BCP — Bindu Control Panel

Development Guide & AI-Assisted Project Workflow

A practical, phased blueprint for building a secure, multi-tenant SaaS control platform that centrally manages Bindu IT, BinduHost, BinduMart, POS products, customers, subscriptions, support operations, and future SaaS products.

Version 1.0

Keep this document as the project’s source-of-truth guide.

1. Vision & Business Goals

Purpose

BCP is the central control plane for the Bindu ecosystem. It should let the owner manage tenants, SaaS products, subscriptions, billing, users, domains/subdomains, customer accounts, support operations, and integrations from one secure platform.

Business ecosystem

Primary ecosystem: Bindu IT as the parent/official brand, BinduHost for hosting and domain services, BinduMart for computer/accessories sales, and future POS/SaaS products such as computer-shop, mobile-shop, fashion/general-item, or pharmacy POS.

Core goals

Centralized administration; tenant lifecycle management; subscription and billing control; product/license management; customer management; unified support; team routing; auditability; and a clean foundation for future automation.

Success criteria

A new client can be provisioned safely, assigned a product and plan, billed monthly or yearly, suspended after non-payment without deleting data, reactivated after payment, and supported from the same control panel.

2. System Architecture

Architecture principle

Use a modular, service-oriented application boundary even if the first release is a modular monolith. Keep BCP separate from individual POS business logic.

Recommended high-level structure

BCP Admin/Core → Authentication & Authorization → Tenant Management → Product & License Management → Subscription/Billing → Customer Hub → Support/Unified Inbox → Integration/API Layer → Audit/Reporting → Notifications.

Tenant model

Treat each customer business as a tenant. Tenant context must be explicit and enforced server-side on every tenant-scoped request. Never trust a tenant ID supplied by the browser.

Deployment model

Prefer one maintainable codebase with isolated tenant data. Tenant access can be represented by subdomains such as client.example.com or a controlled path where appropriate. Keep the option open for custom domains later.

Key rule

BCP controls subscription state; a POS product should consume a small licensing/status interface rather than implementing its own billing engine.

3. Database & Multi-Tenant Data Architecture

Design goals

Strong tenant isolation, clear relationships, safe migrations, auditability, and predictable performance.

Core entities

Users, roles, permissions, tenants, products, product plans, subscriptions, invoices, payments, licenses, domains, API credentials, support conversations, tickets, teams, assignments, notifications, audit logs, and integration records.

Isolation

Every tenant-owned record should have a tenant scope where appropriate. Enforce this in application services/repositories and, where supported by the database, with additional database-level controls.

Billing data

Keep subscription status, billing period, next billing date, grace period, invoice state, and payment references separate from product business data.

Data safety

Use migrations, foreign keys, unique constraints, indexes, soft deletion where appropriate, and transactional updates for lifecycle operations.

4. Authentication & Security

Identity

Use secure authentication with strong password hashing, session/token protection, optional MFA, password reset, email verification, and login throttling.

Authorization

Implement RBAC with roles such as Super Admin, Admin/Manager, Billing, Support, Product/Technical, and limited Staff. Permissions should be granular and server-enforced.

Tenant security

A user must never access another tenant’s data by changing a URL, ID, API payload, or browser state.

Operational security

Protect secrets, API keys, webhooks, payment callbacks, file uploads, and administrative actions. Use CSRF protection where applicable, secure cookies, HTTPS, validation, output encoding, and rate limiting.

Audit

Record sensitive actions such as subscription changes, suspension/reactivation, role changes, API key creation/revocation, payment state changes, and tenant deletion requests.

5. Billing & Subscription Engine

Plans

Support monthly and yearly plans and allow future plan variants, trials, discounts, add-ons, or usage-based components without rewriting the core model.

Lifecycle

Recommended states: trialing → active → past_due/grace → suspended → cancelled. Exact names may follow the chosen payment provider.

Non-payment

Do not delete a tenant when a bill is overdue. Use reminders, a configurable grace period, then suspend access while retaining data. Reactivation should restore service after successful payment.

Invoices & payments

Store invoice status and provider references separately from application transactions. Use webhooks or verified callbacks where the payment provider supports them.

License check

Each SaaS product should be able to ask BCP whether its tenant/license is active. Cache cautiously and define a safe behavior for temporary BCP connectivity failures.

6. Super Admin Panel & Customer Management

Dashboard

Show active tenants, trials, overdue accounts, suspended accounts, revenue/billing summaries, recent activity, support workload, and system health.

Tenant management

Create, view, edit, suspend, reactivate, archive, and inspect tenants. Show assigned products, plan, billing status, users, domains, and activity.

Customer profile

Maintain a central customer record so support and sales staff can see relevant products, subscriptions, invoices, conversations, tickets, and history in one place.

Teams

Create teams such as Hosting, BinduMart/Sales, Technical, Billing, and General Support. Assign users and permissions to teams.

7. API & Product Integration

Integration model

Expose a small, versioned API for product provisioning, tenant identification, license status, subscription state, and selected customer/account operations.

Authentication

Use scoped API credentials or signed tokens. Rotate and revoke credentials from BCP.

POS integration

A POS should not own subscription billing. It should validate the tenant/license through a controlled integration and gracefully handle temporary API failures.

Webhooks

Use signed webhooks for events such as subscription activated, payment received, subscription suspended, or product entitlement changed. Make webhook handlers idempotent.

Versioning

Version external APIs from the beginning, e.g. /api/v1, and avoid breaking existing clients.

8. AI-Assisted Development Workflow

Rule 1 — inspect first

Before changing code, the AI must inspect the current codebase, database schema, routes, models, services, views, tests, and existing integrations relevant to the task.

Rule 2 — source of truth

The current codebase and project documentation are the source of truth. Never assume what a previous AI model intended.

Rule 3 — audit before implementation

For every phase, classify requirements as Complete, Partial, Missing, or Risky. Do not rewrite completed work.

Rule 4 — small changes

Implement one logical phase at a time. Preserve existing behavior and avoid unrelated refactors.

Rule 5 — verify

After implementation, run tests, inspect migrations, check logs/errors, and verify the existing features that could be affected.

Model handoff prompt

When switching between Claude, Gemini, GPT, or another coding model: instruct the new model to inspect the current codebase, read the project docs/change log, identify exactly where the previous model stopped, and continue only from the verified incomplete work.

9. Master Prompt & Phase Prompt Library

Master instruction

You are working on an existing production-oriented BCP codebase. Do not rebuild working functionality. Inspect first, document the current state, identify the exact gap, plan the smallest safe change, implement it, test it, and update the project change log.

Phase template

Every phase prompt should contain: Objective → Existing behavior → Required behavior → Files/modules to inspect → Constraints → Acceptance criteria → Tests → Documentation update.

Handoff record

Maintain a short DEVELOPMENT_STATUS.md or equivalent containing: current phase, completed items, partial items, pending items, changed files, migrations, tests, known issues, and next recommended step.

Why this matters

The status document prevents context loss when changing AI models or returning to the project after a long break.

10. Best Practices & Future Roadmap

Operational best practices

Use Git branches, frequent commits, database backups before migrations, staging before production, environment-specific secrets, automated tests, structured logs, monitoring, and rollback plans.

Future roadmap

Custom domains, automated subdomain/SSL provisioning, payment gateway integration, reseller/partner accounts, customer portal, invoices, knowledge base, notification center, advanced analytics, mobile administration, and deeper SaaS provisioning.

Customer Operations / Unified Inbox

Add a central Customer Hub where messages from Bindu IT, BinduHost, BinduMart, and future products can be routed to the correct team. Hosting questions can go to Hosting Support; product questions to BinduMart/Sales; billing issues to Billing; technical issues to Technical Support.

Smart routing

A conversation can be classified by source, product, topic, priority, customer, and team. Staff should work from one interface while permissions restrict what they can access.

AI support

AI can suggest classifications, summaries, knowledge-base answers, and draft replies. Human staff should remain able to review and send responses.

Project Completion Checklist

☐ Architecture reviewed and documented

☐ Multi-tenant isolation verified

☐ Authentication and RBAC implemented and tested

☐ Subscription lifecycle implemented

☐ Invoices/payments integrated safely

☐ Tenant suspension/reactivation tested

☐ Super Admin tenant/customer management completed

☐ API and webhook security reviewed

☐ Audit logging enabled for sensitive operations

☐ Customer Hub / unified support architecture documented

☐ Automated tests and regression tests passing

☐ Production backup and rollback procedure documented

☐ DEVELOPMENT_STATUS.md updated before every AI-model handoff

> Golden Rule: BCP should evolve as a controlled platform. Never let an AI coding session blindly rewrite working modules. Inspect → Plan → Implement → Test → Document → Handoff.