

Nereus - Application & Technical Review

Prepared for: Mark Baxter and Keza Studio

Date: 6 October 2026

Version: 1.1

Consolidated product, architecture, security, payments and delivery assessment

1. Executive assessment

Nereus is a mobile-oriented carp-fishing intelligence and logging application built with React/Vite and Base44. Its strongest proposition is to turn private fishing history, environmental conditions and location information into useful planning intelligence. It already contains substantial functionality across venues, swims, spots, sessions, captures, sightings, baiting, forecasts, maps, insights, sharing and subscription scaffolding.

The immediate objective is to productionise the existing application: secure customer data, stabilise critical journeys, give Mark visibility over the business, complete subscriptions and validate field usability. A complete rebuild is not justified by the supplied evidence alone.

The proposed admin portal belongs in the first delivery phase, alongside security and development foundations. It is a Keza recommendation endorsed by Eric, rather than an explicit feature request from Mark. Privileged administration must follow verified backend permissions; building an admin interface alone does not establish secure business control.

Payments are partially implemented, rather than absent. The current pricing is Explorer free, Intelligence £6/month and Elite £8/month. Stripe checkout and webhook functions exist, but a hardcoded Elite tier prevents meaningful paid membership enforcement in the frontend. Production Stripe settings and the complete lifecycle remain unverified.

The supplied security review identifies additional release blockers or high-priority risks: unspecified User entity rules, untrusted server-side image fetching, an authenticated test-data function, thin function input validation, vulnerable dependencies and failing lint/type checks. The review also reports positive foundations: owner-scoped rules on most business entities, Stripe signature verification and a passing frontend build.

Recommendation: pursue a controlled, evidence-led launch programme, beginning with Admin & Platform Foundations, then core stabilisation and subscription completion. Critical outdoor usability fixes belong before paid launch even though deeper UX work has a later roadmap phase. Expand offline support, personalised intelligence and native delivery after the foundations are validated.

2. Review basis, evidence and limitations

This document consolidates the following sources. References such as identify the supplied document and section; transcript references use timestamps.

Ref	Source	Use in this review
S1	Mark Baxter_ Initial Call (1).txt; call on 25 September 2026	Product intent, commercial discussion, mobile direction, instability and launch aspirations

Ref	Source	Use in this review
S2	Nereus audit - access and information needed.eml; Mark's reply and Keza's 2 October request	APIs, database concerns, tester feedback, readability, accuracy, connectivity and access
S3	SECURITY_CODE_REVIEW.md; dated 6 October 2026	Source-based security findings, paths, dependency results and build/lint/typecheck outcomes
S4	USER_JOURNEY_AND_INFRASTRUCTURE.md	Journeys, routes, architecture, entities, functions and workflows
S5	Payment implementation findings provided by Eric in this chat	Plans, checkout function, webhook events, tier override and configuration caveats
S6	Screen inventory and roadmap instructions provided in this chat	16 page components, 15 routable screens, early admin priority and agreed roadmap direction

This is a consolidated review of supplied evidence, not a fresh repository audit, penetration test or live checkout test. The underlying repository and live Base44/Stripe accounts were not inspected during preparation. Build results, vulnerability counts and source findings below are attributed to S3 rather than presented as newly executed checks. No assertion is made that reported security weaknesses have been exploited.

Evidence labels used throughout:

- Reported implementation: described in the supplied technical reviews or Eric's findings; live behaviour may still require verification.
- Founder requirement: explicitly described by Mark in the call or email.
- Recommendation: proposed by Keza or this review; not automatically part of contracted scope.
- Unverified: configuration, operational behaviour or product detail not established by the supplied material.

Mark's original business plan is referenced in the call and email but is not among the supplied attachments. Smaller add-ons and final feature scope therefore remain incomplete. His description of the product as approximately 90% complete is a founder assessment of features, not a measured launch-readiness score.

3. Product direction and founder requirements

3.1 The problem Nereus solves

Anglers record catches, conditions and locations in notebooks or phone notes, but struggle to retrieve patterns across sessions. Nereus aims to combine those records with forecasts so the user can identify conditions resembling previous successes or failures. At the venue, it should help recall successful positions, depth and session context.

The intended experience is private, tactical, clean and readable at a glance. Mark described a military-instrument or Black Ops aesthetic. This supports useful intelligence rather than an obligatory social feed. Existing intentional venue sharing should be retained and secured; private positioning does not mean removing sharing.

3.2 Confirmed needs and recommendations

Item	Evidence/status	Implication
Customer data security and database integrity	Founder requirement, S1 12:40-13:16; S2	Verify data rules, privileged writes, backups and operational control
Built-in paid subscriptions	Founder requirement, S1 23:31-25:31	Complete existing Stripe integration rather than start from scratch
Weather reliability and accuracy	Founder requirement, S2	Treat failed or stale weather as core product degradation
Full-product testing	Founder concern, S2	Expand beyond isolated forecasting tests into complete fishing journeys
Outdoor readability and consistent themes	Founder requirement, S2	Improve sunlight contrast, typography, pop-ups and layout
Poor/no-signal operation	Founder consideration, S2	Investigate and define minimum field requirements before launch
Historical matching and favourable-condition alerts	Core founder vision, S1 05:23-06:52	Validate existing implementation before specifying new intelligence work
Native mobile application	Explicit end goal, S1 50:42-51:27	Maintain a distinct native workstream after launch foundations
Admin portal early in delivery	Keza recommendation endorsed by Eric, S6	Deliver business visibility and controlled support tools in Phase 1
Staging, version control, rollback and monitoring	Keza recommendations discussed in S1 17:23-20:02	Reduce regressions and make releases reproducible

3.3 Timing and commercial context

Mark initially aimed to launch by the end of 2026. Later in the call he described having the app available and refined by March as a useful seasonal objective. Given the September 2026 call date, March 2027 is the likely interpretation, but it needs confirmation and is not a committed delivery date. His availability constraints increase the value of a team-led validation and support process.

The earlier £5/month and possible £2-£3 discussion is historical pricing exploration. The supplied implementation now displays £6 and £8 paid tiers. Neither the earlier discussion nor the current UI alone proves approved live commercial pricing. Trials, annual plans and introductory pricing remain optional decisions.

Competitor and marketing discussion in the call is context, not verified market research. It does not establish current competitor revenue, market size or demand. A separate commercial workstream should confirm positioning, pricing, beta recruitment and acquisition without adding unnecessary social features to the application.

4. Application inventory and screens

There are 16 page components in src/pages and 15 routable app screens wired in App.jsx. The difference is explained by OAuthConsent.jsx, which S4 reports as present but not mounted. The root redirect and wildcard 404 are routing behaviours in addition to those 15 app screens. Authentication, onboarding and modals should not be counted as additional routable screens without evidence.

Route	Purpose	Review focus
/demo	Public preview with injected demo user and mock data	Ensure isolation from real data and paid entitlements
/Dashboard	Home, current venue, conditions, alerts and live-session entry	Load reliability, data freshness and clear priorities
/Map	24-hour and seven-day strategic weather/map context	Tiles, location, markers, overlays and poor-signal behaviour
/Forecast	Seven-day forecast, day/night blocks and alignment	Accuracy, timezones, units and stale-data handling
/Log	Venue-management entry point	Ownership, creation, defaults and navigation
/Insights	Aggregate patterns and personal calibration	Confirm role visibility and intended plan access
/Pricing	Explorer, Intelligence and Elite offers	Restore real tier state and checkout availability
/session	Active fishing session	Capture logging, rod setup, swim movement and recovery
/venue-detail	Venue workspace and historical context	Ownership, sharing, sessions and map usability
/alignment-detail	Detailed condition alignment	Explain scoring, inputs and evidence
/swim-detail	Swim workspace, notes and related spots	Data relationships, media and editing
/bait-up	Baiting records and baited areas	Validation, venue linkage and historical consistency
/spot-detail	Individual fishing spot detail	Location/depth data and related records
/sighting-insights	Sighting patterns and map insights	Filtering, history and calculation consistency
/capture-insights	Capture patterns and map insights	Catch history, weather relationships and export behaviour

Additional routing: / redirects to /Dashboard; * renders the 404 page. No dedicated business admin route is identified in the supplied inventory. Admin-specific visibility inside the existing shell is not equivalent to the proposed business portal.

5. Current user journeys and functional coverage

5.1 Discovery, authentication and onboarding

The public /demo route injects a demo user and mocked Base44 data. Visitors can explore dashboard, map, log and forecast previews; blocked actions show prompts. Unsupported real-app navigation is redirected into the demo shell. The real app checks Base44 authentication behind a splash/loading experience, redirects unauthenticated users to login and presents onboarding when incomplete. Onboarding sets profile/preferences and the initial venue context.

Validation should cover returning users, abandoned onboarding, authentication expiry, failed profile persistence, deep links, unknown routes and demo-to-real transitions. A demonstration must not silently write production records or acquire real administrative privileges.

5.2 Venue, swim and spot management

Users create venues, select defaults and manage swims, spots, notes, photographs and annotations. Venue detail connects tactical map context with session history, sightings and capture insights. Venue sharing provides a snapshot of venue/swim information through invitation and acceptance flows. It should be tested as a separate ownership transition, not assumed to be a live collaborative workspace.

5.3 Fishing sessions

Users start or resume sessions and record captures, bites, sightings, rods, bait context, swim moves and conditions. Ending a session creates historical information used by insights and alignment. Scheduled telemetry is reported in the infrastructure review. The validation priority is persistence through interruptions: a lost network connection, closed app or failed background request must not silently lose fishing records.

5.4 Forecasts, maps and intelligence

The map supports a 24-hour weather-influence view and seven-day strategy. Forecasts combine day/night weather, moon/solar signals, activity indices and personal alignment. Insights use captures, sightings, time of day, wind, temperature, pressure and moon phase; stored preferences support personal calibration. These capabilities are reported as existing, so later roadmap work should validate and refine them before assuming a new implementation is required.

The exact quality and explanatory basis of the predictions are unverified. Mark's vision is historical pattern matching; the supplied material does not establish a trained machine-learning model or a particular AI provider. Avoid describing predictive accuracy as proven.

5.5 Membership and existing admin behaviour

Subscription gates exist conceptually, but the global Elite override defeats the intended frontend behaviour. S4 reports that admins and users share the dashboard shell and that admins have extra navigation such as Signals/Insights. Confirm whether /Insights is intended for admins, paid customers or both; screen names and conditional navigation do not settle the entitlement specification.

6. Technical architecture

6.1 Frontend and state

The reported stack is React with Vite, React Router and React Query. App.jsx owns routes, authentication wrapping, demo routing, splash/loading and global providers. Custom context providers manage auth, venue, map and dashboard/weather state. The UI uses Tailwind, Radix UI primitives, lucide icons, Recharts, React Leaflet, Framer Motion and custom design tokens. PDF export uses jsPDF/html2canvas.

localStorage/sessionStorage are reported for UI/offline caches. Their existence does not demonstrate a complete offline system, durable write queue, conflict resolution or background synchronisation. Cache lifetime, sensitive content and account-switch cleanup need verification.

6.2 Backend, data and media

Base44 supplies authentication, entities, serverless functions and hosted media, accessed through @base44/sdk. Most business entities reportedly use owner-scoped row-level security. VenueShare has sender/recipient rules. User is an important exception requiring explicit verification.

Reported model	Role	Integrity/security focus
User	Identity, role, membership, onboarding and personal information	Protect role and membership against client writes
Venue	Fishing location and venue workspace	Ownership, defaults and admin access
SwimData	Swim details and associated context	Correct parent venue and media ownership
Session	Fishing session and condition history	Owner, venue/swim links, active/end state
Capture	Catch records and related measurements	Valid session links, units and private history
Sighting / FishShow	Observation and fish-show records	Location, time, ownership and pattern inputs
BaitLog	Baiting records	Parent relationships and valid quantities/context
VenueShare	Invitations and snapshot sharing	Sender/recipient access, expiry and safe copying
UserSettings / Preferences	Defaults and personal calibration	Owner scope and allowed fields

These are the model names reported by S4, not a freshly validated schema diagram. Exact field constraints, relation enforcement, record volumes, indexes, retention and deletion behaviour remain to be examined in the deployed project.

6.3 Functions and scheduled workflows

Reported backend responsibilities include creating/accepting/declining/cancelling/expiring venue shares, retrieving weather timelines, checking dynamic alerts, capturing session telemetry, creating Stripe checkout, processing Stripe webhooks, maintenance/test helpers and saving default venues. Scheduled workflows capture session telemetry and expire share invitations.

Deployment verification must establish authentication, input bounds, rate limits, service-role use, retries, idempotency and monitoring for each exposed function. Scheduled job definitions in an export are not evidence that those jobs are enabled or running reliably in production.

6.4 Integrations and data flow

- Weather: Mark confirms Open-Meteo; functions retrieve/normalise weather for the app. Validate account configuration, capacity, permitted usage, costs and fallback strategy in the deployed setup.
- Mapping: React Leaflet renders maps; Mark identifies Mapbox as the mapping service, and S3 identifies public Mapbox token usage. These statements are compatible: the rendering library and tile/service provider serve different roles.
- Billing: the browser requests server-created Stripe checkout; Stripe sends signed events to the backend, which updates Base44 membership state.
- Media: Base44-hosted media/CDN URLs support uploads and sharing. Public/private access behaviour and safe copying require verification.

Main data path: browser authenticates → loads user-scoped entities → calls weather/alert functions → receives conditions → records fishing activity → revisits historical insights. The intended payment path is checkout → signed webhook → trusted membership update → refreshed app entitlement, but the final entitlement step is currently bypassed.

6.5 Configuration and platform dependence

The local project expects VITE_BASE44_APP_ID, VITE_BASE44_APP_BASE_URL and optionally VITE_BASE44_FUNCTIONS_VERSION. Stripe server-side configuration includes STRIPE_SECRET_KEY and STRIPE_WEBHOOK_SECRET. Missing values in the reviewed local checkout do not prove missing production secrets. Secrets must remain on the backend; the public Mapbox pk. token is designed for client exposure but requires suitable restrictions.

The reported configuration uses npm install, npm run build, npm run dev and output ./dist. A passing build does not prove authenticated API operation, webhook delivery or production configuration. Exported React code still depends on Base44 authentication, entities, functions, workflows and media; hosting the frontend elsewhere does not remove those dependencies. Migration should follow a capability, cost and portability assessment, with data export and rollback planning, rather than an assumed immediate rewrite.

7. Payments and membership review

7.1 Current plan definitions

Tier	Displayed price	Stored membership value
Explorer	Free	explorer
Intelligence	£6/month	intelligence
Elite	£8/month	elite

These are reported /Pricing definitions, not verified live Stripe prices. Confirm currency, tax presentation, billing period, benefits and exact Stripe Price IDs before customer release. Feature quotas and tier-by-tier benefits are not fully supplied and must be inventoried rather than invented.

7.2 Implemented billing flow

1. The pricing UI is intended to call the Base44 createCheckoutSession function.
2. The backend creates a Stripe Checkout Session in subscription mode using hardcoded Price IDs.
3. Stripe delivers checkout.session.completed, customer.subscription.updated and customer.subscription.deleted events.
4. The webhook maps recognised prices to User.membership_tier: intelligence, elite or explorer.
5. The frontend should reflect that trusted membership state and apply feature/venue restrictions.

S3 reports server-side checkout creation, Stripe signature verification using STRIPE_WEBHOOK_SECRET and tier mapping from Stripe Price IDs rather than blindly trusting the client. These are positive design elements; live settings and event processing are not verified.

7.3 Current enforcement defect

The reported src/lib/membership.js implementation is:

```
export function getUserTier(user) {  
  return 'elite';  
}
```

Every user is treated as Elite. The pricing screen therefore thinks users are already upgraded, normal upgrade actions may be hidden, premium UI/venue limits are bypassed and webhook-written membership is ignored. S3 also identifies an Elite fallback for missing tier data in base44/functions/acceptVenueShare/entry.ts. Both paths need correction.

Reading `user.membership_tier` with an Explorer fallback is a necessary UI correction, not a complete security solution. Users must be unable to edit membership themselves, and protected backend actions must independently check trusted entitlement. Otherwise a modified client or direct function request can bypass presentation-level gates.

7.4 Remaining commercial gaps

Gap	Required outcome
Unverified deployed secrets and webhook endpoint	Confirm correct test/live account, secret, endpoint, products and event delivery
Price IDs duplicated in checkout/webhook	Use a maintained backend plan mapping and verify matching prices
Request-origin success/cancel URLs	Configure or validate an approved app domain and test deployed redirects
No reported customer billing portal	Provide cancellation, payment-method updates, invoices and approved plan changes
Limited lifecycle evidence	Define active, trial, past-due, unpaid, cancelled and expired access rules
Subscription state not fully represented	Store enough trusted status/period/customer identifiers for billing and admin support
Refresh after checkout unverified	Show processing state until membership is confirmed; handle delayed webhooks
Duplicate/out-of-order events unverified	Make webhook updates idempotent and avoid stale events restoring incorrect access
Duplicate checkout/upgrade behaviour unverified	Avoid unintended parallel subscriptions; specify upgrade and downgrade handling

Cancellation at period end should normally preserve access until the paid period expires if that is the chosen policy. Cancellation scheduling, immediate cancellation, subscription deletion and failed renewal are different states. They must not all trigger an immediate Explorer downgrade without an agreed rule. The current deletion mapping expresses intent, not a complete lifecycle policy.

7.5 Payment acceptance criteria

- Explorer sees correct upgrade actions and cannot execute protected paid operations directly.
- Intelligence and Elite test checkout assign the correct verified entitlement and display the correct price.
- Intelligence-to-Elite changes do not accidentally create duplicate subscriptions; downgrade policy and venue limits are explicit.
- Invalid-signature webhooks are rejected; duplicate and delayed events produce consistent state.
- Cancellation, period expiry, failed payment and recovery follow the approved access policy.
- Redirects return to the deployed app; membership updates become visible without granting premature access.
- Customer billing management and admin subscription views reflect the provider's trusted state.
- Test and live environments remain separate, with a documented production smoke test.

8. Security and data protection findings

Priorities below retain S3's classification. P0 denotes immediate launch-blocking concern; P1 high-priority remediation; P2 hardening/operational improvement. Conditional exposure must be verified rather than reported as a confirmed breach.

Priority	Finding and evidence	Required response
P0	User.jsonc has sensitive role/membership/profile fields but no explicit rls block	Verify Base44 User defaults; enforce allowed self-service fields and trusted privileged writes
P0	Global Elite override and share-accept Elite fallback	Restore conservative tier resolution and backend entitlement enforcement
P1	acceptVenueShare fetches client-influenced photo URLs server-side	Prefer trusted hosted assets; enforce host/redirect safety, bounds and image validation
P1	restoreTestCapture permits any authenticated caller	Remove from production or restrict to controlled admin/development use
P1	getWeatherTimeline and checkDynamicAlerts accept weakly bounded inputs	Validate request schemas, coordinates, timezone, numerical ranges and payload size
P2	Base44 client uses requiresAuth: false	Isolate demo behaviour and verify every entity/function enforces backend auth
P2	Admins can read/update/delete all venues	Secure role assignment, minimise privileges and audit admin operations
P2	Public Mapbox token duplicated across several files	Centralise configuration and verify provider restrictions/rotation

8.1 User and administrative permissions

Missing exported User rules create uncertainty, not proof that all users can edit roles. Confirm platform protections with two ordinary accounts and an admin account, including direct SDK/function access. Account A must not read or mutate Account B's private records. A normal user must not set role, membership_tier or trusted billing identifiers. Admin controls must use backend authorization, with audit records for sensitive actions.

8.2 Venue-sharing image handling

S3 reports fetch(photoUrl) followed by upload of the response blob. Restrict fetching to trusted HTTPS media sources where possible. If external URLs are necessary, validate resolved destinations, deny internal/private/link-local ranges, control redirects and impose timeouts and byte limits. Confirm actual image content, not just an untrusted header. This addresses reported server-side request forgery, storage abuse and slow-fetch risks. No live exploit test is supplied.

8.3 Weather and alert inputs

Validate latitude between -90 and 90, longitude between -180 and 180, recognised timezones and bounded arrays/numeric values. Reject malformed and oversized data early. Consider server-fetched authoritative weather inputs for dynamic alerts rather than trusting arbitrary client arrays. Add operational request limits where supported.

8.4 Privacy, resilience and recovery

Exact fishing spots and historical patterns are central private product data. Recommended operational work includes documenting ownership, consent for sharing, retention, account deletion/export, media access and cache cleanup. Confirm backup coverage across entities and media and demonstrate recovery with an agreed recovery point/time objective. Provider hosting geography, retention defaults and backup capability are not established by the supplied files; these are verification tasks rather than confirmed platform deficiencies.

9. Code quality, dependencies and engineering health

Check	Supplied result	Interpretation
npm run build	Pass	Frontend compiles; live integrations remain unverified
Build configuration	Missing Base44 app ID/base URL warnings; stale browser data	Configure local/staging testing and refresh relevant tooling
npm run lint	37 errors	Mostly unused imports; reported invalid playsInline prop on audio in SplashScreen.jsx
npm run typecheck	Substantial failures	Third-party JS/configuration noise plus real prop/API/SDK mismatches
npm audit	35 vulnerabilities: 1 critical, 20 high, 13 moderate, 1 low	Review and remediate reported advisories; counts are the supplied snapshot

Reported affected packages include jsPDF (critical), Vite, axios, lodash, rollup, postcss and ws (high), plus DOMPurify and Quill/react-quill XSS-related advisories (moderate). Runtime exposure, advisory versions and available fixes need a fresh lockfile-based assessment during implementation. A vulnerability total is not proof of exploitation.

Update dependencies in a controlled branch and rerun appropriate build and journey checks, particularly PDF generation, charts, map screens and rich-text surfaces. Avoid blanket forced upgrades without assessing breaking changes. Fix lint, reduce vendor/generated typechecking noise and resolve genuine application errors; do not achieve a green check by suppressing meaningful failures. Establish required CI checks before release. These checks have not been rerun in this consolidation.

10. Weather, mapping and field reliability

Mark reports intermittent weather-fetch failure and suspects limits, but the actual cause is unconfirmed. Instrument provider status, response time, retries, cache hits and function errors before diagnosing rate limits. Establish load assumptions and costs based on real request patterns.

Recommended weather behaviour includes bounded retries, request deduplication, location/time-aware caching, explicit last-updated timestamps, sensible refresh intervals and graceful failure states. Cached forecasts must be labelled stale and must not appear to be live measurements. Validate timezone/daylight-saving boundaries, measurement units, pressure/wind interpretation and calculation consistency across dashboard, map and forecast.

For mapping, test tile failures, unavailable location permission, inaccurate coordinates, markers, pop-ups, map gestures and mobile overlays. Centralise the public token and verify provider controls. Any offline tile

plan must respect the selected provider's actual terms and capacity; offline maps are an investigation item, not a capability proven by browser storage.

10.1 Offline scope

Separate three capabilities: viewing cached data, recording new fishing activity offline and synchronising safely after reconnect. A staged approach can provide cached sessions/venue information and clearly labelled forecasts first, followed by durable queued writes with deduplication and conflict handling. Define how an offline device learns of subscription changes and how shared media behaves without connectivity.

Test with throttled/intermittent signal, airplane mode, browser restart, expired authentication and reconnect. Native delivery may improve device integration but does not automatically implement offline durability or background reliability. These must be designed and verified independently.

11. UX, accessibility and outdoor use

Mark prioritises ease of use, ease of viewing and data accuracy. His friends' limited weather/wind testing was generally positive, but highlighted reading a dark screen in bright sunlight. Light mode was added with reported inconsistencies in page themes, colours and opacity. Pop-ups hide key details and font size may be difficult for older anglers.

Recommended work: define shared theme tokens, apply light/dark modes consistently, improve readable typography and contrast, preserve critical map data around pop-ups, use clear touch targets and prioritise essential information on each screen. Honour the single-screen tactical preference where practical without shrinking text or hiding information to achieve it.

Mark describes the product as designed for mobile and poor on desktop. Verify actual devices and web layouts rather than relying solely on Base44 preview. Define a supported desktop experience or minimum responsive layout; the proposed admin portal should be comfortable on desktop as well as usable on mobile. Test loading/error/empty states and splash duration as part of perceived speed.

12. Proposed Admin Portal V1

The portal is an early business-control recommendation. Existing Base44 collaborator/admin access and role-specific customer navigation do not satisfy this requirement. Deliver a small operational portal before deeper analytics or CMS functionality. Backend roles and privacy boundaries are prerequisites.

Area	V1 scope	Dependency/acceptance
Dashboard	Registered users, clearly defined active users, growth and basic usage	Document metric definitions and exclude demo/test records
Users	Searchable/paginated list, registration date and account state	Server-side admin authorization; no unnecessary private fishing data
User detail	Profile/status, membership, support context and limited activity	Minimise sensitive access and audit privileged actions
Account actions	Controlled suspension/reactivation and support tools	Define policy and ensure backend access is actually revoked
Subscriptions	Tier, trusted billing status, renewal/end date, failed payment/cancellation	Integrate provider-backed data when payments are completed
System health	Weather/function errors, webhook failures and scheduled-job status	Show instrumented evidence rather than invented health indicators

Area	V1 scope	Dependency/acceptance
Settings	Approved business configuration and operational controls	Restrict privileged changes and validate values

In Phase 1, subscription views may display current stored membership with an explicit testing/unverified label. Paying/trial/cancellation counts and revenue must not be inferred from the global Elite override or unverified tier fields. Accurate billing metrics become available after trusted subscription state is connected in Phase 3.

Define active-user periods, recurring-revenue calculation, refunds/discount handling and cancellation measures before publishing business metrics. Trials should only appear as real categories if the commercial model introduces them. Exclude unrestricted impersonation, arbitrary entitlement overrides, broad private-history browsing and destructive bulk actions from V1 unless separately scoped and controlled.

13. Delivery roadmap and release gates

The sequence below consolidates the agreed roadmap. Phases are workstreams with dependencies; they are not estimates or an instruction to defer essential safety/usability checks until after launch.

Phase 1 - Admin & Platform Foundations

Build the initial admin shell, user overview and support visibility alongside verified User/entity permissions. Remove or restrict production test helpers, harden sharing fetches, establish conservative entitlement handling and resolve critical dependency exposure. Set up controlled development/staging/production configuration, version control, rollback, error monitoring and backup/recovery verification.

Exit gate: privileged actions are backend-protected and auditable; ordinary users cannot alter roles or billing entitlement; cross-user access tests pass; critical exposed security risks are resolved; releases can be tested and rolled back. Payment panels remain honest about incomplete live billing data.

Phase 1 workstream - Staging & Live Deployment Pipeline

Create a repeatable deployment pipeline for separate staging and live (production) environments as part of Admin & Platform Foundations. This is a recommended delivery item, not a capability verified in the current deployment. Confirm the supported Base44 deployment and environment options before choosing the implementation; the pipeline must cover backend entities, functions and workflows as well as the frontend.

Why it is needed

Mark reports that changes to one part of Nereus can unexpectedly break another and that these issues may only be discovered after publishing. A staging environment gives Keza and Mark a safe place to test the complete application before releasing changes to customers. A deployment pipeline makes the checks and release steps consistent, reducing manual mistakes and providing a traceable record of what was released. This is particularly important for weather calculations, private fishing records, administrative permissions and subscription access.

Separate environment configuration also prevents testing from affecting live records, sending unintended notifications or using live Stripe billing. A known release version and rollback procedure allow the team to recover quickly if a deployment introduces a regression. Staging reduces release risk; it does not replace monitoring and smoke tests on the live application.

Recommended pipeline

6. Commit changes to version control and review them through a pull request.

7. Run automated validation: production build, lint, typecheck and targeted permission, entitlement and critical-journey tests. Resolve current failures before treating these checks as effective release gates.
8. Deploy a versioned candidate to staging using isolated data, test accounts and Stripe test mode. Keep environment-specific credentials and webhook endpoints separate from production.
9. Run staging smoke and regression tests across login/onboarding, venues, sessions, maps/weather, sharing, admin and subscriptions. Record the results against the candidate version.
10. Promote the tested release to live through a controlled release gate. Preserve the same frontend artifact where supported and record matching backend/configuration changes so the production release corresponds to the tested candidate.
11. Run live smoke tests and monitor errors, weather/API health and webhook processing. Use the documented rollback or recovery procedure if release criteria fail.

Implementation scope

- Separate staging and production URLs, data stores or app projects as supported by the platform, authentication configuration, media access, secrets and billing endpoints.
- Sanitised or synthetic staging data; no routine copying of private production fishing records into test environments.
- Explicit controls for scheduled jobs, share invitations and notifications so staging cannot act on live customers.
- Automated deployment commands or supported provider integrations, with restricted deployment permissions and protected production credentials.
- A release record covering code version, frontend artifact, entity/schema changes, functions, workflows and environment configuration.
- A tested rollback process. Frontend rollback alone may not reverse schema changes, data mutations or billing events; define backward-compatible changes and recovery steps for those cases.
- A short deployment runbook documenting ownership, required checks, release approval, smoke tests and incident recovery.

Acceptance criteria

Staging and live are demonstrably isolated; required checks block failed candidates; staging testing is recorded; production releases are traceable to tested versions; secrets remain protected; post-release checks run; and the team has demonstrated rollback or recovery for a representative failed release. The selected tooling and any Base44 limitations should be documented before implementation is estimated.

Phase 2 - Core Application Stabilisation

Fix critical defects, weather and map reliability, input bounds and calculation consistency. Validate complete onboarding, venue, session, sharing, insight and account journeys. Restore useful lint/type checks and introduce targeted regression coverage. Address sunlight readability and navigation defects that prevent real field use.

Exit gate: critical journeys pass on agreed devices and field conditions; weather failure produces understandable states; session history persists; no unresolved data-loss or cross-account defects; actual angler feedback has been reviewed.

Phase 3 - Complete & Productionise Payments

Finish trusted membership enforcement, provider configuration, price mapping, webhook lifecycle and customer billing management. Define plan benefits, limits, cancellation, grace periods and upgrade/downgrade policy. Connect verified subscription state to admin, then test on the deployed app URL.

Exit gate: full subscription lifecycle passes; direct calls cannot bypass paid permissions; checkout redirects and delayed events work; customer management and admin billing agree with Stripe. Final prices and paid-launch terms are approved.

Phase 4 - UX, UI & Outdoor Usability

Complete consistent light/dark themes, readable typography, tactical layouts, map/pop-up positioning and device responsiveness. Improve at-a-glance information without sacrificing accessibility. Customer usability fixes required for launch may run during Phases 1-3.

Exit gate: representative anglers can complete core tasks outdoors on supported devices; key information remains legible and accessible; themes are consistent.

Phase 5 - Low Connectivity & Offline Capability

Define minimum offline scope, cached history/location/weather, stale-data indicators, durable local records and reconnect synchronisation. Investigate mapping constraints and secure local data handling.

Exit gate: supported offline journeys survive disconnect/restart/reconnect without silent loss or duplicates. Any minimum offline capability considered essential by Mark becomes a prelaunch requirement rather than automatically postlaunch.

Phase 6 - Intelligence, Alerts & Notifications

Validate and refine existing historical matching, alignment, activity calculations and dynamic alerts. Document which data drives predictions, insufficient-data behaviour and useful notification preferences. Evaluate venue, time, pressure, wind, temperature, moon and depth inputs without claiming accuracy that has not been measured.

Exit gate: matching and alerts agree with defined rules, are explainable, respect user preferences and are tested on representative history. Existing broken intelligence is fixed earlier; this phase concerns deeper refinement and expansion.

Phase 7 - Native iOS & Android

Choose an implementation strategy after stabilising the product. Scope native notifications, offline persistence, device permissions, background behaviour, store submissions and UX refinements. Verify the applicable distribution/billing rules when implementation starts; do not assume existing web Stripe checkout can be reused unchanged inside store apps.

Exit gate: mobile permissions, session durability, notifications, billing approach and store-ready builds are validated. Native timing is separate from the web paid-launch commitment unless Mark expressly changes launch scope.

Paid launch gate

Paid release requires verified security, dependable critical journeys, accurate entitlement and billing, viable outdoor readability, customer support/admin visibility, monitoring and a recovery/release process. Offline scope and native requirements must be explicitly decided. A successful frontend build or completion of an admin dashboard is insufficient alone.

14. QA and validation programme

Test group	Essential scenarios
Identity and isolation	Login/logout, expired auth, onboarding, A/B user isolation, denied role/membership edits
Venues and sharing	Create/default/edit venue, swim/spot/media links, invite/accept/decline/cancel/expiry and invalid URLs

Test group	Essential scenarios
Fishing activity	Start/resume/end, captures/bites/sightings/rods/bait, swim moves, reload and interruptions
Weather and maps	Valid/invalid coordinates, timezones, provider failure, stale forecasts, tiles and location denial
Insights	Known-data calculations, empty/small histories, units/time alignment and intended role/tier visibility
Billing	Each paid plan, upgrade/downgrade, failure/recovery, cancellation/expiry, duplicate/delayed webhook and portal
Demo	Mock data isolation, blocked writes, safe route transitions and no admin access
Admin	Non-admin denial, correct metrics, suspension enforcement and audit records
Field usability	Sunlight, dark mode, older-user readability, poor signal and supported mobile browsers
Release and recovery	Required checks, staging validation, production smoke test, rollback and recovery evidence

Prioritise automated tests for permissions, entitlement transitions, calculations and previously broken high-value journeys. Complement them with manual exploratory and field testing; automated checks alone cannot establish outdoor usability. Record severity, reproducible steps, environment, owner and retest result for each defect.

15. Outstanding decisions and access requirements

Item	Why it matters	Suggested owner
Original business plan and guided feature walkthrough	Complete intended-vs-built inventory and smaller add-ons	Mark / Keza product lead
Confirm collaborator access and deployed entity/function/workflow settings	Email says invitations sent; effective access and configuration still need verification	Mark / technical lead
User entity default protections and trusted writes	Resolve conditional privilege/membership exposure	Technical lead
Stripe test/live access, price IDs, endpoint and secrets	Validate current checkout rather than infer configuration from missing local values	Mark / technical lead
Final tiers, benefits, quotas and commercial rules	Prevent inconsistent pricing, gates and billing state	Mark / product lead
Weather/Mapbox capacity, provider settings and costs	Establish launch reliability and operating assumptions	Technical lead
Insight/alert scope, role visibility and accuracy criteria	Separate existing features from later enhancements	Mark / product lead
Offline minimum and device support	Set the field-use launch gate	Mark / QA lead

Item	Why it matters	Suggested owner
Backup, deletion/export, retention and media rules	Establish operational recovery and private-data handling	Technical lead / Mark
Admin support scope and metric definitions	Avoid excessive private access and misleading business metrics	Mark / product lead
Launch target, budget, native strategy and capacity	Convert phases into a realistic dated delivery plan	Mark / Keza delivery lead

These are inputs to implementation and estimation, not reasons to defer currently authorised documentation work. No firm budget, effort estimate or delivery commitment can be derived from the supplied sources alone.

16. Recommended immediate backlog

12. Obtain the business plan, confirm deployed access and run a guided walkthrough against the 15-screen inventory.
13. Verify User rules and direct-call authorization; protect role/membership writes before introducing privileged controls.
14. Correct the global Elite override and sharing fallback, with trusted backend entitlement checks.
15. Remove/restrict restoreTestCapture; harden share image copying and function input limits.
16. Triage dependency exposure and restore useful build/lint/type checks in controlled branches.
17. Establish the staging/live deployment pipeline and release runbook in Phase 1; deliver Admin V1 user visibility, support controls and instrumented health; defer real revenue claims until billing is verified.
18. Instrument and reproduce weather failures, validate map behaviour and test session durability.
19. Verify Stripe deployment and full lifecycle, add customer management and connect trusted billing to admin.
20. Run representative angler testing, fix critical sunlight/layout issues and decide minimum offline support.
21. Approve launch gates and a dated plan; retain intelligence refinement and native delivery as explicit next workstreams.

17. Overall conclusion

Nereus has substantial product coverage and a clear private fishing-intelligence proposition. Its next stage is operational and technical maturity: secure the foundation, introduce early business administration, stabilise existing journeys and complete the partially implemented subscription system. Later work should strengthen field reliability and validated personalised alerts while preserving the native mobile goal. Readiness should be demonstrated through release gates and live verification rather than inferred from feature count or a successful build.

Appendix - Technical evidence locations

Location reported in S3/S4	Relevant concern
src/App.jsx	Route inventory, auth/demo wrapping and providers
src/pages/OAuthConsent.jsx	Present but unmounted page component
src/pages/Pricing.jsx	Plan presentation and checkout initiation

Location reported in S3/S4	Relevant concern
src/lib/membership.js	Hardcoded Elite entitlement
base44/entities/User.jsonc	Role/membership/profile security rules
base44/entities/Venue.jsonc	Broad admin venue permissions
base44/functions/createCheckoutSession/entry.ts	Stripe checkout, prices and redirect origin
base44/functions/stripe-webhook/entry.ts	Signature verification and membership updates
base44/functions/acceptVenueShare/entry.ts	Remote photo fetch and Elite fallback
base44/functions/restoreTestCapture/entry.ts	Authenticated test-data creation
base44/functions/getWeatherTimeline/entry.ts	Coordinate/timezone validation and upstream fetches
base44/functions/checkDynamicAlerts/entry.ts	Client hourly data validation
src/api/base44Client.js	requiresAuth: false
src/lib/mapboxConfig.js	Central mapping configuration
ActiveSession.jsx, VenueDetail.jsx, CaptureInsights.jsx, SightingInsights.jsx, session/pdfHelpers.js	Reported duplicated Mapbox token usage
src/components/SplashScreen.jsx	Reported invalid audio prop
jsconfig.json	Typecheck scope and dependency noise
base44/workflows	Telemetry capture and share-expiry jobs

Source references identify the original supplied materials. This consolidation does not certify production settings or remediation completion.