M5nity Global Corp Platform – System Documentation

1. Introduction

- **Purpose**: Provide a comprehensive technical reference for developers, operators, and stakeholders maintaining the M5nity Global Corp membership platform.
- **Scope**: Covers system architecture, feature modules, critical services, data structures, deployment workflows, and operational practices based on the current codebase (E:\larave-starter-kit\my-app).
- **Audience**: Full-stack engineers, DevOps, QA, technical onboarding teams.

2. System Overview

- **Business domain**: Membership platform for managing a binary network, cashier-assisted onboarding, product sales, and commission payouts.
- **Core capabilities**:
  - Binary genealogy visualisation and placement for members.
  - Registration pin lifecycle managed by cashier staff.
  - Point-of-sale (POS) workflow with unilevel commission distribution.
  - Account settings (profile, credentials, two-factor auth).
  - Multi-role dashboards (member, cashier, admin, superadmin, accounting).
- **Interaction model**: Laravel 12 backend serving JSON/Inertia responses to a React 19 SPA delivered via Vite.

3. Architecture & Technology Stack

- **Backend**:
  - Laravel Framework 12 (composer.json).
  - Authentication via Laravel Fortify with optional two-factor.
  - Inertia.js server adapter for React-powered views.
  - Custom services for compensation logic (•pp/Services).
  - Database-backed queues for asynchronous workloads (queue driver configured in .env).
- **Frontend**:
  - React 19 with TypeScript (esources/js).
  - Tailwind CSS 4 for styling; Radix UI primitives for accessible components.
  - Inertia routing with a sidebar-focused application shell.
- **Persistence**:
  - Relational database (default MySQL; .env uses DB_CONNECTION=mysql).
  - Queue, cache, and session drivers default to the database.
- **Tooling**:
  - Vite 7 bundler, ESLint 9, Prettier 3, TypeScript 5.
  - PHPUnit 11 for automated tests; Laravel Pint for styling.
- **Runtime topology**:
  - Web server (php artisan serve) + queue worker (queue:listen) + Vite dev server (pm run dev).
  - Production build via pm run build and standard Laravel deployment on PHP 8.2+.

4. Application Modules

4.1 Member Experience

- **Dashboards**: MemberOverviewController, MemberDashboardController surface genealogy tree, payouts, and pins via Members/Binary/Dashboard.

- **Binary tree**: esources/js/pages/Members/Binary/Treeview.tsx renders nodes sized by odeSize with overlays for placement metadata.
- **Registration**: MemberRegistrationController::store consumes cashier-issued pins, ensures sponsor eligibility, finds placement slots, and triggers pairing/commissions.
- **Pins & income**: MemberPinController, MemberIncomeController expose pin inventory and pairing history lists via Inertia pages.

4.2 Cashier Workspace

- **Registration hub**: CashierRegistrationController allows staff to pre-register members, generate transaction numbers/pins (IdentifierGenerator), and view recent cashier transactions.
- **POS**: CashierPosController manages product sales, records purchases, and invokes UnilevelCommissionService to cascade commissions up the genealogy.
- **UI**: esources/js/pages/Cashier/Registration.tsx and Cashier/POS.tsx provide high-level forms with toast feedback (esources/js/lib/notifier.ts).

4.3 Administration & Oversight

- **Dashboards**: Controllers for admin (AdminDashboardController), superadmin (SuperadminDashboardController), and accounting (AccountingDashboardController) render placeholder Inertia pages for future KPIs.
- **Settings**: outes/settings.php with controllers for profile, password, appearance, and Fortify-powered two-factor flows.
- **Middlewares**:
  - EnsureMember, EnsureCashier, EnsureRole enforce role-specific access using utype and staff profiles.
  - HandleAppearance injects theme preferences.

5. Domain Services & Support Utilities

- **BinaryPairingService (•pp/Services/BinaryPairingService.php)**:
  - Processes genealogy placements, pairs left/right branches up to level 10, records MemberPairingHistory and MemberIncomeHistory.
  - Uses a queue table (members_pairing_queue) for asymmetrical placements until a matching side appears.
- **UnilevelCommissionService (•pp/Services/UnilevelCommissionService.php)**:
  - Walks ancestor chain up to configured max (default 15), writing Commission records and updating MemberMaintenance for monthly activity.
  - Relies on CompensationService for percentages (CompensationSetting::current).
- **CompensationService**:
  - Central accessor for referral bonus, maintenance minimum, and unilevel percentages; defaults provided on first access.
- **IdentifierGenerator (•pp/Support/IdentifierGenerator.php)**:
  - Generates member IDs (yyPH########) and transaction+pin codes based on incremental IDs plus random suffixes.
- **BinaryService** (utility layer) & BinaryTestPairingCommand CLI command support testing the binary logic.

6. Data Model Summary

Table	Purpose	Key Columns / Relationships

users	Core auth accounts with role type (utype) and optional master_password_id.	id,
ame, email, utype, master_password_id		
master_passwords	Stores hashed master passwords for privileged actions.	
ame, password		
staff_profiles	Extended profile for cashier/admin staff (user_id FK).	
ole, department, contact_number		
members_info	Personal data for member users; linked to users.	MID, email, ♀name, is_active, kick_start_token
members_account	Represents binary accounts owned by members.	member_id, •ccount_name, dsponsor, under_sponsor, upper_nodes
genealogies	Binary tree nodes referencing members_account.	parent_id, position, level, pair_value
members_pin	Registration pins with sponsor linkage and status.	rans_no, payment_method,
ew_member_id, pin, status		
ransaction_histories	Cashier log for issued pins/payments.	cashier_id, member_pin_id, payment_method
members_pairing_queue	Pending pairings for ancestors lacking opposite-side node.	•ncestor_account_id, side, level
members_pairing_history	Completed pairings with MemberIncomeHistory.	left_account_id,
ight_account_id, •mount, paired_at		
members_income_history	Audit of income generated per pairing.	•ncestor_account_id, pairing_history_id, •mount
payouts	Cash-out events per user.	user_id, •mount, ype, level
commissions	Tracks referral and unilevel payouts.	member_account_id, source, level, •mount, purchase_id
products	POS catalogue.	sku,
ame, price, is_active		

purchases	POS orders keyed by transaction number.	cashier_id, □uyer_account_id, otal_amount, paid_at
purchase_items	Line items for purchases.	product_id, quantity, unit_price, subtotal
members_maintenance	Tracks monthly spend and activation.	members_account_id, month, otal_spent, •ctivated_at
compensation_settings	System-wide compensation parameters (JSON).	
eferral_bonus, unilevel_percents		
Queue/cache/session tables	Standard Laravel tables (jobs, cache, sessions, etc.).	—

Seeders (database/seeders) populate master password, genealogy baseline, staff profiles, sample pins, products, and compensation settings for a full demo environment.

7. Key Workflows

7.1 Cashier Pre-Registration

1. Cashier authenticates (EnsureCashier).
2. CashierRegistrationController::store validates payer info, creates a User + MemberInfo, generates IDs (IdentifierGenerator), issues MemberPin, and records TransactionHistory.
3. Returns JSON or Inertia redirect with success flash.

7.2 Member Placement via Binary Tree

1. Authenticated member selects an unused pin.
2. MemberRegistrationController::store validates sponsor eligibility, locates nearest open slot (BFS), locks and consumes the pin, creates MemberAccount + Genealogy.
3. Triggers BinaryPairingService::handleNewPlacement to evaluate pairings up the ancestry.
4. Awards referral bonus from CompensationService when configured, recording a Commission.

7.3 POS Purchase & Commission Cascade

1. Cashier selects active products and member account within CashierPosController::index.
2. store validates line items, creates Purchase and associated PurchaseItem rows, logs to controller-specific file (CashierPOS_logs.log).
3. UnilevelCommissionService::distributeForPurchase walks the parent chain, creates Commission records per level, updates MemberMaintenance.
4. Responses delivered via JSON or redirect with success flash.

7.4 Binary Pairing Propagation

- On each placement, BinaryPairingService:
  - Queries members_pairing_queue for opposite-side entries at each ancestor level.
  - When match found, records MemberPairingHistory + MemberIncomeHistory, removes queue entry.
  - When match not found, enqueues the new node for future pairing.

- Stops after level 10 or when ancestry ends.

8. Frontend Structure

- **Entry point**: esources/js/app.tsx sets up Inertia, theming (initializeTheme), and toast handling.
- **Layouts**: esources/js/layouts/app/app-sidebar-layout.tsx composes AppShell, AppSidebar, breadcrumbs, and content. Auth layout under esources/js/layouts/auth-layout.tsx.
- **Routing**: Pages under esources/js/pages mirror Inertia route names (e.g., Members/Binary/Dashboard.tsx, Cashier/POS.tsx).
- **UI Components**: esources/js/components/ui contains sidebar, nav, tables, Radix wrappers, and toast provider integration.
- **Type definitions**: esources/js/types/index.d.ts supplies shared interface contracts.
- **Styling**: Tailwind config via Vite plugin; global styles in esources/css/app.css.

9. Security, Roles & Access Control

- **Authentication**: Laravel Fortify handles login, registration (optional), password resets, two-factor flows; Fortify features toggled in outes/web.php.
- **Roles & Middleware**:
  - EnsureMember: Confirms user->utype === 'member' and account active.
  - EnsureCashier: Checks utype === 'cashier' and staff profile readiness.
  - EnsureRole: Generalised role gate based on provided parameter (e.g., ole:admin).
- **Session & rate limiting**: Fortify two-factor routes use throttle middleware (hrottle:6,1).
- **Logging**: Base Controller exposes writeControllerLog for per-controller log files in storage/logs, improving auditability without polluting Laravel log stack.
- **Data protection**: Uses hashed passwords with bcrypt (12 rounds). Personal data housed in dedicated tables; queue updates run within transactions to ensure consistency.

10. Configuration & Deployment

- **Environment variables** (.env):
  - APP_URL, APP_DEBUG, APP_LOCALE, LOG_CHANNEL.
  - Database connection defaults to MySQL (DB_* keys).
  - Queue/cache/session drivers configured to database.
  - Mailer defaults to log for local development.
- **Initial setup**:
  1. composer install
  2. cp .env.example .env then configure DB credentials.
  3. php artisan key:generate
  4. php artisan migrate --seed
  5. pm install
  6. pm run build (or pm run dev for development).
- **Development loop**: composer run dev launches PHP server, queue listener, and Vite simultaneously via px concurrently.
- **Production build**: pm run build (optionally pm run build:ssr for server-side rendering). Deploy public/build assets alongside Laravel deployment process.
- **Queue workers**: run php artisan queue:listen --tries=1 (already orchestrated in composer dev script). Use Supervisor/systemd in production for persistence.

11. Testing & Quality Assurance

- **Automated tests**:
  - `ests/Feature/CompensationPlanTest.php verifies binary pairing cap (≤ level 10), unilevel commission cap (≤ level 15), income record parity, maintenance activation.`
  - Uses Laravel RefreshDatabase trait to isolate each scenario.
- **Static analysis & linting:**

  pm run lint (ESLint with React hooks rules).

  pm run types (TypeScript --noEmit).

  pm run format:check (Prettier + Tailwind plugin + import organiser).
- **Manual QA guidance**:
  - Validate new seed data via Inertia dashboards.
  - Monitor log files (storage/logs/*_logs.log) for errors during feature flows.

12. Monitoring & Logging

- Controller-specific log files:
  - MemberDashboard_logs.log, MemberRegistration_logs.log, CashierRegistration_logs.log, CashierPOS_logs.log, BinaryPairing_logs.log, etc.
  - Located under storage/logs; rotate/ship via log aggregation in production.
- Standard Laravel logging still available via laravel.log.
- Test command inary:test-pairing assists in reproducing pairing logic; combine with log review.

13. Deployment & Operations Considerations

- **Database migrations**: Extensive domain tables; review for downstream schema changes before deployment.
- **Seeds**: Designed for demo/dev; disable destructive truncation (GenealogySeeder resets tables) in production.
- **Background jobs**: Ensure queue worker runs to process pairing side-effects triggered through services (pairing uses synchronous transactions but future async jobs can be introduced).
- **Performance**:
  - Genealogy traversals use eager loading & BFS; consider caching heavy tree queries for large networks.
  - UnilevelCommissionService uses iterative parent lookups; add indexes on genealogies.parent_id and members_account.member_id (already present via migrations) and monitor for N+1 growth.
- **Security**:
  - Enforce HTTPS via APP_URL.
  - Populate .env mail credentials for production notifications.
  - Configure queue/database credentials with least privilege.

14. Future Enhancements (Recommendations)

- Introduce centralised logging/monitoring (e.g., Laravel Telescope, Sentry) for faster incident response.
- Extend test coverage to cashier flows and member placement edge cases.
- Implement role-based dashboards with real KPIs and summarised metrics.

- Automate PDF/email export of genealogy or commission statements.
- Consider background jobs for heavy commission calculations to avoid long-running HTTP requests.

Document location: docs/system-documentation.md
Last generated: 2025-10-24