ChatGPT

Ocean Pearl OPS V2 — CEO Takeover Audit & Target-State Blueprint

1 Executive Verdict

Verdict: Conditional Go – the system has reached a usable prototype stage after remediation but is not ready for production at multi-island scale without further work. It can be used by the CEO for limited trials provided the critical risks and missing features are addressed before real money and stock are processed.

Reasoning (summary):

1. **Write-guard issues have been mostly fixed but security gaps remain.** The `Audit Remediation Report` confirms that view-as writes, operate-as confirmation and the expenses bug were patched [1]. However, Firestore rules still allow writes to `expenses`, `vendors` and other collections by any authenticated user [2] and there is no server-side check that a CEO is in view-as mode. This leaves potential for fraud and data tampering if a malicious user bypasses the UI.
2. **Financial and inventory modules are functional but rely heavily on client logic.** Transactions, wallet movements and stock updates are processed through a single cloud function (`postTransaction`) with proper audit logging [3], but the function still accepts `role_v2` fields from the client and does not verify location/unit context. Expenses and approvals use separate functions yet their collections remain publicly writable. This weakens segregation of duties.
3. **UI/UX is well designed for core flows but lacks completeness.** The command centre, wallet manager, reports and admin panel work smoothly. However, some screens (production run, cold storage, admin edits) lack the new write guard; the admin panel still uses browser confirms [4]; and certain metrics (yields, negative stock) show inconsistent data [5].
4. **Audit trail exists but lacks completeness and tamper-proofing.** Each transaction writes an `audit_logs` document with user, amount and timestamp [3], but other actions (expense creation, user edits, context switching) are not logged. There is no immutable append-only log; logs can be modified or deleted by a root admin.
5. **Scalability and extensibility for new units are partially addressed.** Current schema supports multiple locations/units; constants define capabilities [6]. Nevertheless, adding boat/landing entities will require careful schema redesign to avoid mixing concerns and ensure yields, crew payments and landings are correctly linked. The `Shark AI` module remains limited; risk alerts and digest functions are manual rather than automated.

2 Top 12 Critical Risks (ranked)

#	Risk	Impact	Likelihood	Evidence & Notes	Fix approach
1	**Security rules allow direct writes to sensitive collections**	Any authenticated user can create/modify documents in `expenses`, `vendors`, `expense_types` and legacy `partners` because rules simply check `request.auth != null` [2]. A rogue operator could inflate expenses, change vendors or delete partner data.	High – proven by code.	Refactor Firestore rules: deny all writes from client except via cloud functions, define granular rules per collection and role. Deprecate `isAdmin` rule and rely solely on `role_v2` claims.	
2	**Server functions do not validate CEO View-As / Operate-As flags**	If a CEO is in view-as mode, they should be read-only; however, server functions rely on `role_v2` and location fields sent by the client. A malicious user could craft a request with `role_v2` set to `HQ_ADMIN` and perform writes despite being in view-as mode [7].	High – view-as bug already exploited in audit.	Add server-side middleware to verify `ceoMode` context and original role; block writes when `_isViewAs` flag is set and require confirmation for first operate-as write. Include `actorContext` fields (mode, originalRole) in the function payload and check them.	
3	**Publicly writable expenses module**	The `Expenses` page writes directly to Firestore using client SDK【733521204814541†L45-L74】; anyone logged in (even investor) can create expenses or edit existing ones. There is no server-side validation or audit trail.	High	Create a cloud function `createExpense` that enforces role and location checks (unit operator can only create within their unit, manager must approve). Lock `expenses` collection for direct writes.	

#	Risk	Impact	Likelihood	Evidence & Notes	Fix approach
4	**Inconsistent number parsing and NaN propagation**	Wallet calculations previously suffered NaN due to string numbers. The remediation attempted to fix this but code still uses `parseFloat()` in many places (front-end and functions) without robust checks, which can return NaN if inputs are malformed [8].	Medium – leads to negative or 0 balances.	Implement strict schema validation with Joi/Zod on all inputs, convert numbers server-side, and reject non-numeric values. Provide UI formatting and enforced input masks.	
5	**Lack of immutable audit trail**	Audit logs can be modified or deleted by admins. There is no server-timestamped append-only log for all actions (login, context change, expense approval).	Medium	Use Cloud Functions triggers to write to a separate `audit_events` collection with `create` rule only, disallowing updates/deletes. Configure Firestore retention or export to BigQuery for immutability.	
6	**Incomplete write guards and pages not covered**	The new `guardWrite` is implemented in some forms but not in all modules: production run, receiving, admin panel actions (reset password), and data import still allow immediate writes [9].	Medium	Integrate `useWriteGuard` into every component that performs writes. Standardize a wrapper hook around `postTransaction`, `createExpense`, `approveRequest` etc.	
7	**Admin Panel uses blocking browser confirm dialogs**	Reset-password and delete actions still rely on `window.confirm` [4], which is not consistent with the new confirmation modal design and may cause double-submissions or user confusion.	Medium	Replace all `window.confirm` calls with custom modal components that integrate with the guard and provide consistent look & feel.	

#	Risk	Impact	Likelihood	Evidence & Notes	Fix approach
8	**User roles stored inconsistently**	The user schema still includes legacy `role` and `loc` fields along with new `role_v2`, `locationId` and `target_id`. Many functions rely on fallback logic (if `locationId` missing, use `loc` or `target_id`) [10]. This increases complexity and risk of mis-scoped operations.	Medium	Migrate user documents to a single canonical structure: `role_v2`, `locationId`, `unitId`, `target_id` (if necessary). Remove legacy fields and update all functions to rely on canonical fields.	
9	**Incomplete context enforcement on Firestore queries**	Front-end queries for reports sometimes fetch all documents and then filter client-side, which can leak data (e.g., manager queries all transactions then filters by location) [11].	Medium	Create query functions with proper server filters based on location and role, using Firestore composite indexes. Client should never fetch data outside user's scope.	
10	**No automatic anomaly detection ("Shark AI" limited)**	Shark AI currently offers chat assistance and can draft transactions but lacks automated risk alerts. The seeds show unrealistic yields (e.g., -5 raw input, 4.5 output, yield 90 %) [5] but no alert is generated.	Medium	Develop scheduled functions or Firestore triggers that compute KPIs (yield variance, duplicate receiving, abnormal wallet movements) and write risk alerts to `admin_notifications` with references to underlying records.	
11	**Negative stock and wallet positions not enforced across all modules**	`postTransaction` denies negative stock/ wallets [3], but expense approvals and other flows may not check balances. Also, there's no multi-currency or currency conversion support.	Medium	Ensure all functions that withdraw from stock or wallet perform atomic checks and reject transactions leading to negative balances. Add currency field to wallets and implement currency conversions if needed.	

#	Risk	Impact	Likelihood	Evidence & Notes	Fix approach
12	**Scalability limitations for upcoming boat operations**	The current schema lumps many transaction types into `transactions` and uses nested location/unit IDs as strings. Future features (trip logs, crew payments, landing notes) will require more detailed relationships. Without planning, adding boats could result in brittle code.	Medium	Design a separate `boats` collection with sub-collections for trips, landings, crew advances, fuel logs, and link them to processing batches via references. Add generalization in functions to handle new unit types and new transaction categories.	

3 “Ready for Daily Use” Checklist

Item	Status (Pass/ Fail)	Notes
Role-based access for core modules (dashboard, wallet, reports)	**Pass**	Role tests confirm that HQ Admin, Location Manager, Unit Operator and Investor roles see correct modules and cannot access prohibited pages [11] .
Expense creation & approval flows	**Fail**	Bug fixed in remediation, but direct writes are still possible from client, and there is no server audit trail.
Receiving & production run flows	**Partial**	UI is functional and pushes transactions through `postTransaction`. However, write guard not yet integrated on all forms, yields sometimes produce unrealistic numbers and are not validated.
Wallet management & funding requests	**Pass / Partial**	Creating funding requests and approvals works via `financial_v2` functions with proper checks and atomic updates [12] . Yet NaN bug still possible if number parsing fails.
CEO control (view-as and operate-as)	**Partial**	UI prevents writes in view-as and shows a confirmation on first operate-as write [13] . However, server functions still rely solely on role and do not enforce view-as context, making this only a UI guard.
Shark AI chat & risk alerts	**Partial**	Chat works and respects scope [14] , but there are no automated alerts or digest.

Item	Status (Pass/ Fail)	Notes
Audit logs & tamper protection	**Fail**	Audit logs recorded only for `postTransaction`; other actions not logged; logs can be deleted or modified.
Admin panel (users, items, partners)	**Partial**	Basic CRUD works; lacks guard integration; resets use browser confirm; may crash on some actions [9].
Performance & offline queue	**Pass**	Transaction queue stores operations offline and flushes them upon reconnect; improvement to guard prevents silent writes in view-as.
Scalability & multi-location	**Pass / Partial**	Schema uses `locations` and `units` and can support new locations easily. Multi-currency, boat operations, and export flows not yet supported.

4 Final Target State Blueprint

4.1 Final UI Sitemap (roles)

1. **Global (HQ Admin / CEO)**
2. **Dashboard / Command Center** – Snapshot of company: cash position, inventory summary, production status, alerts, yield KPIs, AI insights.
3. **Treasury** – Manage location wallets, fund transfers, approvals, view cash ledger.
4. **Procurement & Receiving** – Create receiving invoices (purchases), see supplier due amounts, verify weights, price variance.
5. **Production / Processing** – Record production runs (drying, freezing), monitor yields, track scrap/ waste.
6. **Inventory (Cold Storage / Dry warehouse)** – Stock lists, batch ageing, reservations, pick/pack for sales orders.
7. **Sales / Orders** – Create sales invoices (export, local), manage customer credit, track shipments.
8. **Expenses & Requests** – Manage expense and funding requests, approvals, attach receipts.
9. **Reports** – Cash summary, stock summary, yields, transactions, profitability, monthly statements.
10. **Admin Panel** – Manage users, roles, items, partners, locations, units; import/export data; view audit logs.
11. **Shark AI** – Chat assistant, risk alerts dashboard, weekly digest with explanation and links to records.
12. **CEO Control Panel** – Switch context (view-as / operate-as), review session logs and history.
13. **Location Manager**
14. Limited to data within their location; pages: Dashboard (location), Procurement & Receiving, Production, Inventory, Expenses & Requests (approve unit operator requests), Sales (local), Reports, Shark AI (location scope).
15. **Unit Operator**

16. **Home** (today's tasks), **Receiving** (record incoming raw material at unit), **Production** (record processing steps), **Inventory** (batch details), **Requests** (create expense/funding requests), **Sales** (local sales), **Shark AI** (only guidance, no global view).

17. **Investor / Read-Only**

18. **Dashboard**, **Reports**, **Audit Logs**, **Shark AI** (read-only), no write operations.

4.2 Final Firestore Schema (including boats)

`users`
- **uid** (doc id) — string
- **displayName**, **email**, **role_v2** (`HQ_ADMIN`, `LOC_MANAGER`, `UNIT_OP`, `INVESTOR`, etc.), **locationId**, **unitId**, **target_id** (for manager), **status** (`active`, `disabled`), **createdAt**, **lastLoginAt**.

`locations`
- **locationId** (doc id)
- **name**, **type** (`WAREHOUSE`, `OFFICE`, `FACTORY`, `LANDING_SITE`), **createdBy**, **createdAt**, **address**, **timezone**.

`locations/{locationId}/units` (sub-collection)
- **unitId** (doc id)
- **type** (`COLD_STORAGE`, `DRY_WAREHOUSE`, `PROCESSING_DRY`, `FROZEN_FACTORY`, `BOAT`), **name**, **capabilities** (`receiving`, `production`, `storage`, `sales`, `fuelLog`, `tripLog`), **parentLocationId**, **status**.

`items`
- **itemId** (doc id)
- **name**, **category**, **species**, **size**, **unitOfMeasure**, **active**, **createdBy**, **createdAt**.

`vendors` (replaces legacy partners)
- **vendorId**, **name**, **type** (`supplier`, `buyer`, `exporter`, `customer`), **contactInfo**, **bankInfo**, **active**.

`transactions`
- **transactionId** (doc id)
- **type** (`PURCHASE`, `PRODUCTION`, `SALE`, `LOCAL_SALE`, `EXPENSE`, `FUNDING`, `CASH_TRANSFER`, `LANDING`, `EXPORT_LOT`),
- **locationId**, **unitId**, **actorId**, **actorRole**, **mode** (`NORMAL`, `OPERATE_AS`), **items** (array of {itemId, qty, unitPrice, weight}), **inputBatchId** (for production), **outputBatchId**, **description**, **totalAmount**, **timestamp** (serverTimestamp), **auditRef**.

`site_wallets`
- **walletId** (doc id, e.g., `locationId_unitId`), **locationId**, **unitId**, **currency** (`IDR`), **balanceCash**, **balanceBank**, **lastUpdated**.

`financial_requests`
- **requestId**, **type** (`EXPENSE`, `FUNDING`), **amount**, **description**, **category**, **locationId**, **unitId**, **requesterId**, **status** (`PENDING`, `APPROVED`, `REJECTED`), **approverId**, **proofImageURL**, **history** (array of actions with timestamps).

`inventory`
- **doc id** (`locationId_unitId_itemId` or nested doc structure), **locationId**, **unitId**, **itemId**, **batchId**, **qty**, **weight**, **expiryDate**, **status** (`AVAILABLE`, `ON_HOLD`, `SOLD`), **lastUpdated**.

`production_batches`
- **batchId**, **locationId**, **unitId**, **inputItems** (raw materials), **outputItems** (finished goods), **startedAt**, **completedAt**, **yieldPercentage**, **notes**, **linkedTripId** (for boat landings), **createdBy**.

`boats`
- **boatId**, **name**, **registrationNo**, **ownerId**, **homePort**, **capacity**, **status** (`ACTIVE`, `MAINTENANCE`).

`boats/{boatId}/trips`
- **tripId**, **startDate**, **endDate**, **landingSiteId**, **crewList** (names or references), **fuelUsed**, **iceUsed**, **notes**, **status** (`ONGOING`, `LANDED`, `CLOSED`).

`boats/{boatId}/landings`
- **landingId**, **tripId**, **locationId** (landing site), **grossWeight**, **netWeight**, **speciesBreakdown**, **iceLeft**, **soldToUnitId**, **timestamp**.

`admin_notifications`
- **notificationId**, **type** (`ALERT`, `DIGEST`, `ERROR`), **level** (`INFO`, `WARNING`, `CRITICAL`), **message**, **relatedDocRef**, **createdAt**, **acknowledgedBy**.

`audit_events` (append-only)
- **eventId**, **actorId**, **actorRole**, **actorOriginalRole**, **mode**, **actionType**, **docRef**, **payloadSummary**, **timestamp** (serverTimestamp), **clientTimestamp**, **locationId**, **unitId**. Security rules: allow only `create` and deny `update` / `delete`.

Indexes should support queries by `locationId + unitId + timestamp`, `actorId + timestamp`, and `type` for efficient reporting.

4.3 Permissions Matrix

Role	Capabilities	Collections & actions
HQ_ADMIN (CEO / Global)	Full read/write across all locations and units; can context switch; can approve funding and expenses; can manage users, items, locations, vendors; can perform transactions on behalf of any unit.	Write to all collections through functions; manage `users`, `locations`, `units`, `items`, `vendors`; approve `financial_requests`; create `transactions` for any location/unit.

Role	Capabilities	Collections & actions
LOC_MANAGER	Limited to their assigned `locationId`. Can approve expenses for units in their location, view site wallets, perform receiving and production, create sales, manage local inventory, run reports.	Can call `postTransaction` with `locationId` scope; can create/approve `financial_requests` for units at location; cannot manage users or change items; cannot create new units.
UNIT_OP (Unit Operator)	Limited to assigned `unitId`. Can record receiving, production, sales and create expense requests; cannot approve expenses or fund transfers; cannot view other units.	Write limited to `transactions` (only with `unitId`); create `financial_requests` with type `EXPENSE`; read `inventory` and `wallet` for their unit.
BOAT_CAPTAIN / CREW *(new role)*	Record trip logs, fuel and ice consumption, crew advances, and landing notes. Cannot access financial modules.	Write to `boats/{boatId}/trips` and `boats/{boatId}/landings` via functions; read limited information.
INVESTOR / READ_ONLY	View dashboards, reports, Shark AI insights; cannot write.	Read on all non-restricted collections (`transactions`, `inventory`, `site_wallets`, `reports`, `audit_events`); no write access.

4.4 Audit Trail Specification

- **Every server-side function** must receive and record: `actorId`, `actorRole`, `actorOriginalRole` (in case of operate-as), `mode` (`NORMAL`, `OPERATE_AS`, `VIEW_AS`), `locationId`, `unitId`, `actionType`, `payloadSummary` (key metrics), and `clientTimestamp`.
- **Audit Events** are written to the `audit_events` collection using a Firestore trigger after each write. The trigger ensures logs cannot be tampered with by disallowing updates/deletes.
- **Important actions** to log: logins and logouts; context switches (view-as / operate-as); creation/approval/rejection of financial requests; transaction postings; edits in admin panel; modifications to user roles or units; seeding or cleanup operations.
- **Retention & export**: configure automatic export of `audit_events` to BigQuery / Cloud Storage for long-term storage. Provide export function for external auditors.

5 Prioritized Roadmap

Phase 1 – Safety, Integrity & Production Readiness

Task	Effort (S/M/L)	Risk	Dependencies	Success criteria
1. Harden Firestore rules: Deny direct writes to sensitive collections (`expenses`, `vendors`, `inventory`, `financial_requests`, `transactions`, etc.) and rely on functions. Remove legacy `isAdmin`.	M	Medium	None	All write attempts from client (other than via callable functions) are rejected. Security rules tested with emulator.
2. Enforce server-side view-as/ operate-as checks: Modify `postTransaction` and `financial_v2` functions to require `actorContext` fields and reject writes when `_isViewAs` is true. Add `ceoConfirmation` flag for operate-as.	M	High	1	Server functions cannot be abused to bypass UI. Audit demonstrates blocked writes when in view-as.
3. Build `createExpense` and `approveExpense` cloud functions; lock `expenses` collection.	M	Medium	1	Expense requests are created and approved only via functions, with location/unit validation and audit logging.
4. Implement append-only audit log: Add `audit_events` trigger, update functions to include full context, and restrict modifications.	L	Medium	2	Every action generates a corresponding log; logs cannot be edited/ deleted.
5. Standardize user schema & migration: Write migration script to update user docs to canonical fields; remove `loc` and `role`.	M	Low	None	`users` documents only have `role_v2`, `locationId`, `unitId`, `target_id`; functions rely on new fields.
6. Integrate write guard across all pages & replace browser confirms.	M	Low	2	No write can bypass `useWriteGuard`; all destructive actions use consistent modals.

Task	Effort (S/M/L)	Risk	Dependencies	Success criteria
7. Audit & fix number parsing: Use schema validation (Zod or Joi) in front-end and functions; add unit tests for numeric fields.	S	Low	None	No `NaN` or string numbers propagate to wallets or transactions.
8. Refactor queries to use server filters: Create API endpoints that return scoped data; remove client-side filtering.	M	Low	1	Managers and operators cannot see data outside their scope.

Phase 2 – Operational Power Features

Task	Effort (S/M/L)	Risk	Dependencies	Success criteria
9. Build structured inventory and production modules: Create `production_batches` and `inventory` sub-collections to track batches, yields and expiry. Add forms for splitting, merging, and re-grading inventory.	L	Medium	1	Production runs produce real-time inventory and yield reports; yields within threshold; negative quantities prevented.
10. Improve Shark AI: Develop scheduled functions that compute risk metrics (duplicate receiving, abnormal yields, wallet anomalies) and write `admin_notifications`. Extend chat to reference these records and provide explanations.	L	Medium	4	Weekly digests summarise key risks and KPIs; risk alerts triggered automatically; Shark responses include citations to DB records.
11. Implement multi-currency and bank deposits: Add currency field to wallets and transactions; integrate FX rates; support bank deposits/withdrawals.	M	Medium	1	Wallet balances correctly handle multiple currencies; cross-location funding uses conversion; bank ledger separate from cash.

Task	Effort (S/M/L)	Risk	Dependencies	Success criteria
12. Expand reports: Develop comprehensive dashboards (profitability, stock ageing, yield variance, cash flow) with export to Excel/PDF.	M	Low	1	Managers and investors receive clear, filterable reports; exports match financial ledger; trending charts available.

Phase 3 – Boats, Export & Advanced Automation

Task	Effort (S/M/L)	Risk	Dependencies	Success criteria
13. Add boats & landing site modules: Create `boats`, `trips` and `landings` collections; design UI for trip logging, crew payment, fuel & ice consumption; integrate with processing units for direct receiving.	L	Medium	9	Boat captains can record trips & landings; data flows to receiving and production modules; yields and payments calculated.
14. Support container shipments & export lots: Define `export_lots` entity with items, container numbers, documentation (B/L, packing list); integrate with sales and inventory.	L	Medium	9, 12	Export shipments tracked end-to-end with documentation; weight reconciled; profit per lot calculated.
15. Implement segregation of duties and approval hierarchy: Introduce dual approvals for large transactions, configurable thresholds, and CFO role.	M	Medium	4	Large expenses require two signatures; high-risk actions flagged; CFO oversees finance but cannot modify production data.
16. Continuous risk modelling: Use machine learning to identify fraud patterns, dynamic thresholds and recommendations. Ensure explainability by linking to transaction history.	L	High	10	Shark AI provides risk scores with explanations; false positives minimized; CFO and CEO get weekly digest of anomalies.

Task	Effort (S/M/L)	Risk	Dependencies	Success criteria
17. Prepare for external audit and compliance: Add audit exports to external storage; implement role training logs; align with Indonesian GAAP/IFRS and HACCP documentation requirements.	M	Medium	4, 12	Auditors can access read-only export; compliance documents (food safety, quality) integrated; system passes external audit.

6 Quick Wins (can be done today)

1. **Lock down open collections immediately** – update `firestore.rules` to deny all writes to `expenses`, `vendors`, `expense_types`, `messages` for non-admin roles. Only allow writes via callable functions.
2. **Enable server-side check for CEO context** – add a required `actorContext` field to `postTransaction` and `financial_v2` functions; reject if `_isViewAs` is true or missing confirmation.
3. **Remove browser confirm dialogs** – replace them with a unified confirmation modal across all pages.
4. **Fix user schema inconsistencies** – run a one-time migration to set `locationId` and `unitId` on all users and remove `loc` and `role` fields.
5. **Add missing audit logs** – create a `logAction` helper and call it in all existing functions (user edits, expense creation/approval, password resets).
6. **Improve number handling** – wrap all number inputs with parse and fallback; show error if NaN.

7 “Investor / Bank Grade” Gap List

To satisfy banks and institutional investors, the following gaps must be addressed:

1. **Governance & Segregation of Duties** – The system currently lacks a CFO role and threshold-based approvals. Banks require clear approval chains for payments and large expenses.
2. **Formal Policies and Documentation** – Provide a written SOP covering receiving, production, storage, quality assurance, finance, and approvals. Include HACCP plans, audit readiness kit, and code of conduct.
3. **Data Integrity & Auditability** – Implement immutable append-only logs, periodic backups, and the ability to produce audit reports that reconcile to bank statements and stock counts. Provide built-in checksums or signatures on transactions.
4. **Compliance with Accounting Standards** – Align financial data structures with Indonesian GAAP / IFRS; separate revenue, COGS, expenses, taxes; support multi-currency; record accruals and liabilities. Provide integration with external accounting software.
5. **Quality & HACCP Tracking** – Capture quality data (temperature logs, sanitation checks, lab tests) for each batch. Provide traceability from supplier to final product. Align with export certification requirements.
6. **Disaster Recovery & Business Continuity** – Document backup strategy; ensure Firestore backups and Cloud functions are replicated; provide offline forms for remote locations with low connectivity.

7. **Security & Privacy** – Conduct a third-party penetration test; enforce MFA for admin accounts; store sensitive documents (passport, bank details) encrypted. Implement logging and alerting on suspicious logins.
8. **User Training & Change Management** – Provide training materials and onboarding for operators, managers and boat captains. Maintain a training log and certification for compliance.

Conclusion

Ocean Pearl Ops V2 shows significant progress toward an integrated, multi-location seafood operations platform. Its architecture – React front-end with Firebase/Firestore backend and Cloud Functions – provides agility and offline capability. The new CEO control panel and write guard mark an important step towards safe context switching. However, for real-world operations with cash, inventory and regulatory requirements, the system must harden security rules, enforce server-side controls, expand audit logging and plan for new entities like boats and export lots. Following the roadmap and addressing the investor-grade gaps will elevate the platform to an international standard ready for scaling across Indonesia's islands and beyond.

---

1 raw.githubusercontent.com
https://raw.githubusercontent.com/tariqtharwat-OPS/ocean-pearl-ops/main/AUDIT_REMEDIATION_REPORT_2026_01_27.md

2 raw.githubusercontent.com
https://raw.githubusercontent.com/tariqtharwat-OPS/ocean-pearl-ops/main/firestore.rules

3 raw.githubusercontent.com
https://raw.githubusercontent.com/tariqtharwat-OPS/ocean-pearl-ops/main/functions/index.js

4 OPS — VIEW AS: KAIMANA / UNIT OP
https://oceanpearl-ops.web.app/admin

5 OPS — VIEW AS: KAIMANA / UNIT OP
https://oceanpearl-ops.web.app/reports

6 raw.githubusercontent.com
https://raw.githubusercontent.com/tariqtharwat-OPS/ocean-pearl-ops/main/src/lib/constants.js

7 OPS V2 Audit Checklist
https://raw.githubusercontent.com/tariqtharwat-OPS/ocean-pearl-ops/main/OPS%20V2%20Audit%20Checklist.pdf

8 10 12 raw.githubusercontent.com
https://raw.githubusercontent.com/tariqtharwat-OPS/ocean-pearl-ops/main/functions/financial_v2.js

9 raw.githubusercontent.com
https://raw.githubusercontent.com/tariqtharwat-OPS/ocean-pearl-ops/main/CEO_CONTROL_MODE_STATUS.md

11 raw.githubusercontent.com
https://raw.githubusercontent.com/tariqtharwat-OPS/ocean-pearl-ops/main/OPS_ROLE_SMOKE_TEST_REPORT.md

13 raw.githubusercontent.com
https://raw.githubusercontent.com/tariqtharwat-OPS/ocean-pearl-ops/main/src/components/OperateAsConfirmation.jsx

14 raw.githubusercontent.com
https://raw.githubusercontent.com/tariqtharwat-OPS/ocean-pearl-ops/main/OPS_SHARK_QA_PROOF.md