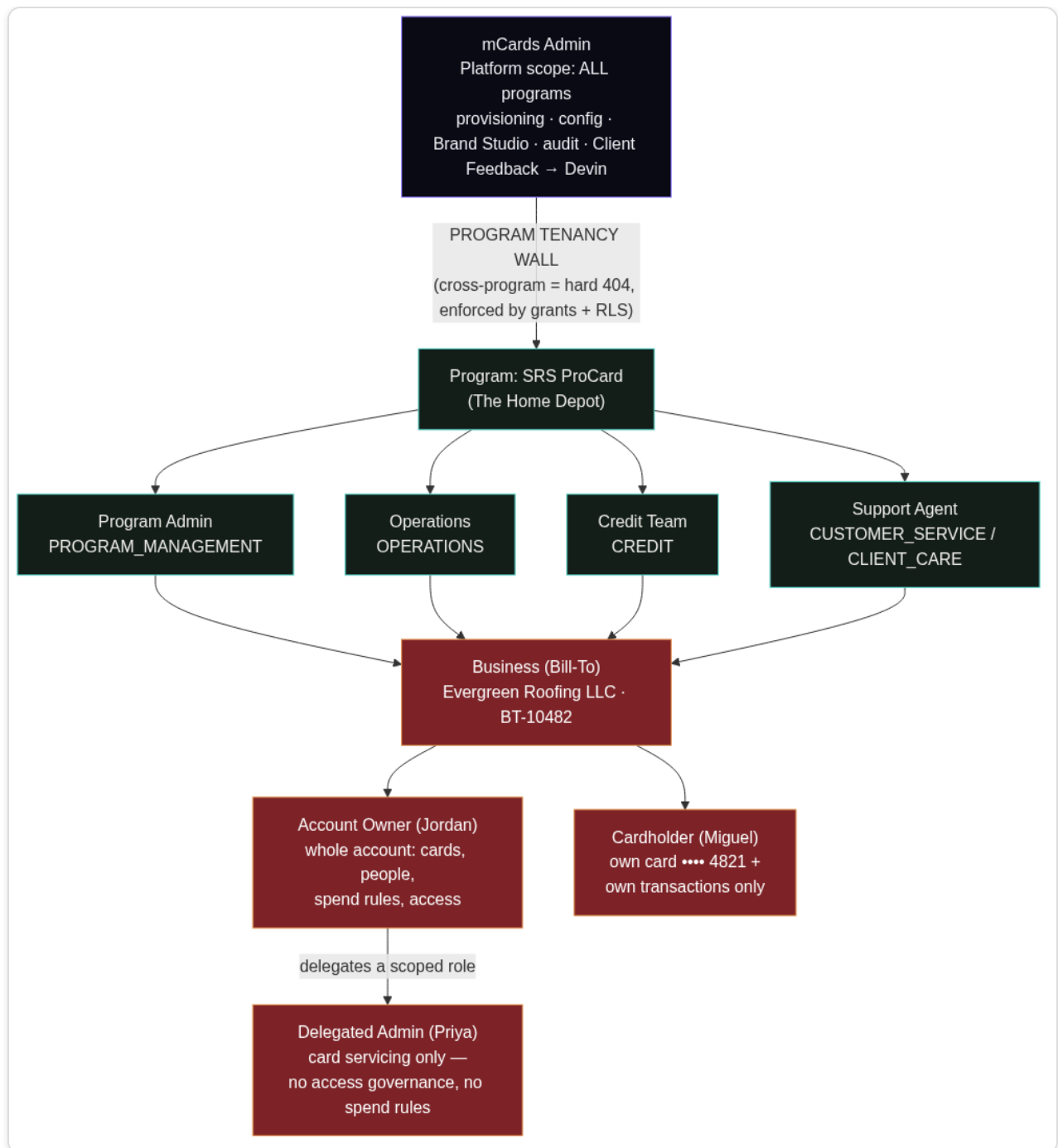


SRS ProCard Experience Platform — Access & Portal

Architecture

Audience: technology teams (SRS, Home Depot, mCards) evaluating how the platform is structured. **Scope:** user types, portals, and how visibility, controls and capabilities cascade from configuration down to every screen and API call.

1. The one-sentence design

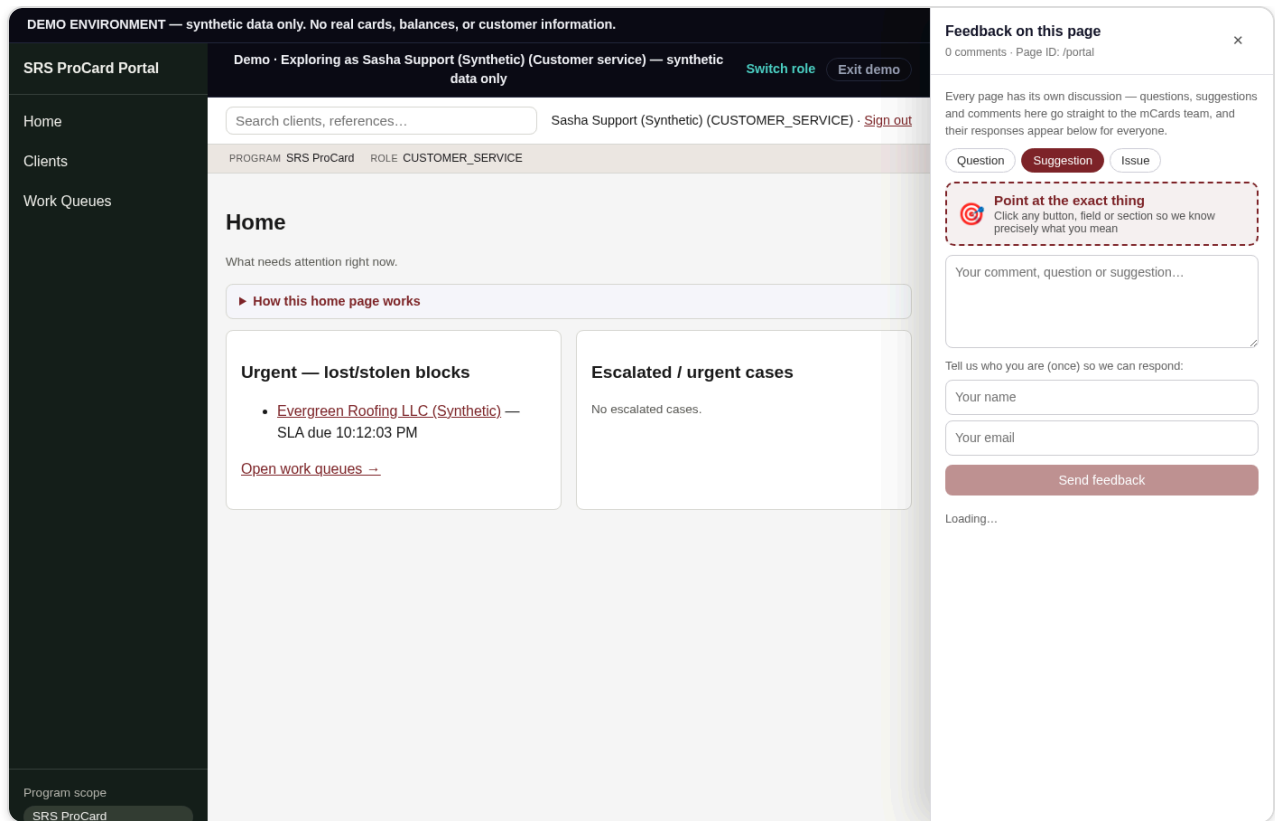


Tenancy and user hierarchy: platform → program wall → business → owner/delegate/cardholder.

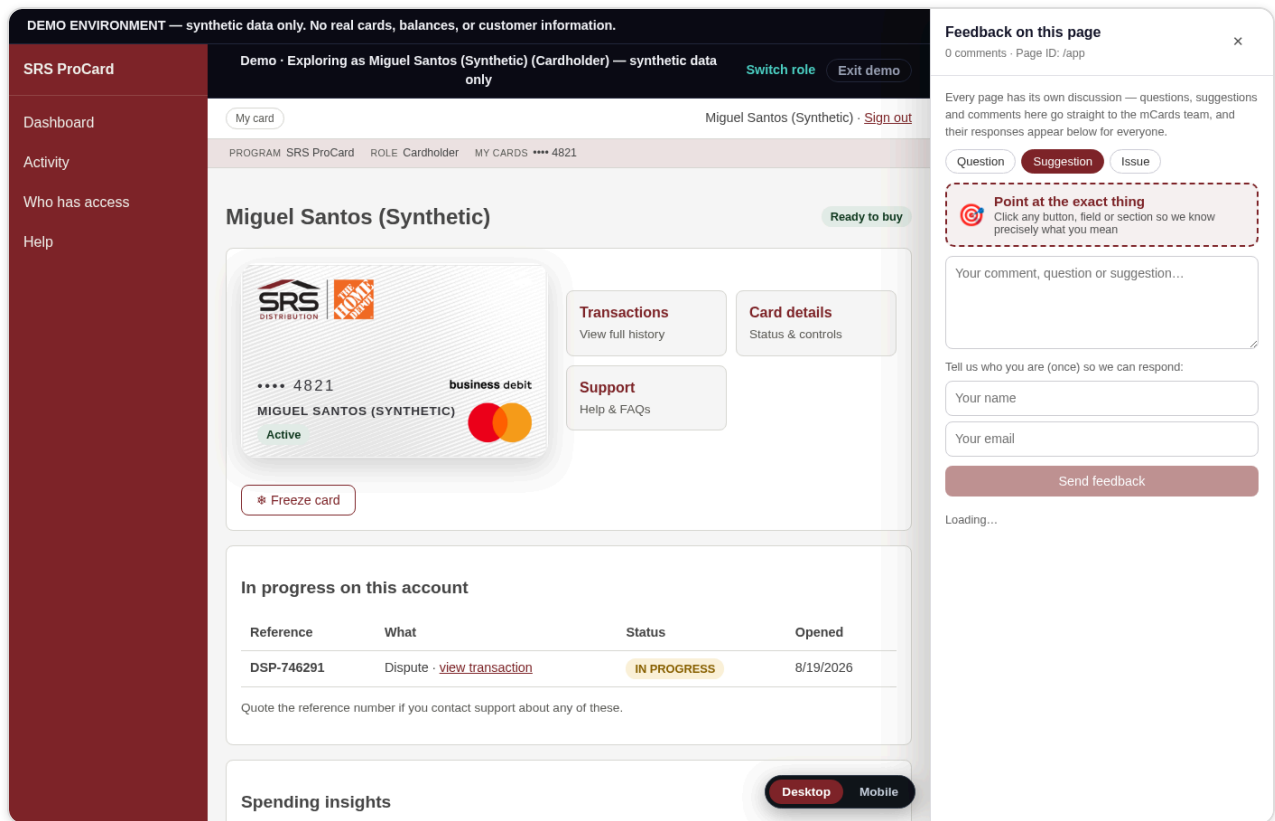
Configure the program once. Resolve each user's identity, role, and account context at sign-in. Present only the data and actions that role needs. Enforce the same scope again at the API layer. Record every sensitive action and its outcome.

Everything below is that sentence, drawn out.

2. The portal landscape



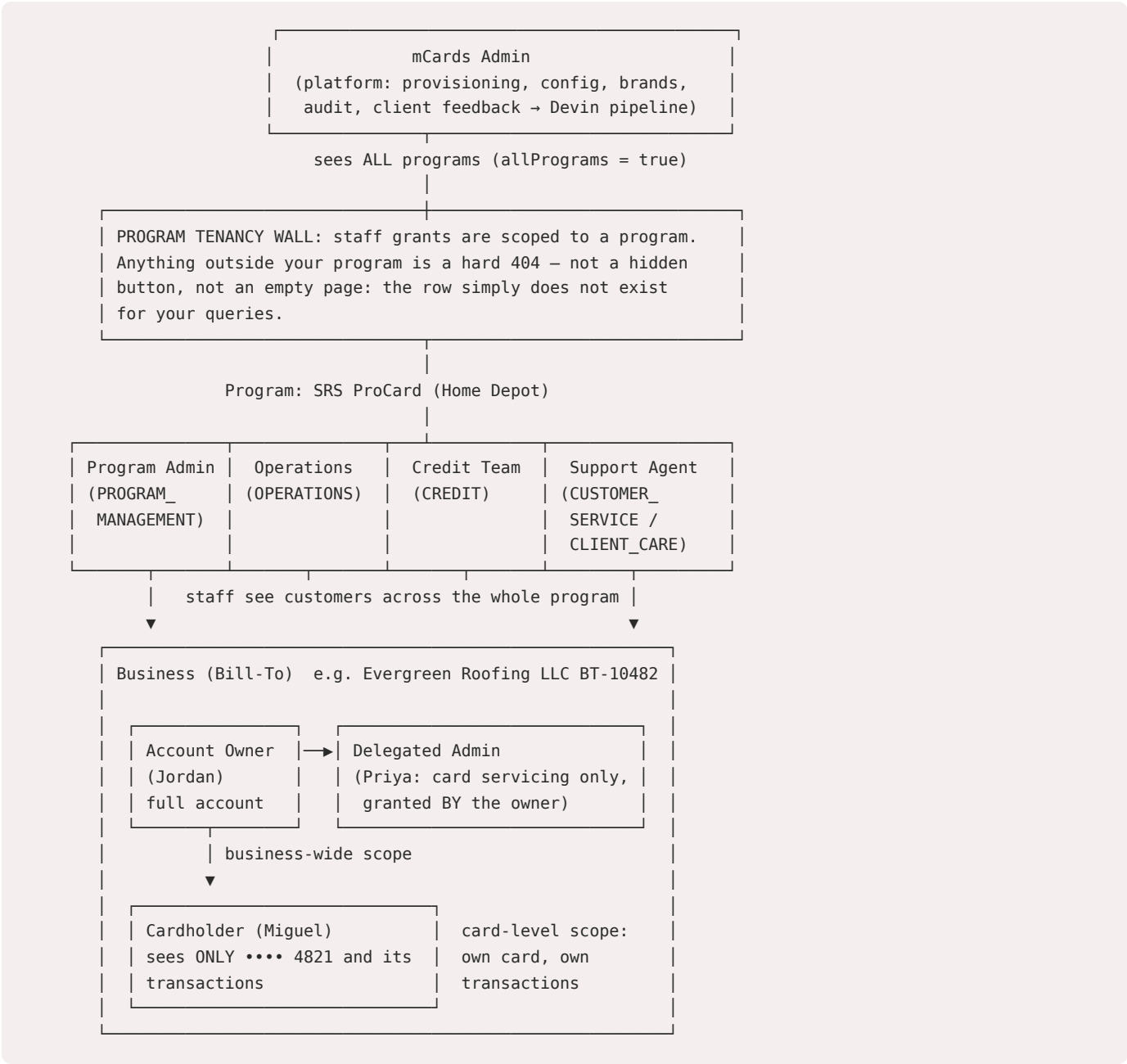
The unified staff portal (/portal) — role-filtered left navigation, program-scoped data.



The cardholder experience (/app) — Miguel sees only his own SRS ProCard, activity and help.

One codebase serves three surfaces. Which surface (and which navigation inside it) a user sees is decided entirely by their resolved roles — there are no separate deployments per audience.

Surface	Route	Who lands there	What it is
Customer app	/app	Account Owner, Delegated Admin, Cardholder	Branded self-service: card, credit line, activity, cards & people, delivery locations, access management, help
Staff portal	/portal	Support Agent, Credit Team, Operations, Program Admin	Clients workspace, work queues, credit, operations reports
mCards platform	/portal/platform + /portal/platform/feedback	mCards Admin only	Program provisioning, Brand Studio, config, audit, Client Feedback → Devin pipeline



3. Roles and the capability cascade

DEMO ENVIRONMENT — synthetic data only. No real cards, balances, or customer information.

SRS ProCard

Dashboard

Activity

Cards & people

Delivery locations

Who has access

Help

Demo · Exploring as Jordan Lee (Synthetic) (Account owner, Cardholder) — synthetic data only

Switch role

Exit demo

Account access

My card

Jordan Lee (Synthetic) · Sign out

PROGRAM SRS ProCard · ROLE Account Owner · BILL-TO Evergreen Roofing LLC (Synthetic) · BT-10482 · MY CARDS **** 9016

Evergreen Roofing LLC (Synthetic)

Ready to buy · 1 active card

SRS DISTRIBUTION

THE HOME DEPOT

**** 9016

JORDAN LEE (SYNTHETIC)

Block in progress

business debit

Transactions

View full history

Card details

Status & controls

Cards & people

Issue cards & set limits

Who has access

Invite & manage users

Delivery

Ship-to locations

Support

Help & FAQs

Available to spend

\$107,235.50

Credit line \$150,000.00 · Balance \$42,764.50

Balance last updated 8/19/2026, 9:13:50 PM — figures may not reflect recent activity.

To make a payment, use your existing SRS trade credit payment process. Payments made today may take up to one business day to reflect here.

Recent activity

Merchant	Amount	Status
THD #4712 — The Home Depot (Synthetic)	\$1,284.60	PENDING
Insulation Depot (Synthetic)	\$412.00	PENDING
SRS Distribution — Branch 118 (Synthetic)	\$2,641.10	PENDING

Desktop Mobile

Feedback on this page

0 comments · Page ID: /app

Every page has its own discussion — questions, suggestions and comments here go straight to the mCards team, and their responses appear below for everyone.

Question

Suggestion

Issue

Point at the exact thing

Click any button, field or section so we know precisely what you mean

Your comment, question or suggestion...

Commenting as David Strebinger - david@mcards.com

Send feedback

Loading...

Account Owner (Jordan) — whole-account command center: shared credit line, cards, people.



The approved SRS ProCard — the artwork used everywhere a card is rendered.

Access is not assigned screen-by-screen. Each role maps to a **capability set**, and staff capability sets are strictly nested — each tier is the previous tier plus more. This is deliberate: an Operations analyst can always do what a Support agent can, so escalation never means "switch accounts."

3.1 Customer-side roles (scope: one business or one card)

CARDHOLDER (own card only)	AC_VIEW_ONLY (whole business, read-only)	AC_CARD_SERVICING (= view + act on cards)	ACCOUNT_OWNER (= servicing + govern the account)
- VIEW_OWN_CARD	- VIEW_DASHBOARD	+ FREEZE / UNFREEZE	+ MANAGE_ACCOUNT_ACCESS
- VIEW_OWN_TRANSACTIONS	- VIEW_CREDIT_LINE	+ REPORT_LOST_STOLEN	+ MANAGE_CARD_RULES
- FREEZE / UNFREEZE (own)	- VIEW_CARDS	+ REQUEST_REPLACEMENT	(issue cards, set per-card spend limits
- REPORT_LOST_STOLEN	- VIEW_TRANSACTIONS	+ REPORT_TXN_PROBLEM	against the shared credit line)
- REQUEST_REPLACEMENT	- VIEW_ACCESS_LIST	+ REQUEST_NEW_CARDHOLDER	
- REPORT_TXN_PROBLEM	- VIEW_SHIP_TOS	+ REQUEST_SHIP_TO	

- **Cardholder** is intentionally *not* a subset of the account roles: it is a different axis. A cardholder's scope is a card, not a business. Miguel sees his `**** 4821` and nothing about Evergreen's credit line or other employees.
- **Delegated Admin (AC_CARD_SERVICING)** exists so the owner can hand off day-to-day card servicing without handing over account governance. Priya can freeze cards and request replacements; she cannot change who has access or set spend rules. The boundary is shown in the UI ("what you can't do") *and* enforced server-side.
- **Account Owner** additionally governs the account: invite/remove users, issue cards, set per-card spend limits that draw on the shared credit line.

3.2 Staff roles (scope: their program(s), all customers within)

Strictly nested — each row is the previous row **plus** the listed additions:

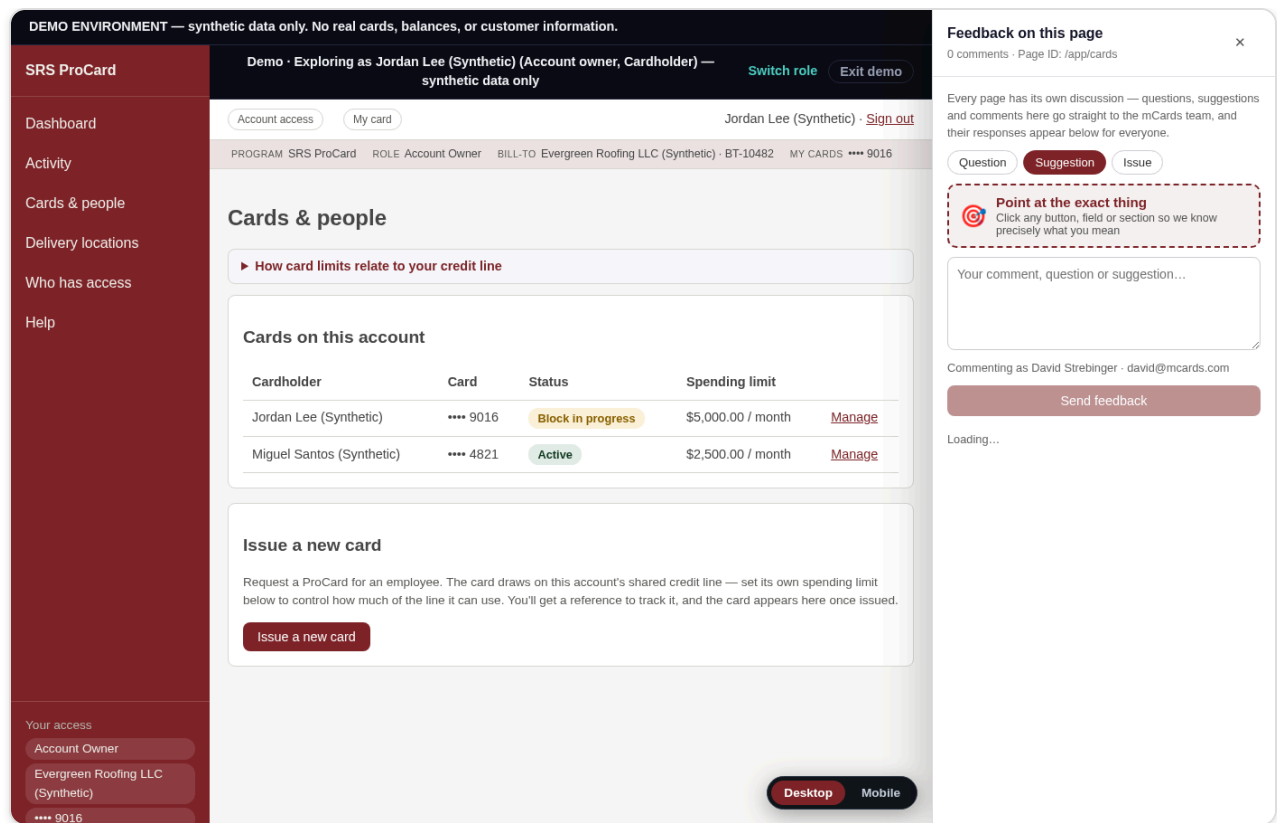
Tier	Adds
CUSTOMER_SERVICE	SEARCH_CUSTOMERS, VIEW_CUSTOMER_360, CREATE_CASE
CLIENT_CARE	MANAGE_CASE, SERVICE_CARDS, CONFIRM_MANUAL_ACTION, START_PRIVILEGED_SESSION
CREDIT	VIEW_LIMITS, CHANGE_LIMIT, APPROVE_SHIP_TO
OPERATIONS	ESCALATE, OPERATIONAL_REPORTS, SET_ENROLLMENT_STATUS, VIEW_FEED_HEALTH
PROGRAM_MANAGEMENT	PROGRAM_REPORTS
MCARDS_ADMIN	MANAGE_PROVISIONING, MANAGE_PROGRAM_CONFIG, VIEW_AUDIT — plus <code>allPrograms</code> scope

Multi-role users get the **union** of their roles' capabilities, computed server-side at session resolution (`capabilitiesForRoles`). Roles never subtract from each other.

3.3 Why capability-based instead of role checks in pages

- Pages and APIs ask "*can this session* `CHANGE_LIMIT` ?", never "*is this a Credit person?*". New roles or re-tiered permissions are a change to one mapping table, not a hunt through the codebase.
- The same capability set drives **navigation** (a nav item renders only if the session holds one of its required capabilities) and **API authorization** — so the menu a user sees is provably the same shape as what the API will let them do. Hiding a button is never the security mechanism; it is a courtesy.

4. The scope cascade: who sees which rows



Cards & people — the owner issues cards and sets spend rules against the shared credit line.

Capabilities say what **actions** you may take. Scope says which **data** those actions may touch. The two are independent and both enforced.

```
sign-in
|
▼
Identity resolution (session)
  email → platform_user → active access_grants
  |
  ▼
AppContext {roles, caps, programIds | allPrograms, businessIds, cardIds, viewAs, ...}
|
├─ mCards Admin ──────────> allPrograms = true   (platform-wide)
├─ Program staff ─────────> programIds = [their program(s)]   ← TENANCY WALL
├─ Account Owner / DA ───> businessIds = [their Bill-To(s)]
├─ Cardholder ───────────> cardIds = [their card(s)]
|
▼
Every query runs inside withScope(dbScopeFor(ctx), ...) which sets
Postgres session variables – and Row-Level Security policies on the
tables re-derive visibility INSIDE the database:
"you can read this card row only if its business_id / card_id
is in your scope, or your session has platform scope"
```

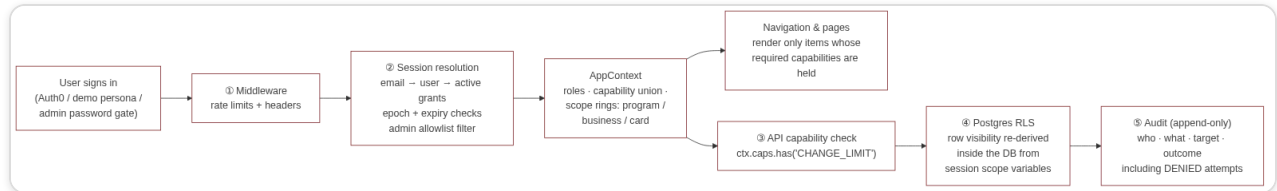
Key properties:

1. **Four nested rings:** platform → program → business → card. Every entity belongs to exactly one ring path, so "who can see this?" always has a single, computable answer.
2. **The program tenancy wall is hard.** A staff grant is scoped to a program. A URL, ID, or API call referencing another program's data returns 404 — the RLS policy means the row does not come back from the database at all. We chose 404 over 403 so the existence

of other tenants' data is not discoverable.

3. **Scope is enforced twice.** The UI filters what it shows (navigation, lists), and the database re-filters via RLS with the session's scope variables. A bug in a page or API handler cannot widen visibility beyond what RLS allows.
4. **Context switching is server-validated.** A user granted multiple businesses picks an active Bill-To; the selection is validated against their grants and can only *narrow*, never widen, scope.

5. Enforcement layers (defense in depth)



Enforcement pipeline: middleware → session → capability check → RLS → audit.

Each request passes through five independent layers. Any single layer failing does not expose data.

① Middleware	rate limits (stricter bucket for /api/auth, /api/dev-login, /api/admin-auth), security headers
② Session	signed cookie / Auth0 → identity; epoch check (bulk invalidation); absolute expiry; mCards Admin additionally gated by the password allowlist (david@ / alabarbera@ mcards.com) — non-allowlisted identities have MCARDS_ADMIN grants filtered OUT of their effective session
③ Capability check	page/API asks <code>ctx.caps.has("...")</code> before rendering or acting
④ Database RLS	row-level policies re-derive business/card/program scope inside Postgres, per query, from session variables
⑤ Audit	append-only event log: who, what, target, outcome — including DENIED attempts (e.g. the live 200-vs-403 authorization proof in the demo writes both outcomes)

Special cases worth knowing:

- **View-as (demo/testing):** an mCards Admin can view the product as another user. The impersonation is a signed, server-verified claim; the acting admin is recorded on every audited action (`viewAs.actorEmail`). It narrows to the target user's scope — it never mixes scopes.
- **Privileged support sessions:** staff acting inside a customer's account do it inside an explicit, time-boxed privileged session (`START_PRIVILEGED_SESSION`), visibly bannered in the UI and recorded in audit.
- **Manual/pending states are honest:** actions that require a human step (e.g. Ops confirming a freeze) stay visibly "pending" rather than pretending instant completion.

6. How the cascade shows up in each portal

DEMO ENVIRONMENT — synthetic data only. No real cards, balances, or customer information.

mCards Portal

Home

Clients

Work Queues

Credit

Operations

Client Feedback

Platform

Demo · Exploring as Ada Admin (Synthetic) (mCards admin) — synthetic data only

Switch role

Exit demo

View as...

Ada Admin (Synthetic) (MCARDS_ADMIN) · Sign out

PROGRAM All programs

ROLE mCards Admin

Platform

+ Create new program

mCards-level administration across all programs.

Setting up a new program

Users & access

Provision internal staff and program/client access grants.

Open user administration →

Brand Studio

Create brands, edit design tokens, and assign them to programs.

Open Brand Studio →

Program configuration

Feature flags, adapter modes, and integration realism.

Open configuration →

Client feedback

Questions, suggestions and issues logged from the demo experience — reply and track status.

Open feedback inbox →

Audit

Search the append-only audit log.

Feedback on this page

0 comments · Page ID: /portal/platform

Every page has its own discussion — questions, suggestions and comments here go straight to the mCards team, and their responses appear below for everyone.

Question

Suggestion

Issue

Point at the exact thing

Click any button, field or section so we know precisely what you mean

Your comment, question or suggestion...

Tell us who you are (once) so we can respond:

Your name

Your email

Send feedback

Loading...

mCards Admin platform hub — all-program administration behind the tenancy wall.

DEMO ENVIRONMENT — synthetic data only. No real cards, balances, or customer information.

SRS ProCard Portal

Home

Clients

Work Queues

Credit

Demo · Exploring as Carla Credit (Synthetic) (Credit) — synthetic data only

Switch role

Exit demo

Search clients, references...

Carla Credit (Synthetic) (CREDIT) · Sign out

PROGRAM SRS ProCard

ROLE CREDIT

Credit workspace

How credit servicing works here

Record a phone or email request

Client

Blue Peak Exteriors (Synthetic)

Received via

Phone

Request type

Credit limit change

Summary

Detail (what the customer asked for, verbatim where possible)

Record request

Open credit requests

No open credit requests. New phone/email intakes and escalations will appear here.

Feedback on this page

0 comments · Page ID: /portal/credit

Every page has its own discussion — questions, suggestions and comments here go straight to the mCards team, and their responses appear below for everyone.

Question

Suggestion

Issue

Point at the exact thing

Click any button, field or section so we know precisely what you mean

Your comment, question or suggestion...

Tell us who you are (once) so we can respond:

Your name

Your email

Send feedback

Loading...

Credit team workspace — program-scoped intakes, approvals and oversight.

User	Lands on	Navigation they get	Data ring
Miguel (Cardholder)	/app	Dashboard, Activity, Help	his card only
Priya (Delegated Admin)	/app	+ Who has access (read), card servicing actions	Evergreen, servicing only
Jordan (Account Owner)	/app	+ Cards & people, Delivery locations, access management	Evergreen, full account
Support Agent	/portal	Home, Clients, Work Queues	all SRS ProCard customers, read + case actions
Credit Team	/portal	+ Credit	+ limits and approvals
Operations	/portal	+ Operations	+ feeds, reports, enrollment
Program Admin	/portal	+ program reports	program-wide
mCards Admin	/portal	+ Client Feedback, Platform	every program; platform config, brands, audit, Devin pipeline

The navigation table above is *generated*, not hand-maintained: each nav item declares required capabilities and the layout filters by the session's capability set — one source of truth for "who sees what."

Branding follows the same cascade: Brand Studio (mCards-only) assigns a brand to a program; the customer surface and program-scoped staff portal render that program's tokens at runtime, while the mCards shell keeps the mCards identity. One deployment, many branded experiences, no tenant bleed.

7. Why we designed it this way

- Roles are products.** Each audience gets a surface shaped around their job, not a mega-app with hidden menus. This is why a cardholder's app has five items and the platform shell has a work-queue spine.
- Nested staff tiers mirror real escalation.** Support → Client Care → Credit → Operations → Program Management is how a servicing organisation actually escalates; nesting capabilities means escalation never requires re-authentication or duplicate tooling.
- Delegation is a first-class object, not a shared login.** The owner grants a *scoped role* to a named person. Every action is attributable; revocation is instant; the delegate's UI honestly shows the boundary.
- Tenancy is structural, not conventional.** The program wall lives in grants + RLS, so a future second program (another distributor, another brand) is isolated by construction — no per-feature "did we remember to filter?" risk.
- UI hints, server decides, database verifies.** Three independent expressions of the same policy mean a defect must occur in all three simultaneously to leak data.
- Everything sensitive is observable.** Append-only audit (including denials) makes the platform supportable and demonstrable — the demo's live 200-vs-403 proof is the same machinery a real operator would use.
- Embeddability.** Because authorization is resolved into a portable `AppContext` and enforced at the API/RLS layers rather than in page code, the same contract can back the partner API suite for SRS/Home Depot to host these experiences inside their own applications.

8. Glossary

- Grant** — a row linking a user to a role at a scope (platform / program / business / card). Revocable, auditable, time-stamped.
- Capability** — a fine-grained permission (e.g. `CHANGE_LIMIT`) derived from roles; the only thing pages and APIs check.
- AppContext** — the server-resolved session object: identity + roles + capability set + scope rings.
- RLS** — Postgres Row-Level Security; per-row visibility policies evaluated inside the database using session scope variables.
- Tenancy wall** — the program-level isolation boundary; cross-program access is a hard 404.
- View-as** — signed, audited admin impersonation for demo/testing; narrows scope to the target user.