DOORSLAM ENGINEERING OPERATIONS PLAYBOOK

Development, Testing, Deployment & Documentation

Stack: React · Tailwind CSS · Supabase · PostgreSQL
Infrastructure: Vercel · GitHub · Google Workspace

CONFIDENTIAL
February 2026

Table of Contents

1. Architecture Overview

This playbook establishes the technical processes and standards for your engineering team. It covers the complete lifecycle from local development through to production deployment, and sets the foundation for scaling your team with remote/contract developers.

1.1 Technology Stack

Layer	Technology	Purpose
Frontend	React + Tailwind CSS	UI components and styling
Backend/BaaS	Supabase	Auth, APIs, real-time, storage
Database	PostgreSQL (via Supabase)	Primary data store
Hosting	Vercel	Frontend hosting + serverless functions
Version Control	GitHub	Source code management
Design	Figma + Figma Make	UI/UX design and handoff
Communication	Google Workspace	Chat, Meet, Drive, Gmail

1.2 Environment Architecture

You will operate three distinct environments, each with its own Vercel project configuration, Supabase project, and purpose. This separation ensures that development work never risks production data or user experience.

Environment	URL Pattern	Supabase Project	Purpose
Development	dev.yourdomain.com	yourapp-dev	Active development, feature work
Staging	staging.yourdomain.com	yourapp-staging	QA, UAT, pre-release testing
Production	yourdomain.com / app.yourdomain.com	yourapp-prod	Live users, real data

Critical Principle

Code flows up (dev → staging → production). Data never flows down. Never copy production user data into lower environments. Use seed data and factories instead.

2. GitHub Repository & Branching Strategy

2.1 Repository Structure

Use a single repository (monorepo or standard) with branch-based access control rather than forks. Forks are appropriate for open-source contributions but add unnecessary complexity for a private team. Contract developers should be added as collaborators with appropriate permissions.

Setting	Recommendation
Repo visibility	Private
Contractor access	Add as collaborators with "Write" role (not Admin)
Branch protection	Enable on main, staging, and develop branches
Forks	Not recommended for private teams — use branches instead

2.2 Branching Model (Git Flow — Simplified)

Adopt a simplified Git Flow model with three long-lived branches that map directly to your Vercel environments:

Branch	Deploys To	Merges From	Protected?
`main`	Production	staging (via release PR)	Yes — require PR + 1 approval
`staging`	Staging	develop (via PR)	Yes — require PR + 1 approval
`develop`	Development	Feature branches (via PR)	Yes — require PR, no self-merge

2.3 Feature Branch Naming Convention

All work happens on short-lived feature branches created from the develop branch. Use consistent naming:

```
feature/TICKET-123-user-login       ← New features
bugfix/TICKET-456-fix-nav-crash      ← Bug fixes
hotfix/TICKET-789-critical-auth      ← Production hotfixes (branch from main)
chore/update-dependencies            ← Maintenance work
refactor/TICKET-101-auth-module      ← Code refactoring
```

2.4 The Development Workflow (Step by Step)

Every developer follows this workflow for all changes:

1. **Pull latest:** git checkout develop && git pull origin develop
2. **Create branch:** git checkout -b feature/TICKET-123-description
3. **Develop locally:** Write code, commit frequently with clear messages
4. **Push branch:** git push origin feature/TICKET-123-description
5. **Open PR to develop:** Use the PR template, link to ticket, add screenshots
6. **Code review:** At least one team member reviews and approves

7. **Merge:** Squash merge into develop. Delete the feature branch.
8. **Verify on dev:** Confirm the deploy to dev.yourdomain.com works correctly

2.5 Commit Message Standard

Use Conventional Commits format for clear, parseable history:

```
feat(auth): add Google OAuth login flow
fix(dashboard): resolve chart rendering on mobile
docs(api): update endpoint documentation for v2
chore(deps): upgrade React to v19
refactor(utils): simplify date formatting helpers
```

PR Template

Create a .github/PULL_REQUEST_TEMPLATE.md file in your repo. Include sections for: Description, Ticket Link, Type of Change, Screenshots/Video, Testing Steps, and Checklist (tests pass, env vars documented, no console.logs).

2.6 Branch Protection Rules

Configure these in GitHub under Settings → Branches → Branch protection rules for each long-lived branch:

- **Require pull request reviews:** At least 1 approval required before merging
- **Require status checks to pass:** CI/CD pipeline (linting, tests, build) must succeed
- **Require branches to be up to date:** Ensure the branch is current with the target
- **Restrict who can push:** Only merge via PR, no direct pushes
- **Require conversation resolution:** All review comments must be resolved

3. Vercel Environment Setup

Vercel's Git integration makes environment management straightforward. Each branch maps to an environment with its own configuration and domain.

3.1 Project Configuration

In your Vercel dashboard, configure the following under Project Settings → Git:

1. **Production Branch:** Set to main. This auto-deploys to your production domain.
2. **Preview Branches:** All non-production branches get automatic preview deployments (useful for PR reviews).
3. **Ignored Build Step:** Optionally skip builds for non-code changes (e.g., docs-only commits).

3.2 Domain Configuration

You need to configure your domain DNS to point to Vercel. Here is the recommended setup:

Domain	Branch	DNS Record	Notes
yourdomain.com	main	A record → 76.76.21.21	Production (or app.yourdomain.com)
staging.yourdomain.com	staging	CNAME → cname.vercel-dns.com	Pre-release testing
dev.yourdomain.com	develop	CNAME → cname.vercel-dns.com	Development integration

Step-by-step domain setup:

1. **In Vercel:** Go to Project Settings → Domains. Add yourdomain.com, staging.yourdomain.com, and dev.yourdomain.com.
2. **Assign branches:** For each domain, click "Edit" and assign the corresponding Git branch (main, staging, develop).
3. **At your domain registrar:** Add the DNS records shown above. Vercel will verify them automatically.
4. **SSL:** Vercel provisions SSL certificates automatically. No action required.

3.3 Environment Variables

Vercel allows you to scope environment variables to specific environments (Production, Preview, Development). This is critical for pointing each environment to the correct Supabase project.

Variable	Production	Preview	Development
`NEXT_PUBLIC_SUPABASE_URL`	prod URL	staging URL	dev URL
`NEXT_PUBLIC_SUPABASE_ANON_KEY`	prod key	staging key	dev key
`SUPABASE_SERVICE_ROLE_KEY`	prod key	staging key	dev key

Security Note

Never commit environment variables to Git. Use Vercel's dashboard to manage them. The SUPABASE_SERVICE_ROLE_KEY should only be used in server-side code (API routes, serverless functions) and never exposed to the client.

3.4 Vercel Configuration File

Create a vercel.json in your repo root for consistent build settings:

```
{
  "buildCommand": "npm run build",
  "outputDirectory": ".next",
  "framework": "nextjs",
  "headers": [
    {
      "source": "/(.*)",
      "headers": [
        { "key": "X-Frame-Options", "value": "DENY" },
        { "key": "X-Content-Type-Options", "value": "nosniff" },
        { "key": "Referrer-Policy", "value": "strict-origin-when-cross-origin" }
      ]
    }
  ]
}
```

4. Supabase Multi-Environment Strategy

Each environment needs its own Supabase project to ensure complete isolation of data, auth users, storage, and edge functions. This prevents development work from ever touching production data.

4.1 Supabase Project Setup

1. **Create three Supabase projects:** yourapp-dev, yourapp-staging, and yourapp-prod. Use the same region for all three to minimise latency differences.
2. **Use Supabase CLI for migrations:** Install with npm install -g supabase. Initialise in your repo with supabase init.
3. **Link each environment:** Use supabase link --project-ref <ref> to connect your CLI to each project when applying migrations.

4.2 Database Migration Workflow

All schema changes must be managed through migration files, never through the Supabase dashboard in staging or production. The workflow is:

1. **Develop schema changes locally:** Use supabase db diff to generate migration files from dashboard changes, or write them manually.
2. **Commit migration files:** Store in supabase/migrations/ directory in your repo.
3. **Apply to dev:** supabase db push --linked (linked to dev project).
4. **Apply to staging:** Link to staging project and push. Verify everything works.
5. **Apply to production:** Link to prod project and push. Only after staging sign-off.

> **Seed Data**
> Create a supabase/seed.sql file with test data for dev and staging environments. Run with supabase db reset to tear down and rebuild with fresh seed data. Never use real user data in non-production environments.

4.3 Row-Level Security (RLS)

RLS policies must be part of your migration files. Every table should have RLS enabled and appropriate policies. Test policies thoroughly in dev before promoting. A common pattern:

```
-- Migration: enable RLS on profiles table
ALTER TABLE profiles ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can view own profile"
  ON profiles FOR SELECT
  USING (auth.uid() = user_id);

CREATE POLICY "Users can update own profile"
  ON profiles FOR UPDATE
  USING (auth.uid() = user_id);
```

5. CI/CD Pipeline (GitHub Actions)

Set up automated checks that run on every pull request. This is your quality gate — no code merges without passing these checks.

5.1 PR Check Workflow

Create .github/workflows/pr-checks.yml:

```
name: PR Checks
on:
  pull_request:
    branches: [develop, staging, main]

jobs:
  quality:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: 'npm'
      - run: npm ci
      - run: npm run lint
      - run: npm run type-check
      - run: npm run test
      - run: npm run build
```

5.2 Recommended package.json Scripts

```
{
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "lint": "eslint . --ext .ts,.tsx",
    "lint:fix": "eslint . --ext .ts,.tsx --fix",
    "type-check": "tsc --noEmit",
    "test": "vitest run",
    "test:watch": "vitest",
    "test:coverage": "vitest run --coverage",
    "format": "prettier --write .",
    "format:check": "prettier --check ."
  }
}
```

5.3 Deployment Flow Summary

Trigger	Branch	Deploys To	Auto/Manual	Approval
PR merged	develop	dev.yourdomain.com	Automatic	PR review

PR merged	staging	staging.yourdomain.com	Automatic	PR review + QA
PR merged	main	yourdomain.com	Automatic	PR review + sign-off
PR opened	feature/*	Preview URL	Automatic	None

6. Testing Strategy

Adopt a practical testing pyramid. For a startup, focus investment on the layers that give you the most confidence per hour of development time.

6.1 Testing Pyramid

Layer	Tool	What to Test	Coverage Goal	Run When
Unit	Vitest	Utility functions, hooks, logic	70%+ of utils	Every PR
Component	Testing Library	React components render correctly	Critical UI paths	Every PR
Integration	Vitest + MSW	API calls, Supabase interactions	Key flows	Every PR
E2E	Playwright	Full user journeys	Happy paths	Pre-release

6.2 What to Test First

Don’t aim for 100% coverage on day one. Prioritise based on business risk:

- **Authentication flows:** Sign up, login, password reset, session management
- **Payment/billing:** If applicable, test every financial transaction path
- **Data mutation:** Any operation that creates, updates, or deletes user data
- **Permissions:** Verify RLS policies actually enforce access control
- **Core user journey:** The primary value loop of your product

6.3 QA Process for Staging

Before any release to production, the following checklist must pass on staging:

- All automated tests pass
- Manual smoke test of core user flows
- Cross-browser check (Chrome, Safari, Firefox)
- Mobile responsive check
- Performance audit (Lighthouse score ≥ 80)
- No console errors or warnings
- Product owner sign-off on any UX changes

7. Design-to-Development Handoff

A clean handoff process between your UX designer and developers prevents misinterpretation, rework, and frustration. Since your designer uses Figma, Figma Make, and GitHub, you have excellent tooling for this.

7.1 Figma Organisation

- **Design System file:** Single source of truth for colours, typography, spacing, and reusable components. Developers reference this when building Tailwind config.
- **Per-feature design files:** Each feature/epic gets its own Figma page or file with all states, responsive variants, and edge cases.
- **Developer handoff mode:** Designer marks designs as "Ready for Dev" using Figma's Dev Mode. This exposes CSS values, spacing, and assets.

7.2 Handoff Workflow

1. Designer creates designs in Figma and links to the relevant GitHub issue/ticket.
2. Designer marks the design as "Ready for Dev" and notifies the team in Google Chat.
3. Developer reviews the design in Figma Dev Mode, extracts tokens, and asks questions as Figma comments.
4. Developer creates a feature branch and implements. PR includes screenshots compared against Figma.
5. Designer reviews the PR's Vercel preview URL and approves or requests changes via GitHub.

7.3 Design Tokens → Tailwind Config

Map your Figma design system directly into your tailwind.config.js to ensure pixel-perfect consistency:

```
// tailwind.config.js
module.exports = {
  theme: {
    extend: {
      colors: {
        brand: {
          primary: '#0F3460',    // From Figma design system
          secondary: '#16213E',
          accent: '#E94560',
        },
      },
      fontFamily: {
        sans: ['Inter', 'system-ui', 'sans-serif'],
      },
      spacing: {
        // Match Figma spacing scale
      },
    },
  },
}
```

8. Documentation Standards

Good documentation is especially critical with a remote/contract team. Invest in it early — it pays dividends when onboarding new developers and debugging issues at 2am.

8.1 Documentation Locations

Document Type	Location	Format	Owner
Architecture & decisions	docs/ in repo	Markdown ADRs	CTO / Tech Lead
API documentation	docs/api/ in repo	OpenAPI / Markdown	Backend devs
Setup & onboarding	README.md in repo root	Markdown	All devs maintain
Runbooks & ops	docs/runbooks/ in repo	Markdown	CTO / DevOps
Meeting notes & specs	Google Drive	Google Docs	Team
Design specs	Figma	Figma annotations	UX Designer

8.2 README.md Structure

Your repo's README should enable any new developer to go from zero to running the app locally within 30 minutes. Include:

- **Project overview:** One paragraph explaining what the app does
- **Prerequisites:** Node version, required CLI tools, accounts needed
- **Setup steps:** Clone, install, configure env, run. Numbered, copy-pasteable commands.
- **Environment variables:** Table listing every env var, what it does, and where to get its value
- **Common commands:** dev, build, test, lint, deploy
- **Architecture overview:** Brief description of folder structure and key patterns
- **Contributing guide:** Link to branching strategy and PR process (can reference this playbook)

8.3 Architecture Decision Records (ADRs)

Record significant technical decisions using ADRs. Store them in docs/decisions/ with a simple format:

```
# ADR-001: Use Supabase for backend services
## Status: Accepted
## Date: 2026-02-06
## Context: We need auth, database, and real-time features...
## Decision: We will use Supabase as our BaaS provider...
## Consequences: Vendor dependency, but rapid development...
```

9. Google Workspace Integration

Google Workspace serves as your communication and collaboration hub. Set it up to complement (not duplicate) your GitHub-based development workflow.

9.1 Google Chat Spaces (Channels)

Space Name	Purpose
#engineering	General dev discussion, daily standups, quick questions
#deployments	Automated deploy notifications from Vercel/GitHub (via webhook or Zapier)
#incidents	Production issues, on-call alerts, postmortems
#design	Design reviews, Figma links, UX feedback
#general	Team-wide announcements and social chat

9.2 Google Drive Structure

- **Engineering/** — Shared drive for all engineering docs
- **Engineering/Specs/** — Product requirements, feature specs
- **Engineering/Meetings/** — Sprint planning, retro notes, meeting recordings
- **Engineering/Onboarding/** — New developer guides, account setup checklists
- **Engineering/Architecture/** — High-level diagrams (link to ADRs in repo for details)

9.3 Meetings Cadence for Remote Teams

Meeting	Cadence	Duration	Tool
Daily standup	Daily	15 min (Google Meet) or async in Chat	Google Meet / Chat
Sprint planning	Bi-weekly	60 min	Google Meet
Design review	Weekly	30 min	Google Meet + Figma
Retrospective	Bi-weekly	45 min	Google Meet
1:1 with contractors	Weekly	30 min	Google Meet

10. Security & Access Control

With remote contractors, security boundaries are essential. Apply the principle of least privilege throughout.

10.1 Access Matrix

Service	CTO/Owner	Senior Dev	Contractor
GitHub repo	Admin	Maintain	Write
Vercel	Owner	Developer	Viewer (or none)
Supabase (prod)	Admin	Read-only dashboard	No access
Supabase (dev)	Admin	Admin	Developer
Google Workspace	Admin	Member	External collaborator
Domain registrar	Owner only	No access	No access

10.2 Security Essentials

- **Enable 2FA on all accounts:** GitHub, Vercel, Supabase, Google Workspace, domain registrar
- **Use GitHub Secrets for CI/CD:** Never hardcode tokens or keys in workflow files
- **Rotate credentials on contractor offboarding:** Remove access immediately, rotate any shared secrets
- **Review Supabase RLS policies regularly:** Audit quarterly or when adding new tables
- **Use .env.local for local development:** Add .env*.local to .gitignore. Provide a .env.example template.

11. Release Process

A structured release process protects production and gives everyone clarity on what's shipping and when.

11.1 Release Workflow

5. **Feature freeze:** Announce in #engineering that the develop branch is frozen for release preparation.
6. **Create release PR:** Open a PR from develop → staging. Title it "Release vX.Y.Z" with a changelog.
7. **QA on staging:** Team tests on staging.yourdomain.com using the staging QA checklist (Section 6.3).
8. **Fix any issues:** Bug fixes go directly into the staging branch, then cherry-pick back to develop.
9. **Promote to production:** Open PR from staging → main. Requires CTO/tech lead approval.
10. **Tag the release:** Create a GitHub release with tag vX.Y.Z and release notes.
11. **Monitor:** Watch error tracking and analytics for 30 minutes post-deploy.

11.2 Hotfix Process

For critical production bugs that cannot wait for the normal release cycle:

- **Branch from main:** git checkout -b hotfix/TICKET-critical-bug main
- Fix and test locally
- **PR directly to main:** Requires expedited review (CTO can approve)
- **After merge:** Cherry-pick the fix back into both staging and develop branches
- **Postmortem:** Document what happened and how to prevent recurrence

11.3 Semantic Versioning

Use semantic versioning (MAJOR.MINOR.PATCH) for all releases:

- **PATCH (1.0.1):** Bug fixes, no API/behaviour changes
- **MINOR (1.1.0):** New features, backwards compatible
- **MAJOR (2.0.0):** Breaking changes (rare at startup stage)

12. Local Development Setup

Every developer should be able to clone the repo and have a working local environment within 30 minutes. Here is the standard setup.

12.1 Prerequisites

- **Node.js 20+** (recommend using nvm for version management)
- **npm or pnpm** (pick one, standardise across team; pnpm recommended for speed)
- **Git** with SSH key configured for GitHub
- **Supabase CLI** (npm install -g supabase)
- **VSCode** with shared extensions (see 12.3)

12.2 First-Time Setup

```
# Clone the repo
git clone git@github.com:your-org/your-app.git
cd your-app

# Install dependencies
npm install

# Copy environment template
cp .env.example .env.local
# Fill in dev Supabase credentials (see team onboarding doc)

# Start Supabase locally (optional, for offline work)
supabase start

# Run the dev server
npm run dev
```

12.3 Shared VSCode Configuration

Commit a .vscode/ directory to your repo with shared settings to ensure consistency:

```
// .vscode/settings.json
{
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "editor.formatOnSave": true,
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": "explicit"
  },
  "typescript.preferences.importModuleSpecifier": "relative"
}

// .vscode/extensions.json
{
  "recommendations": [
    "esbenp.prettier-vscode",
```

```
    "dbaeumer.vscode-eslint",
    "bradlc.vscode-tailwindcss",
    "formulahendry.auto-rename-tag"
  ]
}
```

13. Monitoring & Observability

Set up monitoring early. It's far easier than diagnosing issues after users report them.

13.1 Recommended Stack

Concern	Tool	Free Tier?	Priority
Error tracking	Sentry	Yes (5K events/mo)	Day 1 — Critical
Analytics	Vercel Analytics or PostHog	Yes	Day 1 — Critical
Uptime monitoring	Better Uptime or UptimeRobot	Yes	Week 1
Performance	Vercel Speed Insights	Included with Vercel	Week 1
Logging	Vercel Logs + Supabase logs	Included	Built-in

14. New Developer Onboarding Checklist

Use this checklist when bringing on a new contract developer. Complete all items before they start writing code.

14.1 Access & Accounts

- GitHub: Invite as collaborator with Write role
- Vercel: Add to team with Developer role (or Viewer for contractors)
- Supabase: Add to dev project with Developer role
- Google Workspace: Add as external collaborator to relevant Spaces and Drive folders
- Figma: Add as viewer to design files
- Project management tool: Add to relevant boards/projects

14.2 Knowledge Transfer

- Walk through this Engineering Operations Playbook
- Pair on local environment setup (aim for under 30 minutes)
- Tour of the codebase: folder structure, key patterns, where to find things
- Review current sprint/backlog and assign a starter ticket
- Introduce to the team in #engineering Google Chat space
- Schedule weekly 1:1 for the first month

First PR Goal

Aim for the new developer to have their first PR merged within 2-3 days. Assign a small, well-defined ticket as their first task. This builds confidence and validates their setup is working correctly end-to-end.