LAD Architecture Training Guide This document trains developers on how to work within the LAD SaaS architecture without breaking platform rules. It is mandatory reading for all backend, frontend, and full-stack contributors.

==

1. LAD CORE PHILOSOPHY

Backend + SDK = Source of Truth Web (Next.js) = Presentation Layer Only

• Business logic never lives in the web • UI never drives architecture • Features are isolated • Verticals extend, they do not fork

If something feels hard to do cleanly, stop and refactor.

==

2. BACKEND TRAINING

Location: backend/features//

File Types and Responsibilities:

• routes/*.js

- Define API endpoints only
- Wire middleware and controllers
- NO business logic

• controllers/*.controller.js

- Handle request/response
- Call services
- NO database queries

• services/*.service.js

- Business logic
- Workflow orchestration
- Domain rules
- NO Express objects

• models/*.pg.js

- PostgreSQL queries only
- CRUD and transactions
- NO business logic

• auth.js

- JWT validation
- Tenant extraction

• manifest.js

- Feature metadata
- Plan access
- Feature flags

Rules: • No file > 400 lines • No cross-feature imports • APIs must be /api//*

Naming Standards: • Files: kebab-case (campaign-steps.controller.js) • Classes: PascalCase (CampaignService) • Functions: camelCase (createCampaign, updateCampaignStatus) • Variables: camelCase (campaignData, stepCount) • Constants: UPPER_SNAKE_CASE (MAX_RETRIES, DEFAULT_TIMEOUT) • Database tables: snake_case (campaign_steps, campaign_leads) • API routes: kebab-case (/api/campaigns/create-campaign)

===

2.1. DATABASE & MIGRATIONS

New Tables Creation Rules:

1. Never create tables directly in features • All tables go in migrations/ directory • Use numbered migration files (001_, 002_, etc.) • Include rollback SQL
2. Migration File Structure: migrations/XXX_feature_name_tables.sql

   -- Migration: Add campaign tables -- Date: 2025-12-23 -- Feature: campaigns

   CREATE TABLE IF NOT EXISTS campaigns (id SERIAL PRIMARY KEY, tenant_id INTEGER NOT NULL, name VARCHAR(255) NOT NULL, created_at TIMESTAMP DEFAULT NOW(), updated_at TIMESTAMP DEFAULT NOW());

   -- Add indexes CREATE INDEX idx_campaigns_tenant ON campaigns(tenant_id);

   -- Rollback: -- DROP TABLE IF EXISTS campaigns CASCADE;

3. Table Naming Rules:

   • Plural form: campaigns (not campaign)
   • Feature prefix for complex features: campaign_steps, campaign_leads
   • snake_case always
   • Include tenant_id for multi-tenancy
   • Include created_at, updated_at timestamps

4. Column Naming Rules:
   • snake_case: first_name, email_address
   • Foreign keys: ID (USER_ID, CAMPAIGN_ID)
   • BOOLEAN: IS, HAS_, can_* (is_active, has_access)
   • Dates: *_at (created_at, started_at, completed_at)
   - Indexes:
     • tenant_id always indexed
     • Foreign keys indexed
     • Frequently queried columns
     • Composite indexes for common queries
   - Migration Workflow:
     • Create migration file in migrations/

- Test migration locally
- Document in README
- Apply via migration script
- Never modify existing migrations

==

3. FRONTEND SDK TRAINING

Location: frontend/sdk/features//

Purpose: Expose backend features to frontend apps safely and consistently.

File Types:

- api.ts
  - HTTP calls
  - Feature-prefixed endpoints
  - Typed responses
- hooks/*.ts
  - React hooks only
  - useQuery/useMutation
  - NO JSX
- types.ts
  - DTOs
  - API response types
  - Domain types
- index.ts
  - Public exports only

Rules:

- NO Next.js imports
- NO JSX
- NO CSS
- SDK must run tests without web

SDK Naming Standards:

- API functions: camelCase (getCampaigns, createCampaign)
- Hooks: use prefix (useCampaigns, useCampaign, useCampaignMutation)

• Types: PascalCase (Campaign, CampaignStep, CreateCampaignRequest)

• Interfaces: PascalCase (ICampaignService, ICampaignAPI)

• Type guards: is prefix (isCampaign, isValidStep)

==

4. FRONTEND WEB TRAINING

Location: frontend/web/

Purpose: Render UI and handle routing.

File Types:

• page.tsx

- Page composition
- SDK hook consumption

• components/*.tsx

- UI components
- JSX allowed

• layout.tsx

- App layouts

Rules: • Web never calls backend directly • Web never contains business logic • Web only consumes SDK

==

6. ROLE-BASED WORKFLOWS

Backend / Full-Stack Developer: • Work on backend + SDK • Use web locally for visualization • Commit ONLY backend + SDK

Frontend Developer: • Work on frontend/web • Use deployed backend • SDK changes only if contract changes

==

7. MERGE CONTRACT

Only the following paths are allowed to be merged into LAD:

✓ backend/features//**

✓ frontend/sdk/features//**

The following paths must NEVER be merged:

✗ frontend/web/**

✗ lad-sandbox/**

✗ infra/**

✗ cloudbuild*

Rationale:

• Web code is UI-only and managed by frontend team

• Sandbox is local testing only (never committed)

• Infrastructure is managed separately

• Cloud Build configs are deployment-specific

==

8. FEATURE DEVELOPMENT CHECKLIST

- Pre-Development:
  - Feature repository created
  - Backend structure initialized
  - SDK structure initialized
  - Documentation reviewed
- Development - Backend:
  - Controllers created
  - Models defined
  - Routes implemented (//*)
  - Services written
  - manifest.js present
  - All files < 400 lines
  - No cross-feature imports
- Development - SDK:
  - api.ts implemented
  - hooks/ directory with domain-split hooks
  - types.ts with all interfaces
  - index.ts barrel exports
  - No Next.js imports
  - No JSX/TSX files
  - Framework-independent
- Testing:

        - Backend tests written and passing
        - SDK tests written and passing
        - Test coverage meaningful
        - Edge cases covered
    - Pre-Merge Validation:
        - Run: bash scripts/path-guard.sh
        - No file exceeds 400 lines
        - Feature isolation maintained
        - API routes feature-prefixed
        - SDK framework-independent
        - No hardcoded secrets

See: feature_checklist.md for complete checklist

==

9. COMMON MISTAKES (DO NOT DO)

• JSX in SDK • SQL in controllers • Business logic in routes • UI logic in backend • Committing sandbox or test UI • Files > 400 lines • Cross-feature imports • Non-prefixed API routes • Next.js in SDK • Hardcoded secrets • Web code in backend/SDK PRs

==

10. AUTOMATED VALIDATION

Path Guard (runs on PRs): • bash scripts/path-guard.sh • Validates all architecture rules • Checks file sizes, isolation

==

10.1. PULL REQUEST PROCESS

PR Template Sections:

- Feature Identification: Feature name (e.g., campaigns, deals-pipeline)

•ERROR HANDLING & LOGGING

Backend Error Handling:

- Controller Level: try { const result = await campaignService.createCampaign(data); res.json({ success: true, data: result }); } catch (error) { console.error('Campaign creation failed:', error); res.status(500).json({ success: false, error: 'Failed to create campaign' }); }
- Service Level: • Throw meaningful errors • Use Error classes (ValidationError,

NotFoundError) • Include context in error messages

- Model Level: • Log SQL errors • Wrap database errors • Return null for not found

Logging Standards: • console.log() - Info/debug • console.warn() - Warnings • console.error() - Errors with context • Include feature name in logs: [campaigns] Creating campaign...

SDK Error Handling:

- API Calls: try { const response = await apiClient.post('/campaigns', data); return response.data; } catch (error) { console.error('[campaigns] API call failed:', error); throw new Error('Failed to create campaign'); }
- Hooks:
  • Use React Query error handling
  • Provide error states
  • Clear error messages

==

12. SECURITY BEST PRACTICES

Authentication & Authorization:

• Always validate JWT tokens

• Check tenant_id in all queries

• Use middleware for auth checks

• Never trust client input

Data Validation:

• Validate all inputs at controller level

18. REFERENCE DOCUMENTS

Essential Reading: • feature_checklist.md - Complete development checklist • FEATURE_REPOSITORY_RULES.md - Feature repo governance • lad-feature-developer-playbook.md - Developer guide • FEATURE_REPOSITORIES_INDEX.md - All features index • PULL_REQUEST_TEMPLATE.md - PR requirements

Quick Commands: • Path guard: bash scripts/path-guard.sh • Backend tests: cd backend && npm test • SDK tests: cd frontend/sdk && npm test • Check file sizes: find . -name "*.js" -exec wc -l {} \; | awk '$1 > 400'

Support: • Architecture questions: Check documentation first • Stuck on isolation: Ask before coding • Unsure about approach: Refactor, don't patch

==

13. PERFORMANCE BEST PRACTICES

Database: • Index frequently queried columns • Use LIMIT for large result sets • Avoid N+1 queries • Use database pooling • Optimize joins

Backend: • Cache frequently accessed data • Async operations where possible • Batch database operations • Stream large responses • Use pagination

SDK: • React Query caching • Debounce user inputs • Lazy load data • Prefetch when predictable • Minimize bundle size

==

14. TESTING GUIDELINES

Backend Testing: • Unit tests for services • Integration tests for routes • Mock database for unit tests • Test error cases • Test edge cases

File: backend/features/campaigns/tests/campaign.service.test.js

SDK Testing: • Test API functions • Test hooks with React Testing Library • Mock API responses • Test loading/error states • Test data transformations

File: frontend/sdk/features/campaigns/tests/api.test.ts File:

frontend/sdk/features/campaigns/tests/hooks.test.ts

Test Coverage: • Aim for 70%+ coverage • Critical paths 100% covered • Edge cases tested • Error scenarios tested

Running Tests: • Backend: cd backend && npm test • SDK: cd frontend/sdk && npm test • Specific feature: npm test -- campaigns • Watch mode: npm test -- --watch

==

15. CODE REVIEW CHECKLIST

Before Submitting PR: □ Run path guard: bash scripts/path-guard.sh □ All tests passing □ No console.log() left in code (use proper logging) □ No commented code □ No TODO comments without tickets □ Types properly defined □ Error handling present □ Documentation updated Code Quality: □ Functions are small and focused □ No duplicate code □ Clear variable names □ No magic numbers (use constants) □ Proper error messages □ Consistent formatting Architecture: □ Feature isolation maintained □ No cross-feature dependencies □ Proper layering (routes → controllers → services → models) □ SDK framework-independent □ No business logic in

UI

==

16. DEPLOYMENT CHECKLIST

Pre-Deployment: □ All tests passing in CI/CD □ Database migrations prepared □ Environment variables documented □ Feature flags configured □ Rollback plan documented

Post-Deployment: □ Monitor logs for errors □ Verify feature functionality □ Check performance metrics □ Test with real data □ Document any issues

==

17. PR type (Backend / SDK / Frontend) • Paths modified
18. Architecture Compliance Checklist • File size limits (< 400 lines) • Feature structure (manifest.js, index.ts) • Domain-split hooks • Feature-prefixed APIs • No cross-feature imports • SDK framework independence
19. Testing Requirements • SDK tests passing • Backend tests passing • Test coverage evidence • Screenshot or console output
20. Forbidden Items Check • No lad-sandbox/ committed • No frontend/web/ (unless frontend dev) • No infra/ or cloudbuild changes • No secret files • No hardcoded credentials
21. Migration Steps • Database migrations documented • Feature flags noted • Deployment requirements

Before Creating PR: • Run path guard locally: bash scripts/path-guard.sh • Run all tests: cd backend && npm test • Run SDK tests: cd frontend/sdk && npm test • Check file sizes • Review checklist completely

PR Review Process:

16. Create PR with template filled
17. Automated checks run (path guard, tests)
18. Architecture review by maintainers
19. Address feedback
20. Merge only backend/features and sdk/features, structure • Must pass before merge

GitHub Actions: • PR template with compliance checklist • Automated testing (backend + SDK) • Path guard validation • Required checks before merge

==

11. FINAL RULE

If a change violates isolation: STOP. Refactor instead of patching.

If unsure, ask before coding.

==

12. REFERENCE DOCUMENTS

• feature_checklist.md - Complete development checklist • FEATURE_REPOSITORY_RULES.md - Feature repo governance • lad-feature-developer-playbook.md - Developer guide • FEATURE_REPOSITORIES_INDEX.md - All features index • SANDBOX_SETUP_SUMMARY.md - Sandbox documentation

==

GLOSSARY

Terms:

• Feature: Isolated business capability (campaigns, deals-pipeline)

• SDK: Software Development Kit - framework-independent feature API

• Sandbox: Local testing environment with symlinks

• Manifest: Feature metadata and configuration (manifest.js)

• Path Guard: Automated architecture validation script

• Feature Isolation: No dependencies between features

• MVC: Model-View-Controller pattern

• Multi-tenancy: Data isolation by tenant_id

File Conventions:

• .controller.js - Request/response handling

• .service.js - Business logic

• .pg.js - PostgreSQL queries

• .test.js - Test files

• manifest.js - Feature registration

• index.ts - Public exports

Common Acronyms:

- LAD: Lead Agent Deal
- API: Application Programming Interface
- SDK: Software Development Kit
- JWT: JSON Web Token
- CRUD: Create, Read, Update, Delete
- DTO: Data Transfer Object
- PR: Pull Request
- CI/CD: Continuous Integration/Continuous Deployment

==