Bontado Engineering Improvement Brief

Audience: Engineering team
Purpose: Help the team make Bontado easier to change, review, test, and operate reliably on AWS.

> **Ownership note:** DevOps operations—including AWS infrastructure, CI/CD setup, deployment, monitoring, and production operations—will be handled by Muhammed Mirza Kilic. The development team should coordinate with him on application-level requirements such as configuration, health checks, stateless operation, migrations, logging, queue behavior, and deployment compatibility.

Current position

Bontado is feature-rich and has accumulated a significant amount of complexity as it has evolved. The repository has roughly **143,642 physical lines of source**, **1,280 PHP files**, **245 Vue components**, and **116 Pinia stores**. Its automated test suite currently contains only the two default Laravel example tests, so meaningful business-flow coverage is still to be established. The main API route file is **1,009 lines**, the largest Vue component is **1,229 lines**, and frontend behavior is divided between the Vue admin SPA and Laravel/Blade customer storefronts.

The goal is not to criticize earlier work or rewrite everything at once. The recommended approach is to introduce clearer boundaries and safety checks gradually, allowing changes to be understood, tested, deployed, and rolled back independently.

Main improvement opportunities

Priority	Area	Current observation	Recommended next step
P0	Secrets and configuration defaults	`.env.example` contains an application key, Mailtrap credentials, a Google key, a reCAPTCHA secret, and a shared `VITE_API_KEY`. Anything prefixed with `VITE_` is delivered to the browser and cannot be treated as a secret.	Treat the current values as exposed, rotate them where applicable, replace them with placeholders, restrict client-side keys, use AWS Secrets Manager or SSM Parameter Store, and add secret scanning to CI.
P0	Automated test coverage	Only `ExampleTest.php` exists under Unit and Feature tests. Important behavior such as tenant isolation, permissions, orders, payments, subscriptions, and webhooks does not yet have automated regression coverage.	Begin with critical-path feature and integration tests: authentication, authorization, cross-tenant denial, checkout totals, idempotent order creation, payment/webhook verification, refunds, and plan limits. Make these tests required CI checks as coverage grows.

Priority	Area	Current observation	Recommended next step
P0	Inherited and potentially unnecessary scope	The codebase includes a wide range of payment and SMS integrations, generic platform modules, and legacy storefront code. Some appear to have been carried forward from previous projects or a broader template and may not be required for Bontado's intended launch scope. Every retained integration adds maintenance, security, testing, and support work.	Treat scope reduction as an immediate priority. Create a feature and integration inventory with four decisions: **keep**, **replace**, **defer**, or **retire**. Confirm usage with product and production data before removal, then delete unused packages, routes, settings, migrations, UI, and documentation through small tested PRs. Avoid carrying compatibility code without a named business owner.
P0	Commit conventions	Recent messages include `bug fix`, `bug solve`, and `remove unused file`; some commits include several kinds of change. This makes review, rollback, and incident analysis harder than necessary.	Keep each commit focused on one coherent change and its related files. Use messages such as `fix(checkout): reject orders outside delivery zone`, explaining **what** changed and **why**. Keep formatting/refactoring separate from behavioral changes where practical.
P1	Pull request size	Recent PRs changed 63 files / 4,856 lines, 35 files / 2,175 lines, and 57 files / 1,618 lines. Changes of this size are difficult to review thoroughly.	Aim for fewer than **300 changed lines** per PR where practical; PRs over **500 lines** should include a short reason and review plan. Split schema, refactor, API, UI, and infrastructure work into safe, ordered PRs. Include scope, relevant screenshots/API examples, test evidence, and migration/rollback notes.
P1	Frontend/backend boundaries	Vue is compiled inside the Laravel project, API contracts are implicit, and the customer experience still uses Blade/legacy assets. Frontend releases therefore cannot yet be versioned and deployed independently.	Define a versioned HTTP API with OpenAPI, stable error formats, explicit authentication/tenant context, and contract tests. Build and deploy frontend artifacts independently. This can remain a monorepo initially; separation means clear ownership and deployability, not microservices.

Priority	Area	Current observation	Recommended next step
P1	Runtime scalability	The current proof of concept uses local file sessions, file cache, local media, and a database queue. These choices are appropriate for a proof of concept but need adjustment before horizontal scaling.	Move toward stateless application containers: S3 for media, SQS for jobs, managed database storage, and Redis only when shared cache/session performance requires it. Run API, worker, and scheduler as separate processes.
P1	AWS production baseline	Docker/CloudFormation files currently describe a proof of concept, while repeatable production environments and a committed CI/CD pipeline still need to be formalized.	Manage AWS resources through reviewed IaC, with separate dev/staging/prod configuration, least-privilege IAM, backups, health checks, logs, alarms, deployment rollback, and documented recovery procedures. Prefer pipeline-driven releases over routine manual server changes.
P1	Security verification	Stored descriptions are rendered with `v-html`; the admin can influence environment/debug configuration; many payment/SMS adapters and tenant-aware endpoints create a broad security surface.	Sanitize rich HTML, move runtime `.env` editing out of normal admin flows, keep debug disabled in production, verify authorization server-side, and add SAST, dependency auditing, webhook-signature tests, rate-limit tests, and tenant-boundary tests. Plan an external penetration test before public launch.
P2	Code organization and repeated layers	A 1,009-line route file, components up to 1,229 lines, 116 stores, and near 1:1 controller/service classes add navigation overhead and boilerplate.	Organize code by business domain, split routes by domain, extract reusable composables/forms/tables, keep state local unless it is genuinely shared, and simplify pass-through service/controller layers that do not provide a useful boundary. Refactor incrementally behind tests.

Target AWS deployment shape

For the current Laravel transition, use the minimum production architecture that preserves a clean evolution path:

```
Route 53 + ACM
       |
CloudFront/WAF
  |             |
S3 frontend     ALB -> stateless API containers
                         |-- worker containers -> SQS
                         |-- scheduler (single controlled task)
                         |-- RDS MySQL (Multi-AZ when business critical)
                         |-- S3 media
                         `-- CloudWatch logs, metrics and alarms
```

Use `eu-central-1` as the default application region, subject to the final data-processing review. Start with the smallest justified capacity and scale from measured traffic. Avoid adding EKS, microservices, or always-on Redis before the workload demonstrates the need.

The Vue admin can move to S3/CloudFront once it no longer depends on Laravel's build/deployment lifecycle. The Blade customer storefront cannot become a static frontend merely by moving files; it must first be migrated to the agreed web application architecture while preserving SEO, tenant domains, checkout, payment redirects, and all existing behavior.

Delivery rules from now on

1. Use an approved emergency procedure for any necessary direct production change.
2. Every change starts with a small issue and explicit acceptance criteria.
3. Every behavioral change includes automated tests in the same PR.
4. CI must run formatting/linting, static analysis, unit/feature tests, frontend build, dependency audit, and secret scanning.
5. Database migrations must be backward-compatible and include rollback/recovery notes.
6. Make deployments repeatable through CI/CD and infrastructure as code.
7. Large features should be delivered behind flags through a sequence of reviewable PRs.