Audit of Finanzas Module Implementation vs. Design

Finanzas SD – Architecture, Flows & SOPs

Arquitectura, Flujos y Procedimientos

January 25, 2026

1 Audit of Finanzas Module Implementation vs. Design

1.1 Repository Structure & Component Alignment

The financial-planning-u repository closely follows the planned architecture. It is organized into a front-end (React/Vite app) and a back-end (AWS SAM stack under services/finanzas-api). Key components match the design in finanzas-architecture.md:

1.1.1 CloudFront & S3 (UI Delivery)

The static UI is built with a base path of `/finanzas/` and deployed to an S3 bucket. A CloudFront distribution serves it under the path pattern `/finanzas/*` as intended. The CI pipeline confirms the CloudFront behavior is in place, failing the deploy if the `/finanzas/*` route is missing. After deployment, an invalidation of `/finanzas/*` ensures users get the latest assets.

Note: Initially, the CloudFront behavior had to be added manually – a guard in the workflow now checks and warns if it's not configured. The bucket is private with public access blocked; instructions call for an Origin Access Control policy so CloudFront can fetch the files. This should be verified to avoid 403 errors.

1.1.2 API Gateway (finanzas-sd-api)

The back-end is defined via a SAM template. It provisions an HTTP API with a default dev stage (and supports a prod stage via parameter). The API uses a Cognito JWT Authorizer configured with the shared User Pool and client ID. CORS is locked down to the CloudFront domain (`d7t9x3j66yd8k.cloudfront.net`) as expected.

All expected endpoints from the architecture are present or stubbed in the template:

- Public endpoints: `/health`, `/catalog/rubros`
- Protected routes: projects, handoff, rubros, allocations, plan, payroll, close-month, adjustments, alerts, providers, etc.

There are even endpoints for the prefacturas webhook (`GET/POST /prefacturas/webhook`) not explicitly in the original doc, aligning with the formal proposal's pre-factura module.

DynamoDB tables for all data domains (`finz_projects`, `finz_rubros`, `finz_allocations`, etc.) are defined with the correct naming and on-demand billing. One addition is `finz_rubros_taxonomia` for the rubros catalog taxonomy, which wasn't named in the architecture but is part of the rubros catalog model. Overall, the infrastructure code aligns well with the design and R1 scope.

1.1.3 Lambdas & Handlers

Each API route is backed by a Lambda function, built in Node.js 20 as specified. The repository includes stub implementations for most handlers. For example, the Projects handler supports creating and listing projects in DynamoDB (with a simple PK/SK design). The Rubros handler is stubbed (returns 501 or an empty list) indicating that feature is not fully implemented yet.

This matches the execution plan where some endpoints in R1 are MVP placeholders. The presence of these stubs means the structure is in place, but certain business logic (e.g., adjustments, rubros per project) is slated for completion in upcoming releases. The Lambda environment variables and IAM policies follow least-privilege principles (each function gets access only to its necessary table, etc.), consistent with the security model.

1.1.4 AWS Cognito Integration

The module uses the shared Cognito User Pool (`us-east-2_FyHLt0hiY`) and a dedicated App Client for Finanzas as planned. The API's JWT authorizer is configured with the User Pool issuer and the Finanzas client's ID as the audience.

On the front-end, Amplify is nominally configured via `src/config/aws.ts` but in practice the implementation is mostly custom. The config correctly references the user pool ID, client ID, and the Cognito Hosted UI domain (a missing hyphen was fixed in the domain string).

Allowed OAuth scopes (`email`, `openid`, `profile`) and redirect URIs are set to use the `/finanzas/auth/callback.html` page for sign-in, and `/finanzas/` for sign-out. This matches the design of isolating the Finanzas auth flow from the main app.

Note: The Cognito hosted UI callback URLs must be configured in the User Pool settings for both the production CloudFront URL and localhost (for dev) – the configure-cognito-hosted-ui.sh script likely ensured this. Any misconfiguration here (e.g., missing the `/finanzas/auth/callback.html` in allowed callbacks) would break the hosted login; it appears this was addressed.

1.2 Authentication Flow & Post-Login Behavior

1.2.1 Login & Token Handling

The front-end provides two login methods: direct username/password and Cognito Hosted UI. The custom AuthProvider context implements the USER_PASSWORD_AUTH flow by calling Cognito's InitiateAuth API directly (avoiding the full Amplify library).

On success, it stores the ID token in localStorage under both `cv.jwt` and `finz_jwt` keys (for cross-module compatibility). A refresh token is also stored (`finz_refresh_token`), although no automatic token refresh logic is implemented yet.

After a direct sign-in, the app triggers a role-based redirect: in a Finanzas-only build it always sends the user to the Finanzas home (`/finanzas/`), otherwise it chooses between the main dashboard or Finanzas based on the user's Cognito groups. This logic aligns with the RBAC expectations that Service Delivery users land in the Finanzas module by default.

For the hosted UI flow, the app constructs the correct `/login` URL for the Cognito domain (including the `client_id`, scopes, and `redirect_uri`) and performs a redirect. The hosted UI then returns to the static callback page. The repository includes an `auth/callback.html` in the build output. That page's script parses the Cognito token from the URL fragment and likely calls a small script to store the JWT and then redirect the user to the app. According to the test plan, the callback page shows "Signing you in…" and then navigates to `/finanzas/`, which matches expected behavior.

Action: We should double-check that the callback page sets `cv.jwt` properly – the direct login code does this, and the callback should mimic it.

The logout flow is also handled: `logoutWithHostedUI()` in config clears local storage and redirects to the Cognito `/logout` URL with the app's sign-out URI. This ensures a clean logout from both the app and the IdP.

1.2.2 RBAC Enforcement

On the back-end, all protected Lambdas call `ensureSDT(event)` at the top to enforce that the user's Cognito groups include the Service Delivery team group. If the token has no groups or is missing "SDT", the function throws a 403 error (`"forbidden: SDT required"`). This is exactly as described in the architecture (group-based authorization on top of JWT validation).

Inconsistency: The front-end logic treats several group names as equivalent to finance access – it checks for "SDT", "FIN", "AUD", etc. in the user's groups to decide if a user can access Finanzas. However, the back-end `ensureSDT` only accepts "SDT" explicitly. If some users are placed in a "FIN" or "AUD" Cognito group expecting to grant Finanzas access, the current Lambda check would reject them (403).

This misalignment should be corrected by either standardizing on one group (e.g., use "SDT" for all finance users) or expanding `ensureSDT` to recognize the additional finance roles. For now, any user not literally in the SDT group will fail to call protected APIs, even though the UI might think they're authorized.

1.2.3 Front-End Route-Level Access

On the front-end, route-level access is handled by the AccessControl component. It uses the user's current role and the defined route permission map to either allow navigation or redirect the user.

For example, a user with role SDMT (Service Delivery) or PMO will be allowed to access the Finanzas routes (`/finanzas/catalog/**`, `/finanzas/rules`) because the permissions config includes those paths for both roles. If a user somehow navigates to an unauthorized route, AccessControl will redirect them to a more appropriate landing page (using `getDefaultRouteForRole()` which maps roles to their default module page).

This mechanism was meant to support dual-role users: e.g., if a PMO-only user hits a Finanzas URL, they get sent to the PMO home, and vice versa. In practice, since R1 is focusing on the Finanzas module, most users testing this should have SDT/FIN roles.

Potential Issue: If a dual-role user had previously set their preference to "pmo" (stored in `cv.module` localStorage) and then logs in, the AuthProvider might redirect them to the root (`/`). If the main PMO app is not actually deployed or integrated at that same domain, the user could land on a blank page or a CloudFront error. The integration plan does anticipate deploying the PMO portal on the same distribution (the deploy workflow builds both UIs), so if that was done, the root path would load the PMO app.

We should ensure that in production, both the PMO base and Finanzas sub-app are live, so that these role-based redirects always land on a valid page. If the PMO module isn't actually available yet, one mitigation is to force Finanzas users to stay in `/finanzas` (the code already does this for Finanzas-only builds). This is working now (Finanzas build sets `VITE_FINZ_ENABLED=true`, so the app will always use the FinanzasHome as `/` route), but when the modules integrate fully, we must retest the cross-module navigation.

1.2.4 Post-Login UI Behavior

Once logged in, a Finanzas user is presented with the FinanzasHome dashboard. We verified this component shows the expected content: it has a header and two primary links for "Catálogo de Rubros" and "Reglas de Asignación". This corresponds to the R1 scope (the initial finance features). Clicking those links navigates within the SPA to the rubros catalog page or the allocation rules page. The routes are set up under the `/finanzas` basename so that deep-linking works.

The test plan specifically checks that accessing a deep link like `/finanzas/catalog/rubros` directly works without a 404. For this to succeed, CloudFront should be configured to redirect 404s to the SPA index. It's not explicitly scripted in IaC, so presumably a

custom error response for `404 -> /finanzas/index.html` was configured on the distribution. We should ensure this is done; otherwise a direct refresh on a nested route might yield a CloudFront 404. In practice, QA reported that deep links are loading the app (no 404), indicating this is either configured or CloudFront's default root object catches it.

Once the rubros catalog page loads, it should fetch data from the API. The `/catalog/rubros` endpoint is public (no auth) and returns the list of ~71 rubro entries seeded in the `finz_rubros_taxonomia` table. This is working: the CI seeding step populates that table from a static catalog (the "Catálogo de Rubros") and a smoke test confirms the API responds with data. The UI likely calls this via a fetch or Amplify API call – since the config defines an API endpoint named "FinanzasAPI" pointing to the execute-api URL, it might be using Amplify's API module.

In testing, no CORS issues were observed – the API's CORS is restricted to the CloudFront origin and the CloudFront URL is what the browser uses, so requests succeed with a proper Authorization header for protected calls and a 200 OK for the catalog request. This indicates the end-to-end flow (UI → API → DynamoDB) is functioning for at least the catalog data.

1.3 Deployment Workflows & Stage Management

1.3.1 deploy-api.yml

Deploying the SAM stack is gated on specific branches (`module/finanzas-api-mvp`, `r1-finanzas-dev-wiring`, and `main`). The workflow uses OIDC to assume an AWS role (no static credentials), satisfying the security guardrail of no long-lived keys. It sets the stack name and stage name based on the branch: for `main` it uses `finanzas-sd-api-prod` and `StageName=prod`, otherwise `finanzas-sd-api-dev` and `StageName=dev`. This means we do have a separate prod stage when pushing to `main`, as intended by the architecture's multi-stage strategy.

After build and deploy, the workflow performs extensive post-deployment validation:

- It retrieves the API's ID and base URL from CloudFormation outputs.
- It checks that the API ID matches an expected value (to ensure we're deploying to the correct environment). **Note:** The expected ID is configured for dev by default (`m3g6am67aj`). We must update the CI variables if the prod API has a different ID, or else the check will flag a mismatch in prod.
- It calls `aws apigatewayv2 get-routes` to verify critical routes exist (`GET /health`, `GET /catalog/rubros`, `POST /projects`) and that the Cognito JWT authorizer is

present. This would catch any regression where a route was accidentally removed or the authorizer not attached.
- It waits for the `/health` endpoint to return 200 OK and then performs smoke tests on a protected endpoint. The workflow uses a test Cognito user (credentials stored in GitHub secrets) to generate a JWT and calls `POST /projects` with a sample payload. This tests end-to-end authentication and basic function of a state-changing API.
- It also tests `GET /catalog/rubros` (should succeed with 200) and even checks that `GET /adjustments` returns a 501 Not Implemented (which is the current expected behavior for that stub).
- Logging from the Lambda (e.g., CatalogFn) is pulled in if a test fails, to aid debugging.

Finally, the API workflow seeds the DynamoDB tables for rubros taxonomy (and possibly an initial rubros set) using a TypeScript seed script. It runs `seed_rubros_taxonomia.ts` via `ts-node` to upsert the taxonomy entries. This ensures that right after deployment, the reference data is in place (so the catalog API immediately has data to return). This is a great inclusion for test/QA readiness.

Summary: deploy-api.yml covers post-deployment validation thoroughly, catching missing routes or auth and verifying that both public and secured endpoints respond as expected. This would have caught many configuration errors (for instance, if CORS was wrong or the authorizer wasn't working, the POST with a JWT would likely fail and be noticed).

1.3.2 deploy-ui.yml

There are two UI deployment workflows in the repo; the newer one (Deploy UIs (PMO + Finanzas)) handles both modules. It triggers on pushes to `main` or the integration branch and builds two versions of the app: one with `BUILD_TARGET=pmo` (base path `/`, output `dist-pmo`) and one with `BUILD_TARGET=finanzas` (base `/finanzas/`, output `dist-finanzas`). This matches the planned monorepo approach where the PMO portal and Finanzas portal are built together.

The workflow then uploads the PMO build to the S3 bucket root and the Finanzas build to the `finanzas/` prefix in the bucket. It explicitly checks that the `/finanzas/*` behavior exists in CloudFront (failing with instructions if not) and confirms that the behavior's target origin is the correct S3 origin (`finanzas-ui-s3`). After uploading, it creates a CloudFront invalidation for both `/*` and `/finanzas/*` (to refresh both root and Finanzas paths).

Post-Deploy Smoke Tests:

- It cURLs the CloudFront domain's root (`/`) and `/finanzas/` and records the HTTP

status. The summary shows these URLs and whether they returned 200. For example, a successful run would confirm `https://d7t9x3j66yd8k.cloudfront.net/finanzas/` is reachable (if CloudFront wasn't configured or S3 content missing, this would fail).

- It fetches the first ~50 lines of the Finanzas page HTML and checks that asset links are correctly prefixed with `/finanzas/`. This is crucial: in early iterations, a common bug is forgetting to set the base href, causing the SPA to load assets from root path. The guard here ensures the built HTML contains references like `/finanzas/assets/[hash].js`.
- It also greps the HTML to ensure no unexpected references to GitHub domains or Codespaces remain (just a safety check to catch dev environment URLs that shouldn't be in prod).
- If the Finanzas page is not accessible (non-200), it logs an error in the summary. This would catch, for example, CloudFront permission issues (like OAC not configured, causing 403) or distribution misrouting.

Together, these validations mean the CI/CD would catch major regressions: e.g., if a future change broke the login page from loading, or if the API wasn't reachable. The UI's env `VITE_API_BASE_URL` is set in the workflow based on branch to point to dev or prod API URL, so it should always be correct; a mismatch would likely show up as runtime API errors rather than failing the build, but the manual QA would notice.

API URL Configuration: One thing to confirm is that the prod API URL is correctly wired into the prod UI build. The workflow sets `VITE_API_BASE_URL` to the appropriate execute-api URL for the environment. For `main` branch, it uses the prod API (either via a provided `DEV_API_URL` var or defaulting to the known prod URL). We should ensure the actual API ID for prod is plugged in. If the prod API stack was deployed for the first time, it has a new API ID (not `m3g6am67aj`). The CI's logic tries to use a `DEV_API_URL` var for non-main, and for main it falls back to a hardcoded prod URL with the same ID if not overridden.

Recommendation: Define the real prod API ID in GitHub variables (or have the API deploy job output it somewhere the UI job can pick up) to avoid the UI accidentally pointing at the dev API. Right now, if not updated, the prod UI might still call the dev stage API.

1.4 Findings and Required Corrections

In summary, the implementation aligns well with the intended architecture and R1 functional goals. The core infrastructure (API Gateway, Lambdas, Cognito, CloudFront,

DynamoDB) is in place as designed, and the front-end is wired into it with authentication and routing as expected. The CI/CD pipelines add confidence by automatically catching misconfigurations or regressions.

We have identified a few misalignments and areas to improve:

1.4.1 1. Cognito Group Enforcement

Issue: Ensure consistency in allowed groups for Finanzas. Currently, Lambdas only accept the "SDT" group, but the front-end expects users with FIN or AUD groups to also have access.

Action: If the intention is to include those groups, update `ensureSDT` to recognize them (or decide on a single canonical group). This will prevent finance users from being mistakenly blocked with 403 Forbidden.

1.4.2 2. API Stage Usage

Issue: Confirm that the production UI is pointing to the prod API stage. The config uses environment variables to switch the base URL, so we must verify that `VITE_API_BASE_URL` was set to the prod API's URL during the production deployment. If not, update the CI variables (`FINZ_API_ID_PROD` or `DEV_API_URL` for main) so that the Finanzas UI in prod isn't inadvertently calling the dev stage.

Action: The deploy workflows are designed to handle this, so it's likely a matter of populating the correct value.

1.4.3 3. CloudFront Configuration

Issue: Although the pipeline enforces the `/finanzas/*` behavior now, we should double-check a couple of CloudFront settings:

a) Origin Access Control (OAC) for S3

- The bucket policy must allow the CloudFront distribution to read files. The setup script notes this as a manual step. If this was missed, users would see HTTP 403 on static assets. This is critical to fix immediately if not already done.

b) Custom Error Responses

- To support SPA client-side routing, configure CloudFront to respond to 404/403 for `/finanzas/*` by returning `/finanzas/index.html` (with a 200). Otherwise, a deep link refresh might fail. The QA expectations imply this was desired. Implement this in CloudFront (or via IaC if possible) and test that navigating directly to a nested route works (we want a 200 with the SPA content, not a CloudFront error). This may already be in place, but if any testers encountered a blank page on refresh, it's likely due to this and should be addressed.

1.4.4 4. Finanzas UI Post-Login Routing

Issue: The FinanzasHome and its links are appearing as expected after login, but there was mention of a "lack of Finanzas view post-login". This could have been due to the dual-role redirect issue or the CloudFront path issue.

Actions: Once the above CloudFront fixes are in place, verify that:

- A user with only Finanzas roles goes straight to the FinanzasHome page after login (the current code does this by default).
- A dual-role user (SDT + PMO) follows the intended logic: first login goes to Finanzas by default; if they prefer PMO and we have PMO app deployed, they can be sent to `/` on next login. If the PMO app is not fully ready, consider overriding the redirect to keep them in Finanzas for now to avoid a bad landing page. Also ensure the `cv.module` preference is respected – the code writes it on role switch and reads it on login, which seems correct.
- The FinanzasHome page properly links to subpages and those subpages fetch data without errors. For example, the Rubros catalog page should show the list of rubros from DynamoDB (the data seeding suggests ~71 entries, which matches test expectations). If the page is blank or throwing an error, it might be an issue with the front-end data fetching logic that needs fixing. So far, tests indicate it was successful (the API returned data), so likely the UI is fine, but it's worth a quick manual test in the deployed env.

1.4.5 5. Refresh Token & Session Longevity

Issue: While not an immediate R1 requirement, note that the app does store a refresh token but doesn't use it yet. This means sessions will expire after ~1 hour (the Cognito access token lifespan).

Action: Users will then need to log in again manually. To improve user experience, implement a refresh token flow or use Cognito's hosted UI with the Authorization Code grant (which can be exchanged for fresh tokens silently). This could be an R2 enhancement. For now, document this behavior so users know they must re-authenticate periodically.

1.4.6 6. Prefacturas Webhook Auth

Issue: The prefacturas/webhook endpoints are currently protected by the Cognito authorizer. If the external system that will call this webhook cannot easily supply a Cognito JWT, we may need to adjust this.

Action: Options include making a separate API key or a signed token just for that endpoint, or documenting how the external service can obtain a JWT. Since this was in scope for R1 (basic webhook), consider this: if the design assumed an internal call

or a service with a Cognito identity, then it's okay. Otherwise, this could be a misconfiguration – external webhooks typically aren't authenticated with end-user JWTs. We should clarify this expectation and adjust if needed (e.g., allow an API key or switch Auth to NONE on that route if it's meant to be unauthenticated with a verification token in payload instead).

1.4.7 7. Completing Stubbed Features

Issue: As we progress, the stubbed endpoints (e.g., Adjustments, Rubros POST, Plan calculation, etc.) need to be implemented. From an architectural perspective, the stubs are fine for R1 (since R1 was mainly about laying the groundwork).

Keep these in the tracker:

- `POST /projects/{id}/rubros` should insert a rubro into the `finz_rubros` table and possibly update related allocations.
- `GET /projects/{id}/rubros` should read a project's rubros (currently returns empty list).
- Adjustments, Plan, Close-Month – ensure their logic is implemented or at least returns proper not-implemented responses. Right now, the pipeline expects `GET /adjustments` to return 501, which is acceptable as a placeholder. Just don't forget to remove these placeholders by R2.
- Alerts (PEP-3) – architecture mentions a monthly EventBridge rule for budget alerts. The AlertsPep3Fn is currently just a stub (re-using the health function). That's fine for now (backlog item), but we should mark it clearly so it's not overlooked. No user-facing impact yet, as it's not exposed in UI.

1.4.8 8. Monitoring & Alarms

Issue: R1 included observability. The architecture called for CloudWatch alarms on 5xx errors, etc., and maybe a synthetic canary. It's not clear if those were implemented (there's mention of a separate template-obs.yaml in the plan, possibly not done yet).

Action: We should confirm if any CloudWatch alarms exist for the new APIs and if not, plan to add them in the "Observability & Alarms" milestone (M5). The pipeline does set up an API access log group, which is good for debugging. No immediate user impact, but for completeness, ensure logs and X-Ray (which is enabled in SAM globals) are being used to monitor the system.

1.5 R1 Milestone Status Summary

The primary goal was "MVP backend + UI wiring, tested in dev without regressions". We have achieved the following in R1:

- A working authentication flow (with both direct and SSO login)
- A deployed static UI accessible via CloudFront
- An API with all endpoints defined (some implemented, some stubbed)
- Integration between the two for at least the initial features (rubros catalog & viewing rules)
- CI pipelines and testing artifacts (GitHub summary, etc.) serve as the evidence pack

There were a few hiccups (Cognito domain typo, CloudFront config, env var wiring) that the team addressed along the way. With the fixes noted above, the Finanzas module is on track for R1: the architecture is sound and matches the proposal, and the foundational pieces are in place.

1.5.1 Next Steps

1. Fill in the remaining functionality (per the execution plan milestones M3, M4)
2. Refine any rough edges in auth and routing
3. Address the findings above to fully align with the intended design

1.5.2 Conclusion

R1's deliverables are largely met – we have a functioning end-to-end slice – and with the minor corrections outlined, the system will fully align with the intended design and be ready for user acceptance testing.