errors >5/min, RDS CPU >80%, API latency >3 sec.

2. App Store Submissions:

iOS: Create App Store Connect account, fill metadata (description, keywords, screenshots, privacy policy), upload build via Xcode/EAS.

Android: Create Google Play Console account, fill metadata, upload APK/AAB.

Both: Privacy policy (required). Include: "We do not collect or sell user data. Price lookups are logged anonymously for analytics." Link to privacy policy from app settings.

Both: Screenshots (3-5 per platform showing price alert, barcode scanning, result card). Allocate designer 4 hours for high-quality screenshots.

Budget time: **5-7 days for app review** (Apple is slower than Google, 2-3 days avg).

3. Documentation Finalization:

In-app onboarding: 1-2 screens explaining "Scan suspicious deals" + "See price deviation." Skip deep explanations; keep it <20 seconds.

Privacy policy (required for both app stores).

Support email: support@trustseal.app (monitored by team).

FAQ: "Is this a scam?" (No, we're a verification app), "Why is this deal marked red?" (If price deviation >15%, we flag it).

4. Marketing Assets:

App icon (1024x1024 PNG, design in Figma, 2 hours).

App Store banner/featured image (designer, 2 hours).

1-minute explainer video (optional; defer to Week 12 if tight on time).

Landing page (simple: problem -> screenshot -> download buttons). Use Webflow or Framer (designer, 4 hours).

Week 12:

1. Public Launch:

Confirm app reviews pass (expect 1-2 rejection cycles for metadata clarification; add 2-day buffer).

Set app as "live" on both App Store + Play Store (same day for launch impact).

Announce on Product Hunt (1 day of heavy traffic, 50-500 installs expected).

Email beta testers: "App is now live! Leave a review." (30-50 expected).

Post on Reddit, Twitter (use #ecommerce #shopping #scams hashtags).

Create 3-5 "launch day" blog posts: "How We Stopped Fake Pricing Scams" (publish day of launch).

2. Monitoring Post-Launch:

Team on-call for first 48 hours (monitor Sentry, CloudWatch, email support).

If crash rate >5%, roll back to last stable version immediately.

If API latency >5 sec, debug Lambda concurrency + RDS slow queries.

3. First Week Metrics Dashboard:

Track: installs, daily active users (DAU), price lookups per user, crash rate, App Store rating.

Set weekly target: 500+ installs, 100+ DAU, 4.0+ star rating (based on 50+ reviews).

Estimated Effort: 10 person-days (1 backend for prod deployment, 1 mobile for app store submission, 1 designer for assets, 1 PM for coordination).

Deliverables:

- iOS app on App Store (live).
- Android app on Play Store (live).
- Landing page live.
- Support email monitored.
- Monitoring dashboards active.
- Launch announcement published (Product Hunt, Twitter, email).

Resource Plan

Team Composition (12-week MVP):

Role	Allocation	Duration	Estimated Cost Range
Backend Engineer (Lead, serverless/Node.js)	1.0 FTE	12 weeks	$28K-$35K
Backend Engineer (API integration)	1.0 FTE	12 weeks	$28K-$35K
Mobile Engineer (React Native)	1.0 FTE	12 weeks	$28K-$35K
Designer / Product Lead	0.5 FTE	12 weeks	$12K-$15K
QA / DevOps (part-time)	0.5 FTE	12 weeks	$8K-$10K
Freelance App Store Copywriter (Week 11)	0.1 FTE	1 week	$1K-$2K
SUBTOTAL: Payroll + Contract Labor	--	--	**$105K-$127K**

HEADS UP
The above is at the upper end of the "medium" budget range ($50K-$150K). If your budget is strictly $85K-$105K, you have two options:

Option A (Recommended): Lean Team, Longer Timeline

- 2 engineers (backend + mobile) + 1 designer/PM (0.5 FTE) + 1 QA (0.25 FTE).
- Timeline extends to 14-16 weeks.
- Total cost: $70K-$95K (inside medium range).
- Trade-off: slower iteration, less frequent production deploys.

Option B: Outsource Non-Critical Work

- Keep 3-person core team ($80K).
- Contract QA ($5K) + App Store copywriting ($2K) separately.
- Total: $87K-$95K.
- Trade-off: integration friction with contractors.

For the purposes of this roadmap, I recommend Option A: Extend timeline to Week 14, reduce QA/DevOps allocation, pair the designer with product responsibilities.

Budget Allocation Breakdown (assuming $85K-$105K total):

Category	Estimated Cost	Notes
Development (Payroll)	$70K-$85K	3 engineers (or 2.5 FTE over 14 weeks)
Infrastructure (6 months)	$2K-$4K	AWS (Lambda, RDS, ElastiCache, data transfer). Estimate: $300-500/month at launch; scale to $500-800/month by Month 6.
Third-Party Services (6 months)	$1.5K-$2.5K	Google Shopping API ($50-200/month), Barcode Lookup ($30-100/month), UPC Database ($20-50/month). Start with free/cheap tiers; upgrade as volume grows.
Design & Copywriting	$3K-$5K	Wireframes, App Store screenshots, landing page copy.
Testing & Tools	$1K-$2K	TestFlight, Firebase Crashlytics (free), Sentry (free tier), GitHub Actions (free).
Contingency (10%)	$7K-$10K	Buffer for unexpected delays, additional API costs, or hiring overruns.
TOTAL (MVP to Public Launch)	**$85K-$110K**	Within medium budget range.

NOTE

Pro Tip: If you're bootstrapping, cut payroll by hiring 1 junior engineer at $18K-$22K (instead of mid-level at $28K+) and a part-time designer at $8K-$12K. This drops total to $60K-$75K but extends timeline to 16-18 weeks. Only viable if you have co-founder sweat equity.

Risk & Mitigation Timeline

Risk	Impact	Likelihood	Mitigation	When to Address
Third-party API restrictions (Google Shopping blocks scraping, Trustpilot paywall)	**HIGH** -- Core feature dies	**MEDIUM** (30%)	**Week 1:** Test API access immediately. If blocked, pivot to Barcode Lookup + manual price entry. Negotiate enterprise tier with Google if traction exists.	**Weeks 1-2** (before development starts)

Barcode scanning breaks on Android	**MEDIUM** -- Feature delayed, ship URL-only MVP	**LOW** (15%)	Use `react-native-camera` or Expo Camera API. Test on real Android device in Week 5. If issues, defer barcode to post-launch.	**Week 5** (during Sprint 2)
React Native performance issues (app startup >5 sec, crashes on low-end devices)	**MEDIUM** -- Poor user retention	**MEDIUM** (25%)	Profile with React DevTools in Week 7. Reduce bundle size, lazy-load screens. Test on budget phones (iPhone SE, Pixel 3).	**Week 7** (polish phase)
RDS database performance degrades under concurrent users (price lookups hammer DB)	**HIGH** -- Latency kills user experience	**MEDIUM** (30%)	Cache aggressively in Week 8 (Redis 4-hour TTL). Monitor RDS slow query log. Add read replicas if needed (costs +$300/month but deferred to post-launch).	**Week 8** (integration phase)
App Store review rejects app (privacy policy unclear, misleading category)	**MEDIUM** -- Launch delayed 5-7 days	**LOW** (10%)	Submit Week 10 (not Week 12) for extra review cycles. Use Apple's app review guidelines checklist. Allocate 2-day buffer for resubmission.	**Week 10** (testing phase)
Key engineer quits or gets sick	**HIGH** -- Timeline slips 2-4 weeks	**LOW** (10% across 12 weeks)	Document code continuously (PR reviews, comments). Cross-train mobile engineer on basic backend. Hire contractor backup for Week 8+ (cold-start buffer).	**Ongoing** (sprint planning)
User acquisition is 50% lower than forecast (only 250 installs vs. 500+ target)	**MEDIUM** -- Morale dip, pivot to marketing	**MEDIUM** (40%)	Don't over-index on launch week installs. Focus on DAU retention (target: 20% of installs stay active after 7 days). If true retention is good, growth is a sales problem, not a product problem.	**Week 2+** (post-launch)
Fraud detection false positive rate >20% (too many legit deals flagged as scams)	**HIGH** -- Users distrust the app	**MEDIUM** (35%)	In Week 9 beta, track false positive rate religiously. If >15%, adjust deviation threshold from 15% to 20% (less aggressive). Plan ML-based tuning for post-launch.	**Week 9** (beta testing)

Risk Escalation Rules:

• P0 (Stop-the-line): Third-party API blocked, app crashes >5%. Escalate to founder immediately; halt other work.

• P1 (High priority): Barcode scanning broken, false positive rate >20%. Fix before launch, but don't block others.

• P2 (Medium): Slow RDS queries, minor UI bugs. Schedule for next sprint or defer to post-launch.

• P3 (Low): Dark mode not working, cosmetic issues. Defer to v1.1.

Milestone Calendar

Milestone	Target Date	Success Criteria	Owner
M1: Dev Environment Ready	End of Week 2	AWS account live, GitHub Actions running, React Native compiling, API keys secured, Figma wireframes signed off	DevOps + Designer
M2: Price Verification MVP	End of Week 4	End-to-end price lookup working on simulator, <2 sec latency, >80% test coverage on Lambda functions	Backend Lead + Mobile
M3: Barcode Scanning & Seller Scores	End of Week 6	Barcode scanner working on iOS + Android, seller reputation service returning results, internal testing with 30+ UPC scans	Backend Integration + Mobile
M4: Polish & Edge Cases	End of Week 7	<5 crashes in 20 manual test runs, performance metrics logged, app store metadata drafted	Mobile + QA
M5: Integration & Monitoring	End of Week 8	End-to-end flow tested on real devices, staging environment locked, TestFlight + Play Store internal links live	Backend Lead + QA
M6: Beta Testing Complete	End of Week 10	40+ beta testers active, crash rate <0.5%, false positive rate <10%, feedback themes documented	QA + Designer
M7: Production Live	Mid Week 12	iOS app on App Store, Android app on Play Store, monitoring dashboards active, support email monitored	Eng Lead + Ops
M8: Launch Announced	End of Week 12	Product Hunt live, 500+ installs, 100+ DAU, 4.0+ star rating (on 50+ reviews), landing page traffic tracked	PM + Designer

Budget Estimate Summary

Total MVP to Launch (Weeks 1-12):

Category	Low Estimate	High Estimate	Notes
Development	$70K	$85K	3 engineers × 12 weeks at $18-22K/week (mid-market rates)
Infrastructure (AWS)	$1.5K	$2.5K	Lambda, RDS, ElastiCache, data transfer. ~$150-200/month × 6-12 months. High estimate assumes 10K+ DAU scale testing.
Third-Party APIs	$1K	$2K	Google Shopping, Barcode Lookup, UPC Database. Start free tier, upgrade as volume grows.
Design & UX	$2K	$3K	Wireframes, App Store screenshots, landing page. Designer 0.5 FTE × 12 weeks.
DevOps & Testing Tools	$500	$1K	Firebase Crashlytics (free), Sentry (free), TestFlight (free), GitHub Actions (free). Estimate for premium monitoring if added.
Copywriting & Compliance	$1K	$2K	App Store copy, privacy policy, legal review (optional).
Contingency (8-10%)	$3K	$8K	Buffer for scope creep, unexpected API costs, additional QA cycles.
TOTAL	**$79K**	**$104K**	**Within "medium" budget range ($50K-$150K).**

Post-Launch Monthly Burn (Months 3-6):

Item	Monthly Cost
AWS Infrastructure	$400-$600 (scales with user base)
Third-Party APIs	$100-$300
Team (1 FTE engineer for maintenance)	$5K-$8K
Monitoring & Tools	$200-$500
TOTAL	~$6K-$9K/month

Profitability Assumptions (Post-Launch):

- Freemium model: 80% free users (price checks only), 20% paid (premium features like saved alerts, seller reports).
- Paid tier pricing: $2.99-$4.99/month (in-app subscription).
- Break-even DAU: ~2,000-5,000 at 10-15% conversion to paid, ~$3/ARPU.
- Timeline to break-even: 6-12 months post-launch, assuming consistent 20% MoM growth in DAU.

NOTE
Key Insight: This roadmap is cost-optimized for a 12-week MVP, not for indefinite operation. Post-launch, you'll spend 30% of time on scaling infrastructure (multi-region, CDN, database optimization) and 70% on user acquisition. Budget another $20K-$40K/quarter for that phase if you're shooting for profitability by Month 9.

Post-Launch Plan (Months 3-6)

Phase 2 Feature Development (Weeks 13-26)

Month 3 (Weeks 13-16): Seller Verification & Review Integration

- Full seller verification system: onboard sellers, verify business licenses, display verification badges.
- Integrate Trustpilot + Amazon reviews at scale (currently just scraped).
- Seller Dashboard: sellers can claim profiles, respond to scam reports.
- User reviews of sellers (in-app only, separate from TrustScore).
- Estimated effort: 8 person-weeks (2 engineers, 1 designer).
- Cost: $12K-$18K.

Month 4 (Weeks 17-20): Saved Alerts & Personalization

- Price watch feature: users set price alerts for items they're tracking.
- History of scans: searchable, sortable by category.
- Trending scams report: weekly email of hot topics in user's region.
- Push notifications (opt-in) for price drops on saved items.
- Estimated effort: 6 person-weeks (1 backend, 1 mobile, 0.5 designer).
- Cost: $9K-$14K.

Month 5-6 (Weeks 21-26): Analytics & Scaling

- Analytics dashboard for sellers: view scan volume, false reports, engagement.
- Fraud detection ML model: train on historical data to reduce false positives.
- Regional expansion: localize for 2-3 new markets (pricing data varies by region).
- Performance optimization: optimize for markets with 2G/3G connectivity.
- Estimated effort: 8 person-weeks (2 backend, 1 ops, 0.5 designer).
- Cost: $12K-$18K.

Team Scaling Plan

Month 3: Add 1 full-time backend engineer (focus on seller platform). Promote designer to Head of Product (if early traction validates market fit).

Month 4: Hire 1 customer success person (respond to seller onboarding support, triage scam reports).

Month 5: Consider hiring 1 community manager / growth marketer (if DAU >10K).

Month 6: Evaluate hiring VP of Sales if enterprise seller deals are emerging.

Total team by Month 6: 5-6 people (vs. 4-5 for MVP).

Scaling Triggers & Actions

Trigger	Action	Timeline
DAU reaches 5K	Upgrade RDS to multi-AZ deployment	Week 14
API errors >2%	Audit Lambda concurrency, add provisioned capacity	Week 15
Installs plateau <500/week	Launch paid acquisition campaign (Google Ads, Facebook)	Week 16
Seller requests >10/week	Hire contractor for seller onboarding support	Week 17
Competition launches copy	Add unique feature (ML fraud detection, community reports)	Week 18

Iteration Cycle (Post-Launch)

- Weekly: Review DAU, crash rate, false positive rate. Prioritize top 3 bugs.
- Biweekly: Sprint planning for Phase 2 features. Align with user feedback from Trustpilot app reviews.
- Monthly: Product review: which features drove retention? Which did users ignore? (Data informs next quarter's roadmap.)
- Quarterly: Board/investor update. Metric: DAU, ARR, CAC, LTV.

Revenue Projections (Conservative)

Metric	Month 3	Month 6
DAU	2,000	8,000-12,000
Paying subscribers	100-150	800-1,200
MRR (at $3 ARPU)	$300-450	$2,400-3,600
CAC (via organic + ASO)	$5-10	$3-8 (improves with scale)
Gross margin (SaaS)	~70%	~75%

NOTE

Note: These projections assume steady 10-15% MoM growth in DAU and 15% conversion to paid. Actual results depend heavily on marketing spend, competitive response, and product-market fit validation. If false positive rate is >15% or DAU growth stalls <5% MoM, pivot to a different monetization model (e.g., B2B seller reports, API licensing).

Decision Checklist

Complete these actions within the next 7-30 days to begin execution:

1. Validate Third-Party API Access (Days 1-3): **Apply for Google Shopping API, Barcode Lookup, and UPC Database access. Document API response times and rate limits. If any are unavailable or require enterprise licensing, document the workaround (e.g., fallback to manual price input).** Owner: Lead Backend Engineer. Blockers here can delay MVP by 1-2 weeks.
2. Finalize Team & Hiring Plan (Days 1-5): **Confirm 3-4 core engineers (backend lead, backend integration, mobile). For "lean timeline" option, confirm willingness to extend to Week 14 and reduce QA allocation. Identify freelance designer for App Store assets (Week 11). Post job descriptions if hiring externally.** Owner: Founder/PM. Budget: $1K-$2K for recruitment ads (optional; referrals preferred).
3. Set Up Development Infrastructure (Days 3-7): **Create AWS account, RDS instance, Lambda environment, GitHub org/repo. Enable billing alerts (set limit at $500/month for first 12 weeks). Generate API keys securely (store in 1Password or LastPass, never in code). Set up GitHub Actions CI/CD skeleton.** Owner: DevOps/Backend Lead. Budget: $0 (AWS free tier for initial setup).
4. Draft MVP Wireframes & User Flows (Days 5-10): **Finalize 4-5 core screens (URL input, barcode camera, price result, seller info, onboarding). Create user flow diagram (entry point -> scan -> result -> action). Share with team for feedback. This guides development and prevents mid-sprint design changes.** Owner: Designer. Deliverable: Figma file linked in Notion.
5. Define Success Metrics & Instrumentation (Days 7-14): **Decide on critical metrics by Week 8 (crash rate <0.5%, false positive rate <10%, API latency <2 sec). Plan logging strategy: which events to capture for analytics? Set up CloudWatch dashboards before Week 1 dev starts.** Owner: PM + Backend Lead. Deliverable: Metrics doc in Notion; no code changes needed yet.
6. Negotiate or Validate Pricing APIs & Rate Limits (Days 10-14): **Confirm Google Shopping, Barcode Lookup, and UPC Database cost and volume tiers. If free tier caps are low (<1K queries/day), budget for paid tier ($50-$200/month) or identify cheaper alternatives. Document all API dependencies in a single spreadsheet (vendor, cost, SLA, fallback).** Owner: Backend Lead + PM. Deliverable: API pricing spreadsheet; informs budget finalization.
7. Lock Budget & Timeline with Stakeholders (Days 14-21): **Confirm total spend ($85K-$110K), team size (3-4 people), and 12-week timeline (or 14-week lean option). Get written sign-off on timeline and budget to prevent mid-project scope creep. Establish a "budget escalation" process for unplanned costs.** Owner: Founder. Deliverable: Signed budget memo; shared with team.

Sources & References

1. [1] PRD Summary -- "Build This FIRST" section, directly from submitted idea context.
2. [2] Validation Scores & Competitor Analysis -- Idea context, noting 236 competitors with 1.3/5 avg rating, indicating market is ripe for disruption.
3. [3] User Sentiment Evidence -- Google Play reviews cited in idea context: "false prices makes this useless," "scam promotions," "critical errors, crashes frequently."

4. [4] Tech Stack Recommendation -- Technical Architecture Blueprint (serverless + React Native + PostgreSQL + Redis), tailored to "medium" complexity and global region.
5. [5] Market Opportunity -- Idea context notes $20.35T mobile e-commerce market, with Trend score 76/100 (breakout topic, last 30 days).
6. --
7. END OF ROADMAP
8. --

Key Takeaways for Execution

This is a 12-week, $85K-$110K sprint. Your team is 3-4 people laser-focused on shipping price verification + barcode scanning, NOT on building seller verification or advanced fraud detection in v1.

The biggest execution risk is third-party API dependencies. Validate Google Shopping API access by Day 3. If blocked, pivot immediately to Barcode Lookup + manual fallback.

Months 3-6 are about validating product-market fit and scaling. If DAU breaks 5K and false positive rate is <10%, you're on track. If either metric misses, your post-launch roadmap pivots (e.g., from B2C to B2B seller tools).

Ship the "boring" version first. Price deviation alerts + barcode scanning are table stakes. Don't build seller onboarding, community reviews, or ML fraud detection until users are clamoring for it.

Good luck shipping.

Ready to Build This?

Turn this blueprint into working software.

Our team builds custom software for founders —
we'll take this blueprint and ship the product.

Contact us: **hello@unbuiltlab.com**

Unbuilt Lab

unbuiltlab.com