QVAC SDK — Swift Client

Title	QVAC SDK — Swift Client
Publish Date	Apr 27, 2026
Submission End Date	Jun 24, 2026
Reward	3’000 USDt

Problem Summary

What is the problem, deficiency, or gap which will be addressed by this project?

The QVAC SDK (`@qvac/sdk`) currently exposes a single JavaScript client that supports Node.js, Bare, and Expo (Hermes) through runtime-specific IPC bridges selected via dynamic imports. The client communicates with a Bare worker process over an RPC protocol — the SDK spawns the Bare worker, establishes the RPC server, and the client connects as a consumer.

There is no native Swift client. iOS and macOS applications that want to use QVAC must go through the Expo/React Native bridge, adding an entire JavaScript layer and its associated overhead. A native Swift client that speaks directly to the Bare worker over IPC would allow Swift applications to use QVAC with native type safety, lower latency, and no JavaScript runtime dependency at the call site.

This grant funds the implementation of a **Swift client** for the QVAC SDK. The client is **not** a rewrite of the SDK — the Bare worker remains responsible for spawning, model management, and addon orchestration. The Swift client implements only the RPC consumer side: IPC connectivity to the Bare worker and a type-safe Swift API surface that mirrors the existing JavaScript client.

Impact / Reason Why

What is the metric or goal this bounty aims to improve?

- **Native performance** — Eliminates the React Native / Hermes bridge for iOS and macOS apps, reducing IPC overhead and memory footprint.
- **Developer experience** — Swift developers get an idiomatic, type-safe API (`async`/`await`, `AsyncSequence` for streaming) without needing to understand JavaScript internals.
- **Ecosystem growth** — Extends QVAC to the native Apple developer ecosystem, covering iOS, macOS, watchOS, and visionOS targets.
- **Consistency** — Code-generated from the JavaScript client, ensuring the Swift API surface stays in sync as the SDK evolves.

Scope Items

What important items are in scope for the project?

- **RPC client implementation** — Connect to the Bare worker's RPC server over IPC (Unix domain sockets on macOS/Linux). Implement the full request/response and streaming protocol that the JavaScript client already uses, including `__init_config` initialization, request multiplexing, and graceful shutdown.
- **Code generation tooling** — A code-gen pipeline that reads the JavaScript client's type definitions and RPC message schemas and produces Swift source files. The generator must produce all request/response types, API method signatures, and serialization logic. This ensures the Swift client stays in sync with the JS client as the SDK evolves, with type safety enforced at generation time.
- **Swift API surface** — A public Swift module (`QVACClient`) mirroring the JavaScript client's API. At minimum: `loadModel`, `unloadModel`, `completion` (blocking and streaming via `AsyncSequence`), `embed`, `transcribe`/`transcribeStream`, `textToSpeech`, `translate`, `diffusion`, `ocr`, `downloadAsset`, `heartbeat`, `close`, and all RAG operations (`ragIngest`, `ragSearch`, `ragChunk`, `ragSaveEmbeddings`, `ragDeleteEmbeddings`, `ragListWorkspaces`, `ragCloseWorkspace`, `ragDeleteWorkspace`, `ragReindex`). The `cancel` function must also be supported to abort in-progress operations. Plugin invocation (`invokePlugin`, `invokePluginStream`) must be supported.
- **SDK integration** — The Swift client lives in the `@qvac/sdk` mono-repo alongside the JavaScript client but is excluded from the npm package. The Bare worker spawning and RPC server setup remain in the existing JavaScript/Bare code — the Swift side only implements the client half. Integration glue (e.g. a thin Swift wrapper that shells out or embeds Bare to spawn the worker) must be provided so that a Swift app can `import QVACClient` and use it end-to-end without manual worker management.
- **Swift Package Manager distribution** — The package must be consumable via SPM with a `Package.swift` at the repo root (or a subdirectory-based package). Tag-based versioning on GitHub. The grant should also include guidance and CI configuration for publishing to the Swift Package Index so the package is discoverable by the Swift community.
- **Platform support** — macOS 14+ (arm64) and iOS 17+ (arm64). Both must be CI-tested.
- **Documentation** — README with integration guide, API reference (DocC), and a minimal example app (SwiftUI) that loads a model and runs streaming completion.
- **Tests** — Unit tests for serialization, RPC message framing, and connection lifecycle. Integration tests that spawn a real Bare worker and exercise the full load → infer → unload cycle.

Scope Exclusions

What important items are out of scope for the project?

- Modifying the Bare worker, RPC server, or addon layer — those remain JavaScript/Bare.
- Android / Kotlin client (separate grant).
- P2P features (`startQVACProvider`, `stopQVACProvider`, `suspend`, `resume`, delegated inference) — may be added later.
- Rewriting or forking the SDK — this is a client, not a replacement.

Deliverables

What are the outputs that must be submitted?

- **Code-gen tooling** — Script or tool that generates Swift source from the JS client's type definitions. Must be runnable as part of CI so regeneration is automated when the JS client changes.
- **Swift package** — `QVACClient` module with full API surface, IPC transport, RPC message handling, and integration glue for worker lifecycle.
- **`Package.swift`** — SPM manifest supporting macOS 14+ and iOS 17+, with library product and test targets.
- **CI configuration** — GitHub Actions workflows for building and testing on macOS (arm64) and iOS simulator. Includes a job that verifies the code-gen output is up to date.
- **Documentation** — DocC catalog, README with setup and usage instructions, and a minimal SwiftUI example app.
- **Test suite** — Unit and integration tests covering serialization, RPC lifecycle, streaming, cancellation, and error propagation.

Acceptance Criteria / Definition of done

What needs to be achieved for this project to be considered completed?

- ☐ Code-gen produces compilable Swift from the current JS client types with zero manual edits.
- ☐ `QVACClient` compiles with Swift 5.10+ / Xcode 16+ on macOS 14 (arm64) and iOS 17 (arm64).
- ☐ A SwiftUI app can `import QVACClient`, load a model, run streaming completion, and unload — all with native `async`/`await`.
- ☐ All RPC message types round-trip correctly (encode → send → receive → decode) against the Bare worker.
- ☐ Streaming APIs (`completion`, `transcribeStream`, `invokePluginStream`) deliver incremental results via `AsyncSequence`.
- ☐ `cancel()` aborts an in-progress operation and the worker acknowledges cancellation.
- ☐ `close()` tears down the IPC connection and the worker process terminates cleanly.
- ☐ Error codes from the worker (`SDK_CLIENT_ERROR_CODES`, `SDK_SERVER_ERROR_CODES`) are mapped to typed Swift errors.
- ☐ CI is green on macOS arm64 and iOS 17 simulator.
- ☐ A reviewer can clone the repo, run `swift build`, and execute the example app within 10 minutes using the README.
- ☐ Re-running the code-gen tool produces no diff against the checked-in Swift sources (verifying sync with JS client).

Milestones

What are the intermediary checkpoints necessary to track project progress?

ID	Description	Deliverables	Reward

M1	Code-gen tooling & IPC transport	• Code-gen pipeline reading JS client types and producing Swift request/response types and serialization logic. • IPC transport layer (Unix domain socket client) with connection, framing, and reconnection. • **Delivers** — Runnable code-gen tool, generated Swift types that compile, IPC transport with unit tests.	800 USDt
M2	Core API surface	• Full `QVACClient` API: `loadModel`, `unloadModel`, `completion` (blocking + streaming), `embed`, `transcribe`, `transcribeStream`, `textToSpeech`, `translate`, `diffusion`, `ocr`, `downloadAsset`, `heartbeat`, `close`, `cancel`. • Worker lifecycle integration glue. • **Delivers** — A Swift app can load a model, run streaming completion, and unload. Integration tests pass against a live Bare worker.	1’000 USDt
M3	RAG, plugins, docs & distribution	• RAG operations (`ragIngest`, `ragSearch`,	1’200 USDt

		`ragChunk`, `ragSaveEmbeddings`, `ragDeleteEmbeddings`, `ragListWorkspaces`, `ragCloseWorkspace`, `ragDeleteWorkspace`, `ragReindex`). • Plugin invocation (`invokePlugin`, `invokePluginStream`). • `Package.swift` with SPM support, CI workflows, DocC documentation, SwiftUI example app, Swift Package Index submission guidance. • **Delivers** — Complete, documented, CI-tested Swift package ready for distribution. All tests green on macOS and iOS.	

Success Indicators / Key Results

What are the key results that help us measure if this project was a success?

- [ ] Clone-to-first-inference in under 10 minutes from the README.
- [ ] Streaming completion latency overhead (Swift client vs. JS client on same machine) < 5%.
- [ ] Code-gen regeneration completes in under 30 seconds.
- [ ] Zero manual Swift edits required when a new SDK function is added to the JS client (code-gen handles it).
- [ ] SwiftUI example app runs on both macOS and iOS physical device.

Applicants Requirements

What requirements that applicants must meet?

- Proficiency in Swift (5.10+), Swift Concurrency (`async`/`await`, `AsyncSequence`, structured concurrency), and SwiftUI.
- Experience with IPC mechanisms (Unix domain sockets, message framing) and RPC protocols.
- Familiarity with code generation tooling (e.g. Sourcery, SwiftGen, or custom AST-based generators).
- Understanding of the QVAC SDK architecture — specifically the client/worker split and `bare-rpc` protocol.
- Access to macOS arm64 (Apple Silicon) and an iOS device for testing.
- Weekly progress updates in English via GitHub issues/PRs.

Reward & Payment Schedule

What are the payout structure and conditions?

Total: 3’000 USDt, three milestone-based installments:

- **M1** — 800 USDt
- **M2** — 1’000 USDt
- **M3** — 1’200 USDt

Each milestone is reviewed within 5 business days. Payment released after reviewer approval. No partial payouts.

Resources & Links

What are additional useful links and resources for applicants?

- QVAC SDK documentation
- QVAC SDK — How it works — Client/worker architecture and RPC lifecycle.
- QVAC SDK — JS API reference — Full API surface the Swift client must mirror.
- QVAC SDK — Installation — Supported environments and platforms.
- QVAC SDK — Error codes — Client and server error codes to map to Swift errors.
- `bare-rpc` — RPC library used by the SDK for client/worker communication.
- Bare runtime — JS runtime that hosts the worker process.
- Swift Package Manager — Package Registry — Registry specification.
- Swift Package Index — Community package discovery.
- Publishing a Swift package with Xcode — Apple's guide to SPM distribution.