TABLE OF CONTENTS

Project 50

Executive Feasibility Snapshot

Comparative assessment of backend infrastructure, native voice APIs, and broader RTC room models for a live multiplayer quiz app

Product concept

Project 50 is a moderator-led live quiz app for 4 to 8 participants. Each match contains 50 questions and combines buzzer rounds, estimation questions, live scoring, and synchronized session flow.
The concept phase tested three service categories: Firebase for real-time gameplay infrastructure, voice-only APIs for native in-app audio, and broader managed RTC room models as a benchmark for fully embedded communication.

Executive decision

Proceed with Firebase-backed live gameplay, moderator controls, and score synchronization.
Exclude native in-app voice from the initial scope and rely on third-party platforms where voice is required.
Reason: the backend model remains commercially manageable, while voice and full RTC room services create a materially higher recurring cost base because they scale by connected participant minutes rather than quiz logic.

Firebase backend

Realtime session sync, moderator state, answer handling
Assessment: viable for MVP

Voice-only APIs

Native in-app audio between connected participants
Assessment: technically feasible, financially high-risk

Managed RTC rooms

Video + voice room model used as a conservative upper market benchmark

Assessment: not suitable for MVP economics

Firebase backend

$136

100,000 players / hype case

Agora Voice

$35.6k

100,000 players / hype case

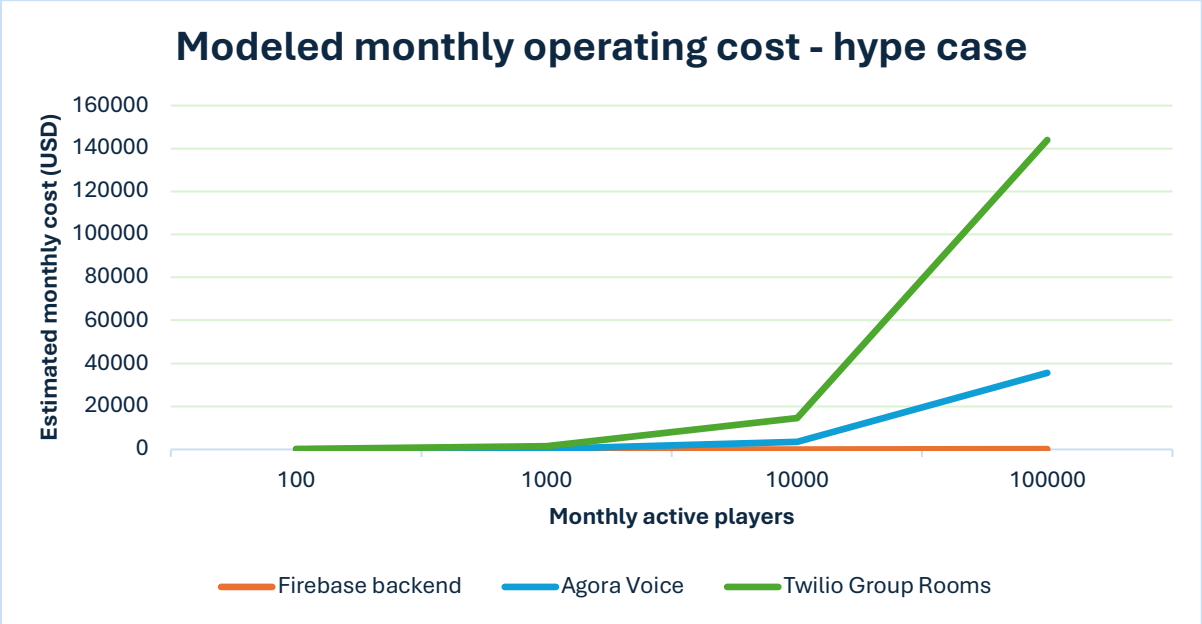
Twilio Group Rooms

$144k

100,000 players / hype case

Cost-growth view

Hype-case model shown below; stepped player tiers are used to compare cost growth across all three service categories.



Note: the 100-player Firebase point is effectively $0 because the benchmark remains within Firestore free-tier usage at that scale.

Benchmark model and management interpretation

Illustrative benchmark assumptions

- 4-8 players per match; model uses 6 average participants
- 50 questions per moderated match
- Normal case: 4 matches per user / month at 35 minutes
- Hype case: 8 matches per user / month at 45 minutes
- Firebase benchmark: about 2,500 reads, 300 writes, and 50 deletes per completed match
- Voice and RTC room benchmarks priced per participant minute using public list pricing
- Model excludes tax, negotiated discounts, premium add-ons, and unusual bandwidth spikes

Why the spread is so large

Firebase mainly scales with app transactions such as reads, writes, and deletes.
Voice APIs and RTC room services scale with connected participant time, which turns session duration into a recurring cost driver.
A successful user spike can therefore remain manageable on the gameplay layer while becoming expensive on the communication layer.

Twilio Group Rooms is included as a broader video + voice room benchmark rather than the intended MVP architecture. It serves as a deliberately conservative upper-cost market reference.

Decision framework

Monthly cost benchmark (USD)

Normal monthly usage benchmark

Monthly active players	Firebase	Agora Voice	Twilio Group Rooms
100	$0	$3.96	$56
1,000	$0.05	$128.70	$560
10,000	$5.81	$1,376.10	$5,600
100,000	$67.28	$13,850.10	$56,000

Hype monthly usage benchmark

Monthly active players	Firebase	Agora Voice	Twilio Group Rooms
100	$0	$25.74	$144
1,000	$0.55	$346.50	$1,440
10,000	$12.62	$3,554.10	$14,400
100,000	$135.62	$35,630.10	$144,000

Management recommendation

Project 50 should launch with Firebase-supported multiplayer gameplay only. Native in-app voice should remain out of scope until monetization, retention, or vendor pricing materially changes. What this document demonstrates: market comparison across multiple service categories, translation of architecture choices into operating-cost exposure, and disciplined scope reduction based on financial risk rather than technical preference.

Benchmark input sources: official public pricing pages for Google Cloud Firestore, Agora Voice Calling, and Twilio Video Group Rooms. Pricing should be revalidated before launch.

SCHMITT STUDIO

INFO@SCHMITT-STUDIO.COM

+61 493 673 358

Unreal Engine 5.5

Houdini

C++ / Blueprints

Lumen & Nanite

Feature Highlights

- **Survival Mechanics:** *Crafting, Building, Farming*
- **Exploration:** *Modular Boat Construction & Navigation*
- **Environment:** *Dynamic Weather & Open World*
- **Ecosystem:** *Complex AI & Animal Husbandry*

DEADOCEAN

Technical Foundation of DeadOcean

Description: DeadOcean is a high-fidelity open-world survival simulator developed with Unreal Engine 5.5 and Houdini. It combines procedural environment generation with advanced survival mechanics to create an immersive, persistent world.

Technical Pillars:

- **Procedural World Building:** Leveraging Houdini for realistic terrain, water simulation, and asset distribution.
- **High-End Rendering:** Utilizing Lumen for dynamic global illumination and Nanite for cinematic-quality geometry.
- **Native Steam Integration:** Built for PC with optimized performance for modern high-end hardware.

Advanced Survival Systems: Players must gather resources to construct custom shelters and manage sustainable life cycles through farming, fishing, and animal breeding.

Deep Crafting & Engineering: Features a modular boat-building system allowing players to traverse the open ocean and explore derelict ships.

Dynamic Ecology: A vast biological simulation featuring dozens of species, including predators (Sharks, Crocodiles), marine life (Whales, Dolphins), and terrestrial animals (Pigs, Rabbits, Cats).

Data Management: Custom-built inventory and data management system.

DEAD OCEAN

Executive Feasibility Snapshot

Production-scale assessment of procedural world-building investment, staged release sequencing, and flagship-scope delivery for a high-fidelity survival simulator

Product concept

DeadOcean is a high-fidelity open-world survival simulator built around Unreal Engine 5.5, Houdini-driven procedural world building, advanced survival systems, modular boat construction, dynamic weather, and a complex ecology spanning marine and terrestrial life. The concept phase tested whether early production should handcraft a small island set for a faster short-term result or invest that effort into Houdini and Houdini Engine to create a reusable world-building system.

Technical foundation

Unreal Engine 5.5 | Houdini Engine | C++ / Blueprints | Lumen & Nanite

Executive decision

Proceed with a Houdini-centered production pipeline, a single-player-first release on Steam, and multiplayer deferred to a later phase. Reason: DeadOcean's value depends on reusable world-production systems, not on spending one-off labor on a small handcrafted island set. The tooling complexity and Houdini license cost were accepted because this is the studio's flagship-scale project and by far the most production-intensive initiative in the portfolio.

Handcrafted island path

Curated terrain, dressing, and layout for a small set of islands. Assessment: faster to visualize once, but poor long-range scalability for a survival world that needs iteration and growth.

Houdini-centered pipeline

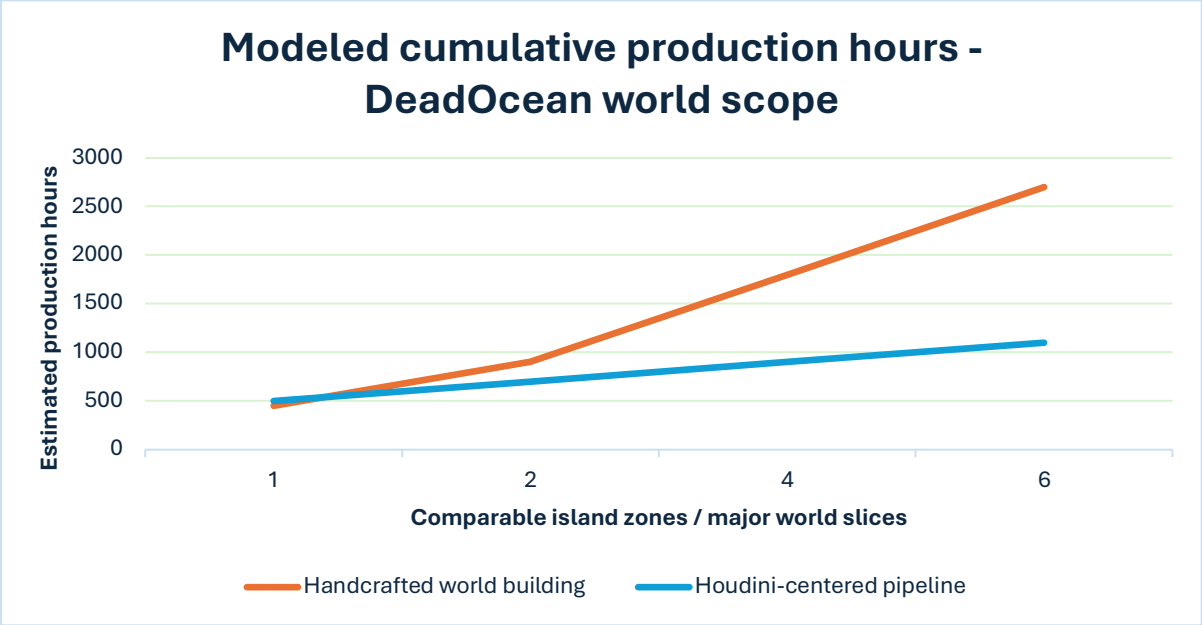
Procedural island generation, distribution logic, shoreline variation, and Unreal ingestion through Houdini Engine. Assessment: high upfront complexity, strongest long-term production leverage.

Accepted project posture

Houdini Indie: $299 annual license. Steam Direct: $100 one-time app fee. Interpretation: tool spend was accepted; the real economic issue is hundreds of hours of production labor.

Production leverage view

Illustrative hours model shown below; the chart compares one-off handcrafted island production with a Houdini-centered reusable pipeline as world scope expands.



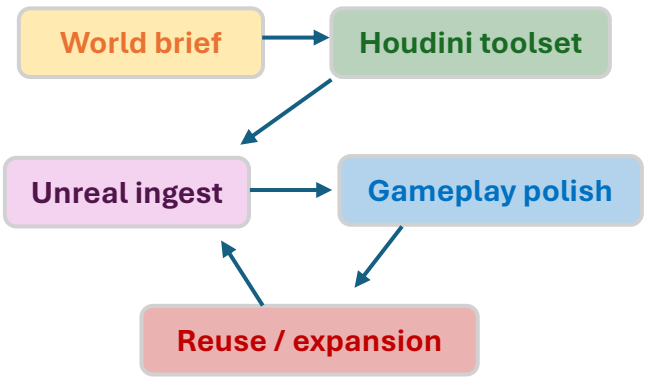
Note: the plotted curve is a management illustration. It uses a conservative production model rather than measured post-build telemetry. Actual effort will vary with biome count, quality bar, gameplay integration, optimization needs, and the maturity of the procedural toolchain.

Benchmark model and management interpretation

Illustrative benchmark assumptions

- Single-player release planning; multiplayer is intentionally excluded from launch build assumptions.
- Handcrafted benchmark: about 450 production hours per comparable island zone, including terrain shaping, scatter placement, traversal cleanup, and iteration.
- Procedural benchmark: about 500 hours initial investment for Houdini learning, generator design, Houdini Engine integration, debugging, and first production-ready setup.
- Additional comparable zone after pipeline setup: about 110 hours for parameter authoring, gameplay polish, and validation.
- Model excludes character animation, narrative content, audio, marketing assets, formal certification, and late-cycle QA spikes.

Why the leverage becomes decisive



The procedural route does not mainly save time at the first prototype. Its value appears once island creation, shoreline layout, distribution, and cleanup stop being repeated from scratch. Under this model, the same rough effort that would hand-build one small polished map can instead establish a generator that supports several comparable world slices over time.

Decision framework

Illustrative production benchmark, platform rationale, and risk interpretation for the studio's most ambitious project. Illustrative production benchmark (hours)

World scope	Handcrafted	Houdini pipeline	Hours avoided
1 island zone	450	500	-
2 island zones	900	610	290
4 island zones	1,800	830	970
6 island zones	2,700	1,050	1,650

Labor-value illustration

At a conservative $90/hour production-equivalent rate, the modeled 1,650-hour gap at six island zones corresponds to roughly $148,500 of repeated labor avoided.

Why Steam

Despite platform fees, Steam remains the primary commercial platform for PC indie game discovery and distribution. Steam Direct entry costs remain $100 per app, making it a viable launch path for a single-player indie release.

Risk and management interpretation

Risk area	Impact	Operational response	Expected outcome
Procedural tooling ramp	High	Front-load Houdini R&D, generator tests, and Unreal integration before full production rollout.	Reusable world systems instead of repeated one-off map work.
Content ambition / scope creep	High	Define a ship-ready single-player slice and cap initial biome count before expanding the ocean.	Ambition stays staged instead of overwhelming delivery.
Performance burden	High	Profile Nanite, Lumen, streaming, AI density, and water systems continuously during content scaling.	Technical fidelity remains aligned with playable performance.
Later multiplayer expansion	High	Defer replication, persistence, session logic, backend services, moderation, and support overhead to a later phase.	Launch risk is reduced while future online scope remains possible.

Management recommendation

DeadOcean should proceed as a Steam-bound single-player release built on a Houdini-centered content pipeline, with multiplayer held for a later expansion phase. The $299 annual Houdini Indie cost was acceptable because the decisive economic issue is not software price but whether hundreds of hours are spent on one small handcrafted map or on a reusable production system that can multiply output over time.

SOLDOUT!

Executive Feasibility Snapshot

Assessment of standard Unity tooling, custom retail-economy systems, and the deliberate rejection of Houdini-heavy workflows and early multiplayer scope

Technical foundation

- Unity Engine / C#
- ProBuilder + Polybrush
- Prefabs + Grid Snapping
- ScriptableObjects
- Unity UI for interfaces

Product concept

SoldOut! is a retail management and economic simulation built around goods flow rather than spectacle-first world generation. Its value sits in pallets, cartons, unpacking, shelf distribution, warehouse handling, wholesale purchasing, occasional small-lot purchasing, dynamic customer demand, and a simulated market layer that reacts to supply shocks, weather events, factory fires, overproduction, and other pricing events.

Executive decision

Proceed with a standard Unity workflow, a custom data-driven warehouse and economy layer, and a single-player-first release on Steam. Keep Houdini-heavy workflows and early multiplayer out of the current scope. Defer advanced inventory rules, supplier behavior, logistics loops, and multiplayer economy systems to later phases.

Reason: this genre's leverage comes less from procedural geometry and more from reliable simulation design. Houdini would add setup burden without proportional return, while multiplayer would materially expand implementation risk and ship time.

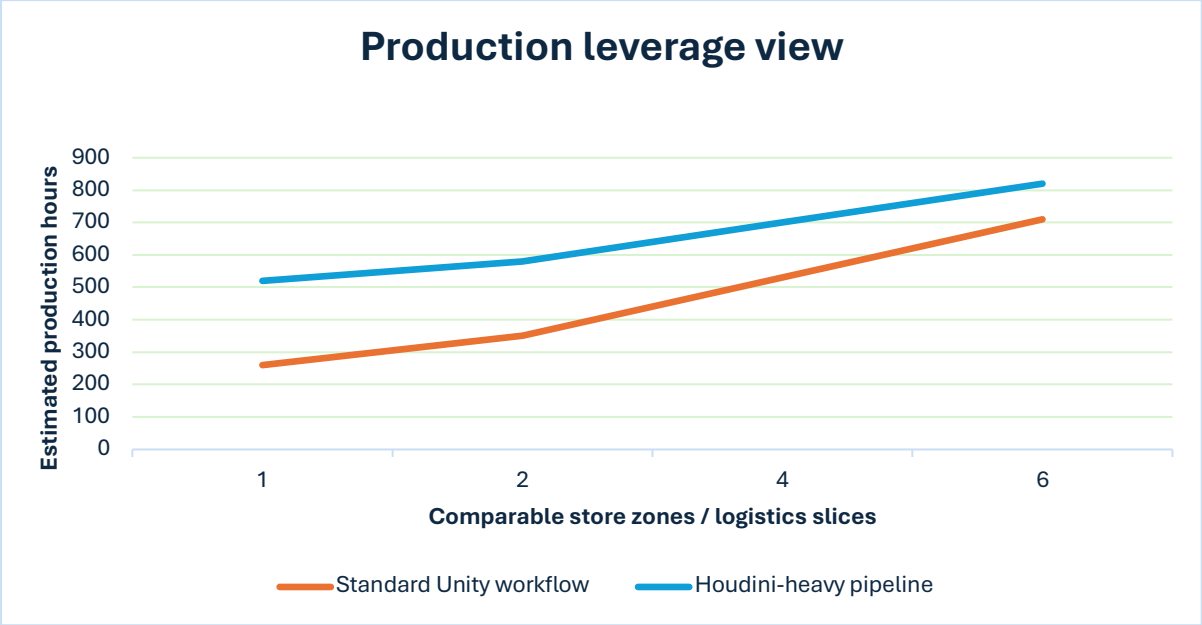
Standard Unity workflow
ProBuilder, Polybrush, Prefabs, grid-based layout, and ScriptableObject - driven data structures. Assessment: right-sized production stack for this type of environment.

Custom retail simulation core
Pallets, cartons, product units, supplier channels, customer demand, price reactions, and event-driven market shifts. Assessment: highest-value system investment.

Deferred multiplayer
Management benchmark: about 2.4x total implementation effort for a comparable ship-ready MVP. Assessment: not justified during the current stage.

Production leverage view

Illustrative hours model shown below; the chart compares a standard Unity workflow with a Houdini-heavy procedural workflow as store and logistics scope expands.



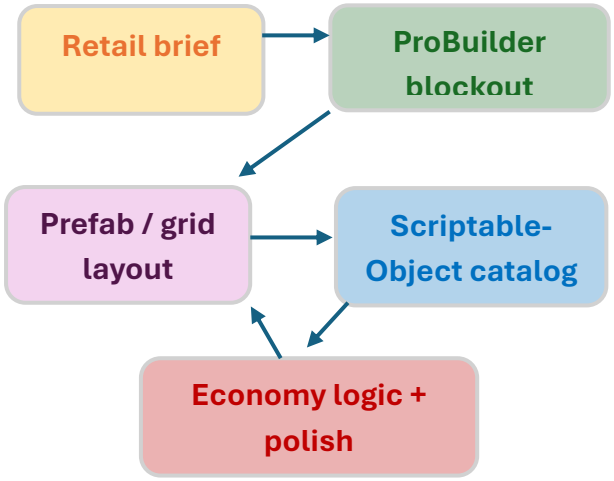
Note: the plotted curve is a management illustration. It uses a conservative production model rather than measured post-build telemetry. Actual effort will vary with art direction, simulation depth, optimization needs, and the amount of custom editor tooling that becomes necessary later.

Benchmark model and management interpretation

Illustrative benchmark assumptions

- Single-player release planning; multiplayer stays out of the baseline model.
- Standard Unity benchmark: about 260 hours for a first production-ready store slice including blockout, shelf prefabs, grid setup, product schema, and warehouse interactions.
- Additional comparable slice after setup: about 90 hours.
- Houdini benchmark: about 520 hours initial investment; additional comparable slice after setup: about 60 hours.
- Model excludes certification, marketing assets, voice content, and late-cycle QA spikes.

Why the leverage stays with standard Unity



For SoldOut!, the real burden sits in the simulation layer: stock states, supplier logic, price reactions, demand curves, warehouse throughput, and the link between sourcing and shelf availability. Because the playable space is mostly simple architecture and repeated fixtures, the project does not gain enough from a procedural toolchain to justify its setup cost.

Decision framework

Illustrative production benchmark, multiplayer burden estimate, and management interpretation for a single-player-first retail economy simulator. Illustrative production benchmark (hours)

Store/ logistics scope	Standard Unity workflow	Houdini-heavy pipeline	Hours avoided by Unity
1 logistic slice	260	520	260
2 logistic slices	350	580	230
4 logistic slices	530	700	170
6 logistic slices	710	820	110

Labor-value illustration

At a modeled 1,100-hour single-player MVP, a 2.4x multiplayer path implies about 2,640 hours total – roughly 1,540 extra hours.

Multiplayer expansion

2.4x total MVP effort vs. comparable single-player build.
Interpretation: the realistic burden band is about 2x-3x once synchronized simulation, authority, persistence, QA, and live support are included.

Risk and management interpretation

Risk area	Impact	Operational response	Expected outcome
Economy-system complexity	High	Lock product, event, and stock schemas early.	Depth grows where value sits.
Houdini overengineering	Medium	Keep layout in standard Unity; add tools only for real bottlenecks.	Lower setup and debugging burden.
Scope creep in realism	High	Cap initial store size, supplier types, and event count.	Ambition stays staged and shippable.
Early multiplayer burden	High	Defer authority, persistence, anti-cheat, and live ops.	Core economy loop ships sooner.
Performance / entity count	High	Profile shelf density, shopper AI, and UI refresh loops.	Stable simulation at scale.

Management recommendation

SoldOut! should proceed as a Steam-bound single-player retail economy simulator built in Unity with ProBuilder, Prefabs, grid-based layout discipline, ScriptableObject-driven product data, and a purpose-built warehouse / economy system. Houdini should remain out of scope because the geometry does not justify the tooling investment, and multiplayer should remain a later phase because the additional burden is better estimated in the 2x-3x range than the 1.5x range for this kind of synchronized simulation.

SCHMITT STUDIO

INFO@SCHMITT-STUDIO.COM

+61 493 673 358

SwiftUI *for the user interface*
SwiftData *for local model persistence*
Firebase Auth *for secure sign-in*
Cloud Firestore *for synced match data*
Head-to-head stats and leaderboards

Technical Foundation of RivalStats

RivalStats is a fully native iOS app built in Swift and SwiftUI for structured match tracking, rivalry analysis, and account-based stat management on Apple devices. The app is designed to help users create players and games, record sessions, compare results, and review competitive performance in one streamlined workflow.

RivalStats supports custom game setups with different scoring models, reusable friend profiles, and searchable libraries for games and sessions. Users can track results over time, inspect detailed match history, and analyze rankings through automated statistics such as wins, draws, losses, win rate, points scored and conceded.

Built with modern Apple technologies and Firebase services, RivalStats combines a polished native interface with secure sign-in and cloud-backed data storage. Local models are managed efficiently on-device, while account-specific records for friends, games, and sessions can be synchronized and used for features such as head-to-head comparisons, leaderboards, and long-term rivalry tracking.

RivalStats

Executive Feasibility Snapshot

Assessment of Firebase-backed account architecture, cross-device rivalry history, and the transition from deferred online scope to justified product core

Product concept

RivalStats represents the broader online direction that was deliberately rejected in narrower products such as FlipStats and WorldFrame. Instead of adding cloud scope to a single-game utility, this concept makes persistent online history the core product value: tracking games, friends, sessions, rankings, and head-to-head performance over time.
The current iOS codebase combines SwiftUI and SwiftData with Firebase Authentication and Cloud Firestore collections for per-user friends, games, and sessions.

Executive decision

Proceed with Firebase-backed account architecture, per-user cloud persistence, and local on-device caching. Keep public social features and live collaborative match rooms out of the initial release to preserve a controlled v1 scope. Reason: RivalStats gains real value from persistent identity, reusable friend
libraries, cross-game history, and long-term rivalry analysis.

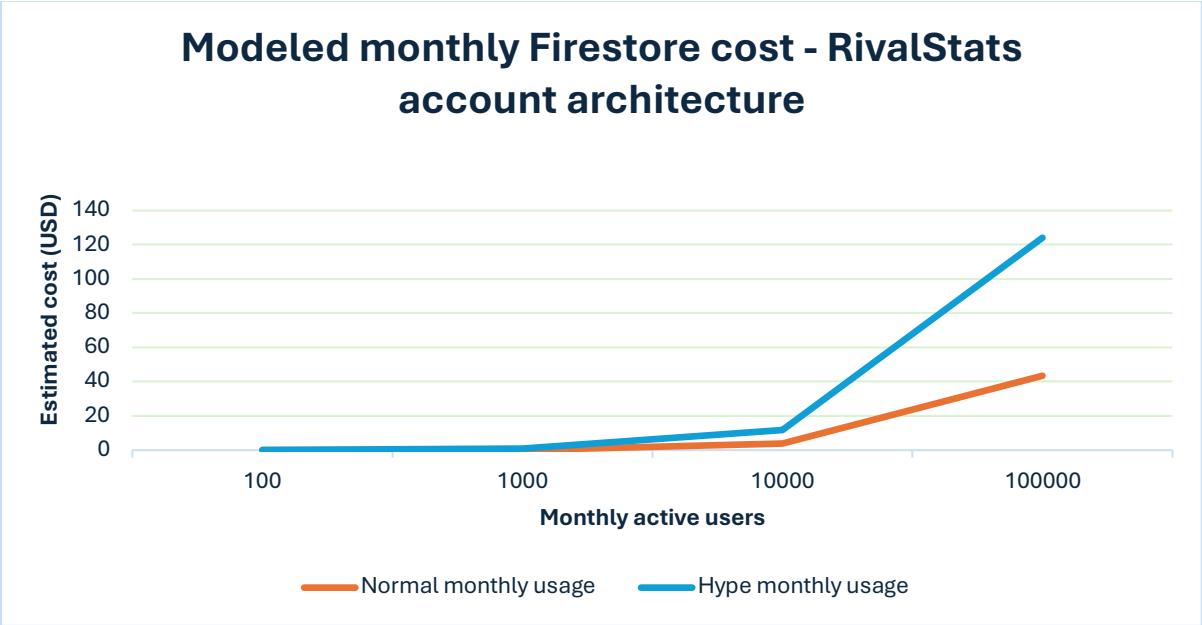
Local-only tracker
Single-device history and no sign-in. Assessment: faster to ship, but underpowered for the broader rivalry concept.

Firebase account architecture
Email sign-in, cloud-backed libraries, SwiftData caching, and cumulative rivalry analytics. Assessment: right-sized online scope for v1.

Upper stress test
$124.11 modeled monthly Firestore cost at 100,000 monthly active users in the hype case.

Cost-growth view

Current Firestore benchmark shown below; the chart models the monthly document-operation cost of RivalStats' account-based sync layer under normal and heavier usage patterns. Read volume grows mainly with collection size and how often users reopen the app.



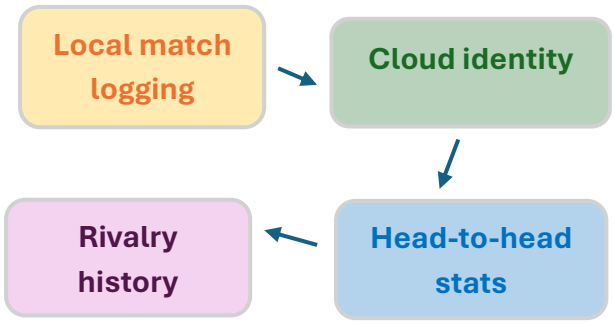
Note: management illustration based on simplified Cloud Firestore operations. The model excludes storage, bandwidth, negotiated discounts, Firebase Authentication overage beyond the no-cost tier, and unusual listener churn.

Benchmark model and management interpretation

Illustrative assumptions

- Signed-in single-user model; no shared public profiles or live multi-user rooms in v1.
- Normal case: 12 active app days per month; about 15 friends, 8 games, and 90 synced sessions; about 18 new sessions logged.
- Hype case: 20 active app days; about 24 friends, 10 games, and 160 synced sessions; about 40 new sessions logged.
- Benchmark uses current public Firestore document pricing plus free-tier quotas of 50k reads/day, 20k writes/day, and 20k deletes/day.

Why online scope fits



Cloud sync keeps friend libraries, game definitions, and cumulative sessions stable across devices. It also powers rankings and head-to-head summaries without requiring live multiplayer infrastructure. Here, Firebase supports the core product.

Decision framework

Monthly cost benchmark, scope-risk interpretation, and management recommendation.

Normal benchmark

Monthly active users	Firestore
100	$0
1.000	$0
10.000	$3.80
100.000	$43.47

Hype benchmark

Monthly active users	Firestore
100	$0
1.000	$0.76
10.000	$11.61
100.000	$124.11

Annualized Firestore view at 100,000 monthly active users: about $521.64 per year in the normal benchmark and about $1,489.32 per year in the hype benchmark, before storage and transfer costs. Firebase Authentication is not included in the table.

Risk and management interpretation

Risk area	Management interpretation
Product scope inflation	Keep v1 centered on private account data, friend and game libraries, sessions, rankings, and head-to-head stats. Result: clearer identity and controlled scope.
Read-volume growth from history sync	Use lean documents, practical retention, and SwiftData caching for responsive UI. Result: useful history with manageable cloud cost.
Auth + privacy responsibility	Use Firebase Auth, per-user partitioning, account deletion, and clear privacy handling. Result: lower privacy and data-governance risk.
Future social / public expansion	Defer public leaderboards, invites, and live multi-user features. Result: the first release keeps the right online depth.

Management recommendation

RivalStats should proceed as an account-based iOS app built on SwiftUI, SwiftData, Firebase Authentication, and Cloud Firestore. In this product, online scope is justified because persistent identity, reusable friend and game libraries, and long-term rivalry history are core to the concept rather than optional decoration.
The recommended posture is cloud-backed private stats first, with broader social, sharing, or live network features deferred until user demand and product maturity support them.

SCHMITT STUDIO

INFO@SCHMITT-STUDIO.COM

+61 493 673 358

SwiftUI *for the user interface*

Live score tracking *for Flip7 games*

Player management and reusable club members

Saved game history and member statistics

Local on-device data storage *for privacy-focused use*

Technical Foundation of FlipStats

FlipStats is a fully native iOS app built in Swift and SwiftUI for fast, simple and reliable score tracking on Apple devices. The app is designed to help users create players, run Flip7 games, enter round results, and follow live rankings without manual calculation.

FlipStats supports multiplayer score tracking across multiple rounds, including automatic score calculation, Flip7 bonus handling, modifier card support, and instant leaderboard updates. Users can save completed games to history and review club-based player statistics such as games played, wins, win rate, best score, and Flip7 counts.

Built with Apple-native technologies, FlipStats is lightweight, maintainable, and optimized for a smooth on-device experience. Player data, game progress, saved history, and app settings are stored locally on the device, supporting a straightforward and privacy-conscious workflow without unnecessary complexity.

FlipStats
Executive Feasibility Snapshot

Scope and cost assessment of a Firebase-backed online expansion for a niche Flip 7 live score-tracking app

Product concept

FlipStats is a focused companion app for Flip 7.
The product is designed for live hand entry, automatic score calculation, running totals, and lightweight history so a table can move through the game faster without manual arithmetic.
The concept phase evaluated whether the product should remain a local-first single-game utility or include a Firebase-backed online layer with cloud sync, shared history, and persistent player data. Prepared as an internal concept evaluation for product scope discipline, implementation effort, and backend cost exposure.

Executive decision

Launch FlipStats as a local-first utility and keep Firebase-backed online scope out of the initial product. Reason: the modeled Firestore cost remains commercially manageable, but the online layer would materially expand implementation time, account / sync complexity, and product surface area for an app that is expected to remain a highly specific single-game tool.

Local-first app

Live round entry, automatic totals, and on-device game history. Assessment: right-sized for MVP.

Firebase online extension

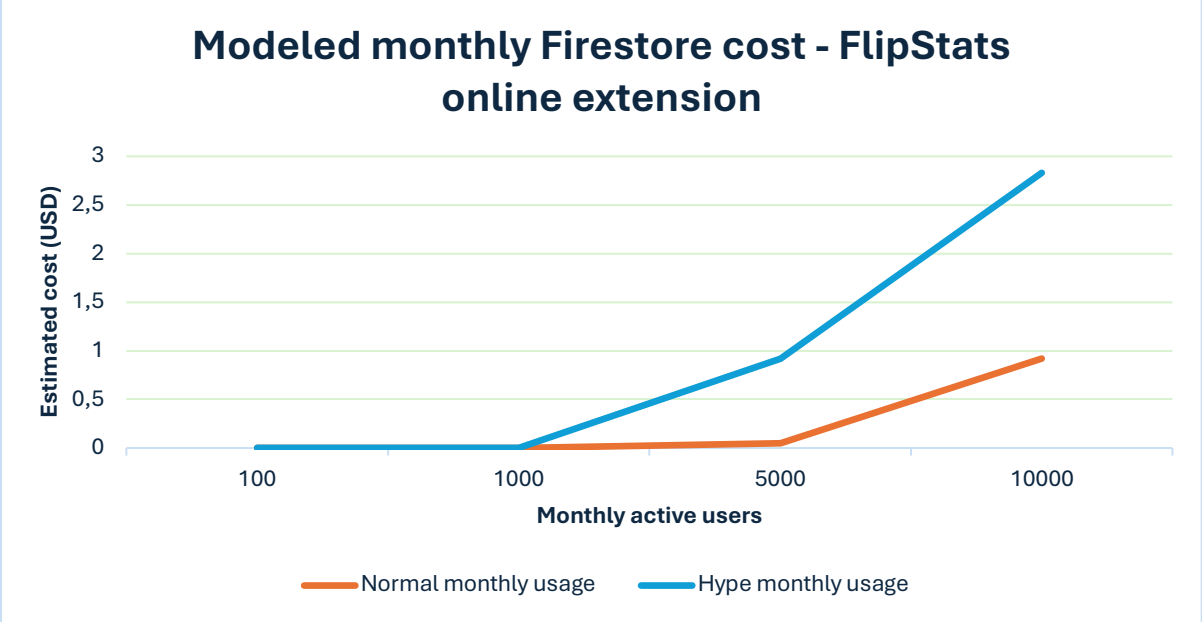
Cloud sync, shared history, and persistent player data. Assessment: financially viable, strategically oversized.

Upper stress test

$2.83 modeled monthly Firestore cost at 10,000 monthly active users in the hype case. Interpretation: cost was not the decisive blocker.

Cost-growth view

Niche-oriented demand model shown below; the chart uses a compact Firestore benchmark and assumes usage is distributed broadly enough across the month for free-tier quotas to offset smaller traffic levels.



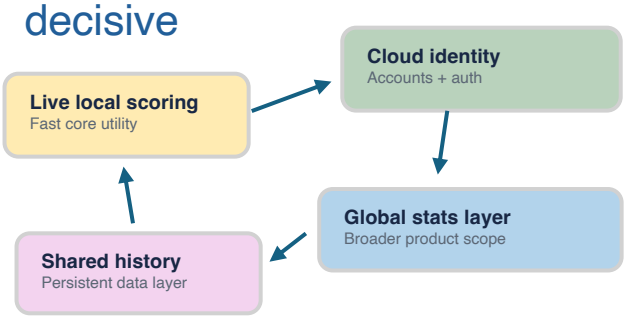
Note: the plotted curve is a management illustration. It uses a simplified Cloud Firestore operating model and excludes storage, bandwidth, tax, negotiated discounts, and unusual real-time listener patterns. Actual spend will vary with document structure, security rules, sync frequency, and traffic concentration.

Benchmark model and management interpretation

Illustrative benchmark assumptions

- 2-6 players per game; model uses 4 average players
- 1 completed game averages 8 rounds until a player reaches 200 points
- Normal case: 6 completed games per user / month
- Hype case: 12 completed games per user / month
- Compact Firestore benchmark: about 40 reads, 22 writes, and 2 deletes per completed game
- Monthly estimate applies current default Firestore rates beyond free quota and approximates free-tier usage as a 30-day month
- Model excludes storage, bandwidth, tax, negotiated discounts, and unusual listener spikes

Why the leverage becomes decisive



Why this matters: even low backend cost would have turned a fast single-game utility into a broader online product. The key decision factor was scope discipline: avoid adding account systems, sync states, and support overhead to a niche Flip 7 companion app when those features were not essential to its core job.

Decision framework

Monthly cost benchmark, scope-risk interpretation, and later portfolio context.

Monthly cost benchmark (USD)

Normal monthly usage benchmark

Monthly active users	Firestore
100	$0
1,000	$0
5,000	$0.05
10,000	$0.92

Hype monthly usage benchmark

Monthly active users	Firestore
100	$0
1,000	$0
5,000	$0.92
10,000	$2.83

Risk and management interpretation

Risk area	Impact	Operational response	Expected outcome
Product scope inflation	High	Keep FlipStats focused on live score entry, automatic calculation, and on-device history.	Faster delivery and clearer product identity.
Implementation time expansion	High	Avoid introducing auth, cloud schema, sync logic, security rules, and cross-device QA into the initial release.	Lower build complexity and quicker path to a usable app.
Operational overhead	Medium	Defer cloud monitoring, support, and data maintenance until a broader use case justifies them.	Reduced maintenance burden for a niche utility.
Market-size mismatch	High	Match feature depth to realistic audience size for a single-card-game companion app.	Product stays proportional to likely demand.

Management recommendation

FlipStats should remain a focused local-first Flip 7 tool. A Firebase-backed online layer is financially feasible, but it should not be added merely because it is technically possible. For this product, right-sized scope and shorter implementation time create more value than expanding into a cloud feature set with limited strategic upside.

The deferred Firebase-backed online direction was later transferred into a broader concept, Project Global Stats, which eventually became the app RivalStats. In that context, global online tracking across board games, online games, and real-life games against friends provided a much stronger reason to invest in cloud identity, shared history, and broader stat infrastructure. What this demonstrates: disciplined feature reduction based on product fit and delivery focus, while preserving a transferable backend concept for a broader future product. Benchmark input sources: official public pricing pages for Google Cloud Firestore and Firebase pricing documentation.

SCHMITT
STUDIO

INFO@SCHMITT-STUDIO.COM

+61 493 673 358

SwiftUI *for the user interface*

Codable / JSONDecoder / JSONEncoder *for structured data handling*

AVFoundation *for music and sound integration*

Asset Catalogs *for visual resource management*

Native support *for Light and Dark Mode*

Technical Foundation of WorldFrame

WorldFrame is a fully native iOS app designed and developed in Swift 5 with SwiftUI, combining modern Apple technologies with a clean, data-driven architecture.

The app's gameplay systems, level logic, content structure, and legal information are managed through locally integrated JSON files, enabling a lightweight, scalable, and maintainable application design. Key resources include structured datasets for country attributes, gameplay criteria, progression logic, and legal content.

To ensure performance, consistency, and seamless platform integration, WorldFrame relies exclusively on Apple-native frameworks for data handling, audio integration, visual resource management, and native Light and Dark Mode support.

WorldFrame is intentionally built without external dependencies, resulting in a streamlined and platform-optimized user experience.

WorldFrame

Executive Feasibility Snapshot

Assessment of offline-first quiz scope, local JSON content architecture, Apple-native competitive features, and the deliberate rejection of a Firebase-backed online game layer

Product concept

WorldFrame is a fully native iOS country-knowledge quiz / puzzle game built around local structured datasets instead of a server-dependent gameplay model. The current SwiftUI codebase manages country attributes, criteria logic, progression, localization, audio feedback, legal information, and save behavior through Apple-native frameworks and local JSON content.

The project reflects a meaningful content base — 195 countries, 81 criteria entries, and a 75-level structure — while remaining focused on offline-first execution. Because WorldFrame was the studio's first shipped app, the first release intentionally avoided custom backend, account, and live-service complexity.

Executive decision

Launch WorldFrame as an offline-first iOS quiz game built around local JSON content, on-device progression, Apple-native architecture, Game Center competition, and lightweight iCloud-backed progress persistence. Keep Firebase-backed online play out of scope.
Reason: even with manageable backend cost, online mode would still have expanded delivery risk through authentication, live sync, security rules, room handling, and operational support. For a first shipped app, gameplay execution and native polish had higher priority.

Local-first app

Local country datasets, criteria logic, level progression, audio feedback, legal content, Game Center competition, and on-device save behavior.
Assessment: right-sized for a first shipped app.

Rejected Firebase online mode

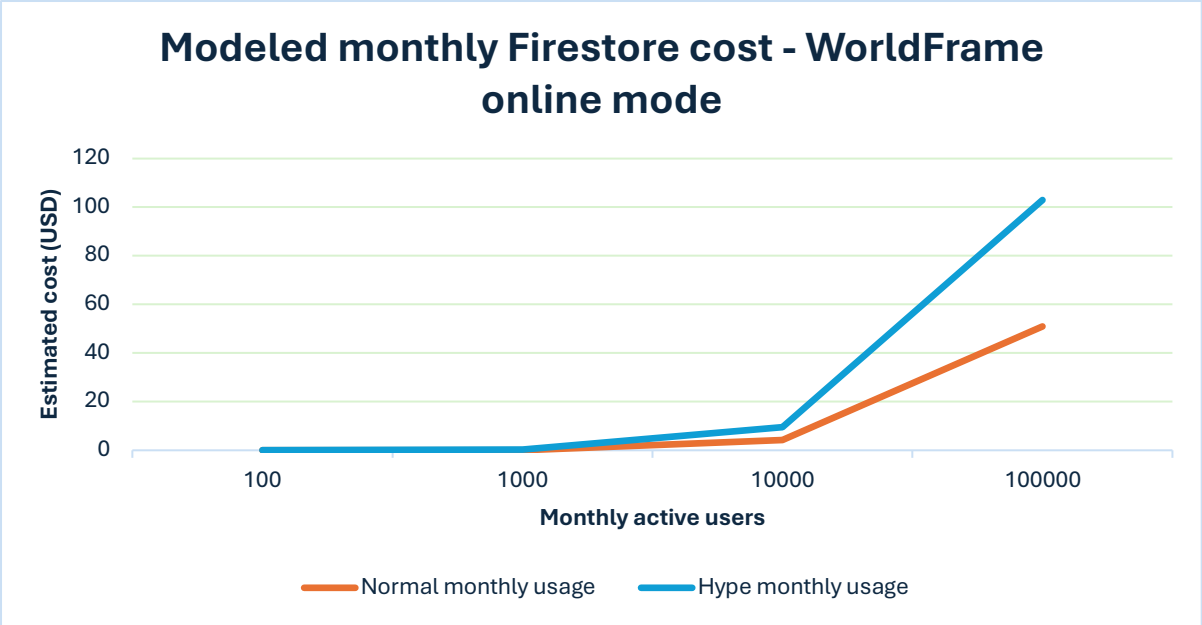
1v1 synchronized country-grid sessions, room state, answer sync, scoring, and cleanup.
Assessment: technically feasible, but strategically premature for v1.

Upper stress test

$102.95 modeled monthly Firestore cost at 100,000 monthly active users in the rejected online-mode hype case. Interpretation: cost was manageable, but backend complexity was not justified.

Cost-growth view

Niche-oriented demand model shown below; the chart uses a compact Firestore benchmark and assumes usage is distributed broadly enough across the month for free-tier quotas to offset smaller traffic levels.



Note: the plotted curve is a management illustration. It uses a simplified Firestore operating model and excludes Firebase Auth edge cases, storage, bandwidth, moderation, tax, and unusual realtime listener spikes. Actual spend would vary with room design, sync frequency, security rules, and traffic concentration.

Benchmark model and management interpretation

Illustrative benchmark assumptions

- Illustrative online concept: 1v1 synchronized country-grid sessions.
- One completed online match modeled as about 500 reads, 120 writes, and 20 deletes across room creation, answer sync, scoring, and cleanup.
- Normal case: 4 online matches per user / month; hype case: 8 online matches per user / month.
- Firestore pricing model uses current public reads / writes / deletes pricing beyond free tier and approximates free usage across a 30-day month.

Why the backend path was premature

For WorldFrame, the online question was not purely a hosting-cost question. A live mode would have introduced account flows, synchronization conflicts, match lifecycle handling, room cleanup, and production debugging into the studio's first shipped app. The chosen architecture instead focused effort on country logic, criteria design, level pacing, localization, audio feedback, polished native UI, and Apple-native competitive features. Game Center preserved lightweight competition value without forcing a custom backend from day one.

Decision framework

Monthly cost benchmark, first-app scope interpretation, and management recommendation.

Monthly cost benchmark (USD)

Normal monthly usage benchmark

Monthly active users	Firestore
100	$0
1,000	$0
10,000	$4.17
100,000	$50.95

Hype monthly usage benchmark

Monthly active users	Firestore
100	$0
1,000	$0.15
10,000	$9.35
100,000	$102.95

Annualized view at 100,000 monthly active users: about $611.40 per year in the normal benchmark and about $1,235.40 per year in the hype benchmark, before bandwidth and operational add-ons. Apple Developer Program membership remains $99 per year by comparison.

Risk and management interpretation

Risk area	Management interpretation
First-app backend complexity	Avoid coupling the first release to auth, room sync, conflict resolution, and live incident handling. Result: cleaner execution path and lower delivery risk.
Product scope inflation	Keep WorldFrame centered on level design, country data, progression, and quiz polish. Result: clearer product identity and faster delivery.
Live-service operational overhead	Defer moderation, monitoring, backend analytics, and abuse tooling until broader demand justifies them. Result: lower maintenance burden.
Competitive feature balance	Use Game Center for leaderboard and achievement features instead of building full multiplayer infrastructure. Result: lightweight competitive value without backend lock-in.

Management recommendation

WorldFrame should remain an offline-first iOS quiz game built on Apple-native frameworks, structured local JSON content, Game Center, and lightweight iCloud-backed progress persistence. Firebase-backed online play should remain outside the current product scope because execution maturity, quiz polish, and scope discipline matter more than backend reach for a first shipped app.

The rejected Firebase-backed online direction was later transferred into a broader concept, Project Global Stats, which eventually became the app RivalStats. In that context, global online tracking across board games, online games, and real-life games against friends provided a much stronger reason to invest in cloud identity, shared history, and broader stat infrastructure.
What this demonstrates: disciplined feature reduction based on product fit and delivery focus, while preserving a transferable backend concept for a broader future product.
Benchmark input sources: official public pricing pages for Google Cloud Firestore and Firebase pricing documentation.

SCHMITT STUDIO

INFO@SCHMITT-STUDIO.COM

+61 493 673 358

SwiftUI *for the user interface*

PhotosUI *for image selection*

Batch image conversion workflows

Export size reduction through format and resize settings

Optional metadata removal for privacy-focused exports

Technical Foundation of PixConvert

PixConvert is a fully native iOS app built in Swift and SwiftUI for fast, private and reliable image processing on Apple devices. The app is designed to convert image files, reduce storage usage, and remove metadata without unnecessary complexity.

PixConvert supports batch-based image conversion and export workflows, allowing users to process multiple files in one pass. Users can convert between common image formats, apply predefined size reductions or custom pixel dimensions, and generate smaller output files when needed.

Built with Apple-native technologies, PixConvert is lightweight, maintainable, and optimized for a streamlined on-device experience.

SCHMITT STUDIO

INFO@SCHMITT-STUDIO.COM

+61 493 673 358

SwiftUI *for the user interface*

Native file selection and Drag & Drop *for easy image import*

Batch image conversion workflows

Export size reduction through format and resize settings

Optional metadata removal for privacy-focused exports

Technical Foundation of PixConvert for Mac

PixConvert for Mac is a fully native macOS app built in Swift and SwiftUI for fast, private, and reliable image processing on Apple computers. The app is designed to convert image files, reduce storage usage, and remove metadata without unnecessary complexity.

PixConvert for Mac supports batch-based image conversion and export workflows, allowing users to process multiple files in one pass. Users can convert between common image formats, apply predefined size reductions or custom pixel dimensions, and generate smaller output files when needed.

Built with Apple-native technologies, PixConvert for Mac is lightweight, maintainable, and optimized for a streamlined desktop experience.

PixConvert

Executive Feasibility Snapshot

Assessment of native two-app scope, offline image conversion, metadata privacy controls, and the deliberate exclusion of server-based video conversion

Product concept

PixConvert and PixConvert for Mac were created to solve a practical Apple-platform workflow gap: a fast, trustworthy way to convert images without sending personal files to dubious websites or adding unnecessary workflow friction.
Both apps convert images locally, support batch processing, reduce file size through format and resize settings, and offer optional metadata removal. The concept phase also tested whether video conversion or a more complicated combined macOS path truly fit the product.

Executive decision

Proceed with two separate native apps — one for iOS and one for macOS — with aligned purpose but platform-specific interaction models. Keep the product strictly offline and image-focused. Exclude server-based video conversion from the current scope.
Reason: building a separate macOS app was operationally cleaner than forcing desktop behavior into the existing iOS codebase, while a cloud video layer would weaken the privacy-first product promise, materially expand technical scope, and introduce recurring operating cost without proportional user value.

Separate native apps

iOS and macOS remain separate so each platform keeps its own natural import and export flow.
Assessment: lower integration complexity.

On-device image workflow

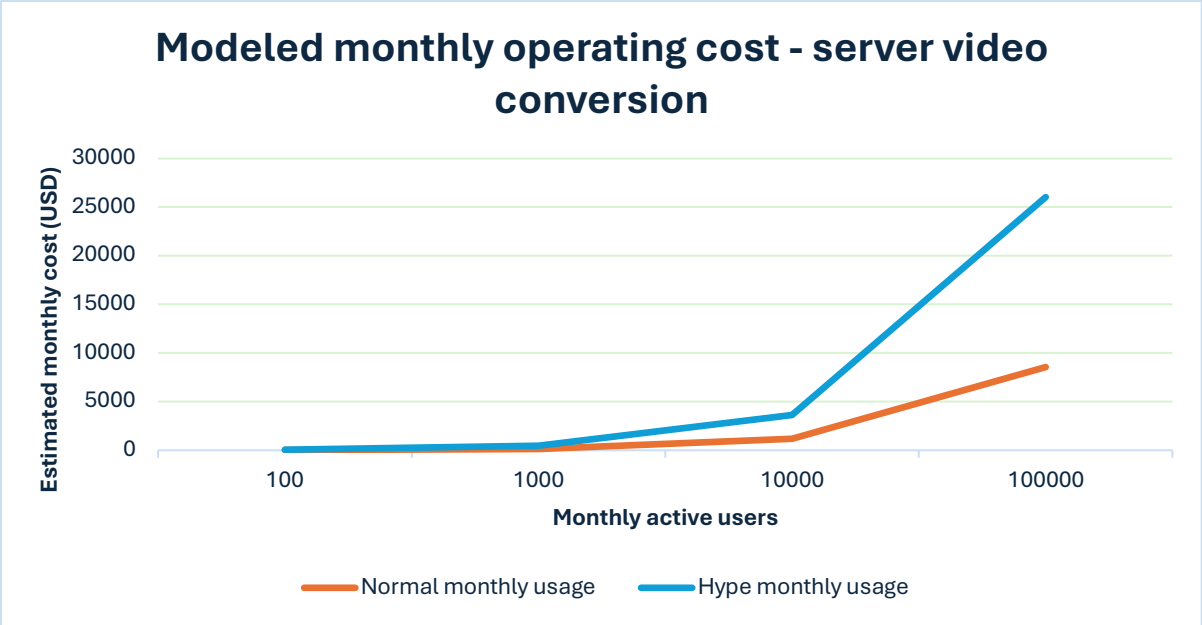
Images stay on device, and metadata can be stripped during export.
Assessment: matches the product's privacy purpose.

Rejected cloud video layer

About $26,040 modeled monthly transcoding-only cost at 100,000 monthly active users in the hype case.
Interpretation: recurring server burden without equal upside.

Cost-growth view

Server-based HD video conversion benchmark; the chart illustrates the recurring monthly cost that would have been introduced by adding cloud video processing to an otherwise offline utility.



Note: the plotted curve is a management illustration. It uses a simplified HD transcoding model and excludes storage, bandwidth, taxes, negotiated discounts, abuse tooling, and support overhead. Actual spend would vary with job length, traffic concentration, and chosen vendor architecture.

Benchmark model and management interpretation

Illustrative benchmark assumptions

- Illustrative vendor baseline: AWS Elemental MediaConvert Basic tier equivalent, modeled from current public HD output pricing.
- Benchmark reflects one 1080p HD output per conversion job.
- Normal case: 3 video exports per user / month at 3 minutes average = 9 output minutes.
- Hype case: 8 video exports per user / month at 4 minutes average = 32 output minutes.
- Transcoding-only view: storage, upload / download bandwidth, auth, abuse controls, support, and tax remain excluded.

Why the leverage becomes decisive

Image conversion on device mostly scales with local hardware and does not create a direct per-use server bill.
Server video conversion scales with processed output minutes, which turns growth in active users into recurring infrastructure spend.
The same feature would also require users to upload private media, wait for queuing and processing, and trust a cloud handling path that the product was intentionally designed to avoid.

Decision framework

Monthly server-cost benchmark, privacy / scope-risk interpretation, and management recommendation.

Monthly cost benchmark (USD)

Normal monthly usage benchmark **Hype monthly usage benchmark**

Monthly active users	Cloud transcoding cost	Monthly active users	Cloud transcoding cost
100	$13.50	**100**	$48
1,000	$135	**1,000**	$480
10,000	$1,174	**10,000**	$3,612
100,000	$8,560	**100,000**	$26,040

Annualized view at 100,000 monthly active users: about $102,720 per year in the normal benchmark and about $312,480 per year in the hype benchmark, before storage and transfer costs. Apple Developer Program membership remains $99 per year by comparison.

Risk and management interpretation

Risk area	Impact	Operational response	Expected outcome
User media privacy	High	Keep conversion local, avoid central media retention, and preserve optional metadata removal for export.	Lower data-exposure risk and a clearer trust story.
Product scope inflation	High	Keep PixConvert focused on image conversion rather than broadening into a heavier media suite.	Faster delivery and a more maintainable product.
Operating-cost volatility	Medium	Exclude cloud video processing from the current scope and keep infrastructure optional rather than foundational.	Near-zero recurring media-processing cost.
Cross-platform complexity	High	Use two native apps with aligned intent instead of forcing macOS behavior into the iOS code path.	Cleaner UX and fewer platform compromises.

Management recommendation

PixConvert should remain a privacy-first, offline image utility across iOS and macOS. The two-app structure is justified by lower engineering friction and better platform fit, while server-based video conversion should remain out of scope unless a future version gains clear monetization and a privacy model that users explicitly accept.
What this demonstrates: disciplined platform scoping, privacy-aware feature design, and cost-aware rejection of server dependency even when a feature is technically feasible.

Document basis: PixConvert iOS and macOS technical briefs plus project source files. Video-cost benchmark uses public AWS Elemental MediaConvert pricing; Apple distribution context was cross-checked against Apple Developer Program pricing.