**PM0: Phased Deployment & Refactoring Strategy**

**EXECUTIVE SUMMARY**

**Current State:** PM0 (Phase Minus Zero) is a strategic planning protocol built with Vercel + Supabase + React (Vite + TypeScript). The application has critical bugs preventing deployment and needs systematic refactoring to ensure stable, iterative releases.

**Goal:** Transform the codebase into a stable, deployable application through systematic phases that each deliver a working, testable increment.

**Approach:** Incremental delivery with strict isolation between phases. Each phase results in a deployable state, even if feature-incomplete.

**PHASE 0: FOUNDATION STABILIZATION**

**Duration:** 2-3 hours | **Risk:** Critical | **Deploy:** Yes

**Objective**

Establish a minimal viable deployment that builds and runs successfully in all environments (local dev, demo mode, and production).

**Deliverables**

**0.1: Emergency Type System Fix**

**Branch:** phase-0/type-system-foundation

**Changes:**

1. Generate Supabase types locally and commit to repository
2. SUPABASE\_PROJECT\_REF=<your-ref> npm run typegen
3. git add src/types/database.ts
4. Create fallback type definitions if generation fails
5. // src/types/database.fallback.ts
6. export type Database = {
7. public: {
8. Tables: Record<string, any>;
9. Views: Record<string, any>;
10. Functions: Record<string, any>;
11. };
12. };
13. Update typegen.sh to use fallback if generation unavailable
14. if [[ -f "$OUTPUT\_FILE" ]]; then
15. echo "✓ Types exist. Skipping generation."
16. exit 0
17. fi
18. if [[ -z "$PROJECT\_REF" ]] || ! command -v supabase >/dev/null 2>&1; then
19. echo "⚠ Using fallback types"
20. cp "${ROOT\_DIR}/src/types/database.fallback.ts" "$OUTPUT\_FILE"
21. exit 0
22. fi

**Success Criteria:**

* ✅ npm run build succeeds without SUPABASE\_PROJECT\_REF
* ✅ TypeScript compilation completes without errors
* ✅ Build works in Vercel CI/CD environment

**Git Workflow:**

git checkout -b phase-0/type-system-foundation

# Make changes

git add .

git commit -m "phase-0: establish stable type system foundation"

git push origin phase-0/type-system-foundation

gh pr create --title "Phase 0.1: Type System Foundation" --base main

gh pr merge --squash --delete-branch --auto

**0.2: Configuration Layer Rebuild**

**Branch:** phase-0/config-rebuild

**Changes:**

1. Rewrite src/lib/env.ts with graceful degradation
   * Use safeParse instead of throwing during module import
   * Export validation status with config
   * Provide clear fallback behavior
2. Rewrite src/lib/supabaseClient.ts with demo mode support
   * Respect env.useDemoData flag
   * Return null in demo mode
   * Throw clear errors when misconfigured
   * Return type: SupabaseClientType | null
3. Create configuration integration layer
4. // src/lib/config.ts
5. import { env } from './env';
6. import { getSupabaseClient } from './supabaseClient';
7. export const config = {
8. isValid: env.isValid,
9. isDemoMode: env.useDemoData,
10. supabase: getSupabaseClient(),
12. requireDatabase(): NonNullable<typeof config.supabase> {
13. if (!config.supabase) {
14. throw new Error('Database required but running in demo mode');
15. }
16. return config.supabase;
17. },
18. };

**Success Criteria:**

* ✅ Demo mode works: VITE\_USE\_DEMO\_DATA=true npm run dev
* ✅ Production mode fails gracefully without credentials
* ✅ No white screens; all errors show helpful UI
* ✅ App respects configuration in all code paths

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-0/config-rebuild

# Make changes

git add .

git commit -m "phase-0: rebuild configuration layer with demo mode support"

git push origin phase-0/config-rebuild

gh pr create --title "Phase 0.2: Configuration Rebuild" --base main

gh pr merge --squash --delete-branch --auto

**0.3: Error Boundary & Dev Experience**

**Branch:** phase-0/error-boundaries

**Changes:**

1. Add top-level error boundary in src/App.tsx
2. if (!env.isValid) {
3. return <ConfigurationErrorScreen />;
4. }
5. Create helpful error screens
   * Configuration missing screen
   * Database connection error screen
   * Generic error fallback with stack traces (dev only)
6. Add development utilities
7. // src/lib/dev.ts
8. export const dev = {
9. isDev: import.meta.env.DEV,
10. logLevel: import.meta.env.DEV ? 'debug' : 'error',
11. showStackTraces: import.meta.env.DEV,
12. };

**Success Criteria:**

* ✅ Invalid config shows friendly error page
* ✅ Runtime errors caught and displayed appropriately
* ✅ Development mode has verbose logging
* ✅ Production mode has minimal, safe logging

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-0/error-boundaries

# Make changes

git add .

git commit -m "phase-0: add error boundaries and dev experience improvements"

git push origin phase-0/error-boundaries

gh pr create --title "Phase 0.3: Error Boundaries" --base main

gh pr merge --squash --delete-branch --auto

**Phase 0 Exit Criteria**

Before proceeding to Phase 1:

* [ ] All Phase 0 PRs merged
* [ ] Application builds successfully on Vercel
* [ ] Application runs in all three modes:
  + [ ] Local dev with demo data
  + [ ] Local dev with real Supabase
  + [ ] Production deployment
* [ ] No console errors on initial load
* [ ] Error screens tested and working
* [ ] Main branch is clean (all feature branches deleted)

**Deploy to:** Staging environment for Phase 0 verification

**PHASE 1: CLIENT USAGE AUDIT & STANDARDIZATION**

**Duration:** 3-4 hours | **Risk:** Medium | **Deploy:** Yes

**Objective**

Ensure all code that uses Supabase client handles demo mode correctly and follows consistent patterns.

**Deliverables**

**1.1: Client Usage Audit**

**Branch:** phase-1/audit-client-usage

**Process:**

1. Search for all getSupabaseClient() calls
2. grep -r "getSupabaseClient" src/ --include="\*.ts" --include="\*.tsx"
3. Document each usage in docs/client-audit.md:
   * File path
   * Context (what data operation)
   * Current null-handling (if any)
   * Required changes
4. Categorize by priority:
   * **Critical:** Authentication, core data fetching
   * **Important:** User actions, mutations
   * **Nice-to-have:** Analytics, logging

**Deliverable:** Comprehensive audit document

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-1/audit-client-usage

# Create audit document

git add docs/client-audit.md

git commit -m "phase-1: complete Supabase client usage audit"

git push origin phase-1/audit-client-usage

gh pr create --title "Phase 1.1: Client Usage Audit" --base main

gh pr merge --squash --delete-branch --auto

**1.2: Create Standardized Hooks**

**Branch:** phase-1/standardized-hooks

**Changes:**

1. Create src/hooks/useSupabase.ts
2. export function useSupabase() {
3. const client = getSupabaseClient();
5. return {
6. client,
7. isDemo: client === null,
8. isConnected: client !== null,
9. requireClient: () => {
10. if (!client) {
11. throw new Error('Database operation not available in demo mode');
12. }
13. return client;
14. },
15. };
16. }
17. Create src/hooks/useQuery.ts (wraps React Query with demo mode)
18. export function useSupabaseQuery<T>(
19. key: string[],
20. queryFn: (client: SupabaseClient) => Promise<T>,
21. options?: { demoData?: T }
22. ) {
23. const { client, isDemo } = useSupabase();
25. return useQuery({
26. queryKey: key,
27. queryFn: async () => {
28. if (isDemo) {
29. if (options?.demoData) {
30. return options.demoData;
31. }
32. throw new Error('Demo data not provided for this query');
33. }
34. return queryFn(client!);
35. },
36. ...options,
37. });
38. }
39. Create src/hooks/useMutation.ts
40. export function useSupabaseMutation<T, V>(
41. mutationFn: (client: SupabaseClient, vars: V) => Promise<T>,
42. options?: { demoMode?: (vars: V) => T }
43. ) {
44. const { client, isDemo } = useSupabase();
46. return useMutation({
47. mutationFn: async (vars: V) => {
48. if (isDemo) {
49. if (options?.demoMode) {
50. return options.demoMode(vars);
51. }
52. throw new Error('Mutation not available in demo mode');
53. }
54. return mutationFn(client!, vars);
55. },
56. ...options,
57. });
58. }

**Success Criteria:**

* ✅ Hooks created and tested
* ✅ Demo mode behavior clearly defined
* ✅ TypeScript types are strict and helpful
* ✅ Error messages are clear

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-1/standardized-hooks

# Create hooks

git add src/hooks/

git commit -m "phase-1: create standardized Supabase hooks with demo mode support"

git push origin phase-1/standardized-hooks

gh pr create --title "Phase 1.2: Standardized Hooks" --base main

gh pr merge --squash --delete-branch --auto

**1.3: Refactor Critical Data Operations**

**Branch:** phase-1/refactor-critical-ops

**Changes:**

1. Update authentication flows to use new hooks
2. Update core data fetching (projects, scenarios, etc.)
3. Add demo data for critical operations
4. Test both demo and production modes

**Priority Order:**

1. Authentication (auth flows)
2. Project/workspace loading
3. Scenario management
4. User preferences

**Success Criteria:**

* ✅ All critical paths work in demo mode
* ✅ All critical paths work in production mode
* ✅ No null reference errors
* ✅ Clear feedback when operations unavailable

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-1/refactor-critical-ops

# Refactor critical operations

git add .

git commit -m "phase-1: refactor critical data operations to use standardized hooks"

git push origin phase-1/refactor-critical-ops

gh pr create --title "Phase 1.3: Refactor Critical Operations" --base main

gh pr merge --squash --delete-branch --auto

**1.4: Refactor Secondary Operations**

**Branch:** phase-1/refactor-secondary-ops

**Changes:**

1. Update all remaining Supabase client usages
2. Add demo data where appropriate
3. Disable features gracefully if not available in demo
4. Add UI indicators for demo mode

**Success Criteria:**

* ✅ Zero direct getSupabaseClient() calls outside of hooks
* ✅ All features either work or degrade gracefully in demo mode
* ✅ Demo mode clearly indicated in UI

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-1/refactor-secondary-ops

# Refactor remaining operations

git add .

git commit -m "phase-1: refactor secondary operations and complete client standardization"

git push origin phase-1/refactor-secondary-ops

gh pr create --title "Phase 1.4: Refactor Secondary Operations" --base main

gh pr merge --squash --delete-branch --auto

**Phase 1 Exit Criteria**

Before proceeding to Phase 2:

* [ ] All Phase 1 PRs merged
* [ ] Audit document complete
* [ ] All standardized hooks created and tested
* [ ] No direct Supabase client usage outside hooks
* [ ] Demo mode fully functional for all features
* [ ] Production mode tested and working
* [ ] No TypeScript errors
* [ ] No runtime null reference errors

**Deploy to:** Staging + Production (incremental rollout)

**PHASE 2: CODEBASE ORGANIZATION & CLEANUP**

**Duration:** 4-6 hours | **Risk:** Low | **Deploy:** Yes

**Objective**

Organize the codebase logically, remove technical debt, and establish maintainable patterns.

**Deliverables**

**2.1: File Structure Reorganization**

**Branch:** phase-2/reorganize-structure

**Changes:**

1. Establish clear directory structure:
2. src/
3. ├── components/
4. │ ├── common/ # Reusable UI components
5. │ ├── features/ # Feature-specific components
6. │ └── layouts/ # Layout components
7. ├── hooks/ # Custom React hooks
8. ├── lib/ # Core utilities
9. │ ├── config/ # Configuration (env, supabase)
10. │ ├── api/ # API utilities
11. │ └── utils/ # Helper functions
12. ├── pages/ # Page components
13. ├── stores/ # Zustand stores
14. ├── types/ # TypeScript types
15. └── demo/ # Demo data and utilities
16. Move files to appropriate locations
17. Update all imports
18. Create index files for cleaner imports

**Success Criteria:**

* ✅ Logical directory structure
* ✅ All imports updated
* ✅ No broken imports
* ✅ Build succeeds

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-2/reorganize-structure

# Reorganize files

git add .

git commit -m "phase-2: reorganize file structure for better maintainability"

git push origin phase-2/reorganize-structure

gh pr create --title "Phase 2.1: File Structure Reorganization" --base main

gh pr merge --squash --delete-branch --auto

**2.2: Remove Dead Code**

**Branch:** phase-2/remove-dead-code

**Process:**

1. Identify unused files
2. npx ts-prune
3. Identify unused exports
4. Remove commented-out code
5. Remove TODO comments without corresponding issues
6. Remove deprecated functions

**Success Criteria:**

* ✅ No unused files
* ✅ No commented-out code blocks
* ✅ All TODOs tracked in issues
* ✅ Build succeeds and tests pass

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-2/remove-dead-code

# Remove dead code

git add .

git commit -m "phase-2: remove dead code and deprecated functions"

git push origin phase-2/remove-dead-code

gh pr create --title "Phase 2.2: Remove Dead Code" --base main

gh pr merge --squash --delete-branch --auto

**2.3: Consolidate Duplicate Logic**

**Branch:** phase-2/consolidate-logic

**Focus Areas:**

1. Data fetching patterns
2. Form handling
3. Error handling
4. Loading states
5. Validation logic

**Changes:**

1. Create shared utilities for common patterns
2. Extract duplicate components
3. Standardize naming conventions
4. Document patterns in docs/patterns.md

**Success Criteria:**

* ✅ Duplicate logic consolidated into utilities
* ✅ Consistent patterns across codebase
* ✅ Documentation updated
* ✅ No regression in functionality

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-2/consolidate-logic

# Consolidate logic

git add .

git commit -m "phase-2: consolidate duplicate logic and establish patterns"

git push origin phase-2/consolidate-logic

gh pr create --title "Phase 2.3: Consolidate Duplicate Logic" --base main

gh pr merge --squash --delete-branch --auto

**2.4: Documentation & Code Comments**

**Branch:** phase-2/documentation

**Changes:**

1. Add JSDoc comments to all exported functions
2. Create docs/ directory with:
   * architecture.md - System architecture
   * patterns.md - Code patterns and conventions
   * deployment.md - Deployment guide
   * development.md - Local development guide
   * demo-mode.md - Demo mode documentation
3. Update README.md with current state
4. Add inline comments for complex logic

**Success Criteria:**

* ✅ All public APIs documented
* ✅ Architecture documented
* ✅ README accurate and helpful
* ✅ Complex logic explained

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-2/documentation

# Add documentation

git add .

git commit -m "phase-2: add comprehensive documentation"

git push origin phase-2/documentation

gh pr create --title "Phase 2.4: Documentation" --base main

gh pr merge --squash --delete-branch --auto

**Phase 2 Exit Criteria**

Before proceeding to Phase 3:

* [ ] All Phase 2 PRs merged
* [ ] File structure logical and organized
* [ ] No dead code or unused files
* [ ] Duplicate logic consolidated
* [ ] Documentation complete and accurate
* [ ] Code review shows high maintainability score
* [ ] No new bugs introduced

**Deploy to:** Production (full rollout)

**PHASE 3: TESTING & QUALITY ASSURANCE**

**Duration:** 4-6 hours | **Risk:** Low | **Deploy:** Yes

**Objective**

Establish comprehensive testing to prevent regressions and document expected behavior.

**Deliverables**

**3.1: Unit Tests for Core Utilities**

**Branch:** phase-3/unit-tests

**Coverage Targets:**

1. src/lib/config/ - 100%
2. src/lib/utils/ - 90%
3. src/hooks/ - 90%
4. Demo data utilities - 100%

**Tools:**

* Vitest for unit tests
* Testing Library for React components

**Success Criteria:**

* ✅ Core utilities covered by unit tests
* ✅ All tests passing
* ✅ Coverage meets targets

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-3/unit-tests

# Add unit tests

git add .

git commit -m "phase-3: add unit tests for core utilities"

git push origin phase-3/unit-tests

gh pr create --title "Phase 3.1: Unit Tests" --base main

gh pr merge --squash --delete-branch --auto

**3.2: Integration Tests**

**Branch:** phase-3/integration-tests

**Test Scenarios:**

1. Demo mode workflow
2. Production mode workflow
3. Error handling paths
4. Authentication flows

**Tools:**

* Playwright for E2E tests
* MSW for API mocking

**Success Criteria:**

* ✅ Critical user paths tested
* ✅ Demo and production modes tested
* ✅ All tests passing

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-3/integration-tests

# Add integration tests

git add .

git commit -m "phase-3: add integration tests for critical paths"

git push origin phase-3/integration-tests

gh pr create --title "Phase 3.2: Integration Tests" --base main

gh pr merge --squash --delete-branch --auto

**3.3: CI/CD Pipeline Enhancement**

**Branch:** phase-3/ci-cd-enhancement

**Changes:**

1. Add test runs to CI pipeline
2. Add type checking to CI
3. Add linting to CI
4. Add build verification
5. Add deployment smoke tests

**.github/workflows/ci.yml:**

name: CI

on: [push, pull\_request]

jobs:

test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- uses: actions/setup-node@v3

with:

node-version: '18'

- run: npm ci

- run: npm run typecheck

- run: npm run lint

- run: npm run test

- run: npm run build

**Success Criteria:**

* ✅ CI pipeline runs all checks
* ✅ Failed PRs blocked from merge
* ✅ Green checks required for deployment

**Git Workflow:**

git checkout main && git pull

git checkout -b phase-3/ci-cd-enhancement

# Enhance CI/CD

git add .

git commit -m "phase-3: enhance CI/CD pipeline with comprehensive checks"

git push origin phase-3/ci-cd-enhancement

gh pr create --title "Phase 3.3: CI/CD Enhancement" --base main

gh pr merge --squash --delete-branch --auto

**Phase 3 Exit Criteria**

Before marking project complete:

* [ ] All Phase 3 PRs merged
* [ ] Unit tests passing with good coverage
* [ ] Integration tests covering critical paths
* [ ] CI/CD pipeline running all checks
* [ ] All deployments tested and verified
* [ ] Documentation updated with testing info

**Deploy to:** Production (monitored rollout)

**SUCCESS METRICS**

**Technical Health**

* ✅ Zero build failures
* ✅ Zero TypeScript errors
* ✅ Zero runtime null reference errors
* ✅ <100ms initial load time
* ✅ Test coverage >80%

**Code Quality**

* ✅ No code duplication >10 lines
* ✅ All functions <50 lines
* ✅ All files <300 lines
* ✅ Cyclomatic complexity <10
* ✅ All public APIs documented

**Deployment**

* ✅ Successful builds in all environments
* ✅ Demo mode functional
* ✅ Production mode functional
* ✅ No deployment rollbacks
* ✅ <5 minute deployment time

**RISK MANAGEMENT**

**High-Risk Activities**

1. **Type system changes** - Could break builds
   * Mitigation: Commit types to repo first
2. **Config refactoring** - Could break all initialization
   * Mitigation: Add fallback behavior
3. **Large file moves** - Could break imports
   * Mitigation: Use automated refactoring tools

**Rollback Strategy**

Each phase is independently deployable. If issues arise:

1. Revert the problematic PR
2. Fix issues in new branch
3. Re-test thoroughly
4. Re-deploy

**AUTONOMOUS EXECUTION PROTOCOL**

**For Each Phase:**

1. Create feature branch from latest main
2. Complete all changes for that phase
3. Run local verification:
4. npm run typechecknpm run lintnpm run testnpm run buildnpm run dev # Manual smoke test
5. Commit with descriptive message
6. Push and create PR
7. Auto-merge if checks pass
8. Delete branch
9. Verify deployment
10. Proceed to next phase

**Decision Authority:**

* ✅ Auto-merge all PRs in defined phases
* ✅ Resolve merge conflicts within scope
* ✅ Create additional sub-phases if needed
* ❌ Skip phases or phase steps
* ❌ Merge failing PRs
* ❌ Deploy without verification

**COMPLETION CHECKLIST**

**Phase 0 Complete**

* [ ] Types system stable
* [ ] Configuration rebuilt
* [ ] Error boundaries added
* [ ] Deployed to staging

**Phase 1 Complete**

* [ ] Client usage audited
* [ ] Standardized hooks created
* [ ] All operations refactored
* [ ] Demo mode fully functional

**Phase 2 Complete**

* [ ] File structure organized
* [ ] Dead code removed
* [ ] Logic consolidated
* [ ] Documentation complete

**Phase 3 Complete**

* [ ] Unit tests added
* [ ] Integration tests added
* [ ] CI/CD enhanced
* [ ] Production deployment verified

**Final Verification**

* [ ] All deployments successful
* [ ] All tests passing
* [ ] All documentation current
* [ ] Zero known critical bugs
* [ ] Main branch clean
* [ ] Team notified of completion

**EXECUTION START:** Begin with Phase 0.1 immediately upon approval.