RFC 001: Phase 0 – SaaS Multi-Tenant Foundation (v2.0 Revision)

Project: Printeast (The AI-Native POD OS)

Phase: 0 (Weeks 1–2)

Status: READY FOR IMPLEMENTATION By: Soumyadyuti Dey

Date: January 23, 2026

---

1. Executive Summary

Phase 0 establishes the "Construction of the Factory." We are building the high-scale infrastructure that enables an **AI-native, SaaS-ready POD platform**. This phase implements a multi-tenant architecture with a shared-database and strict discriminator-column isolation.

By the end of Week 2, we will deliver a **"Walking Skeleton"**: A fully deployed, end-to-end connected system where a user can Register, verify via Magic Link, select one of **7 Roles (RBAC)**, and land on a secured, tenant-isolated Dashboard.

Success Criteria:

1. **CI/CD:** Automated pipelines for linting, type-checking, and staging deployment.
2. **Auth:** Zero-trust JWT flow with **Redis session whitelisting** and Magic Link verification.
3. **Data:** 15-table relational schema with strictly enforced **Tenant Isolation** at the Prisma level.
4. **UX:** Parallax Landing Page + 7 Role-based Dashboard Shells (Admin, Creator, Seller, etc.) live.

---

2. Technical Stack

2.1 Core Architecture

Monorepo Manager: TurboRepo. Orchestrates parallel builds and remote caching across `apps/` and `packages/`.

Package Manager: pnpm. Utilizes content-addressable storage for disk-efficient dependency management.

Language: TypeScript v5.x. Strict mode enabled (`noImplicitAny`, `strictNullChecks`).

2.2 Frontend (The Interface)

Framework: Next.js 16 (App Router). Uses React Server Components (RSC) for optimized data fetching.

Styling: Tailwind CSS. Custom theme extension for **Printeast Pink (#E23E83)** and **Orange (#FF7A39)**.

Typography: Poppins (Headers) + **Inter** (UI & Data).

State Management: Zustand (UI state) + **TanStack Query v5** (Server state/caching).

2.3 Backend (The Brain)

Runtime: Node.js (LTS).

Framework: Express.js v4.x with custom TypeScript wrappers for type-safety.

Security: `helmet` for header security, `cors` for origin control, and `express-rate-limit`.

Observability: Pino for high-performance JSON logging and structured error handling.

2.4 Data & Infrastructure

Database: PostgreSQL 16. Hosted on Supabase (Managed) with **pgvector** for AI design search.

ORM: Prisma. Used as a type-safe query builder with Middleware for automatic Tenant Filtering.

Caching: Redis. Used for JWT refresh-token whitelisting and rate-limit tracking.

File Storage: Supabase Storage (S3-compatible) + CloudFront CDN for global asset delivery.

Hosting:

- **Frontend:** Vercel (Edge-optimized).
- **Backend: AWS** (Dedicated instance for high-scale "Iron Rails" stability and trial-period efficiency).

---

3. Monorepo File Structure

Plaintext

```
/printeast-monorepo
 ├── /apps
 │   ├── /web                  # Next.js 16 Application
 │   │   ├── /src/app          # App Router (Routes &
Layouts)
 │   │   ├── /src/features     # Domain logic (auth, studio,
shop)
 │   │   ├── /src/components   # PRINTEAST-UX Kit
(Glassmorphism)
 │   │   └── /src/hooks        # Custom React Hooks
(useAuth, useTenant)
 │   │
 │   └── /backend              # Node.js Express API
 │       ├── /src/middleware   # TenantGuard, AuthGuard,
RoleGuard
 │       ├── /src/modules      # Domain modules (Auth,
```

```
Tenants, Design)
 │       ├── /src/models        # Zod schemas for validation
 │       └── main.ts            # Server entry point
 │
 ├── /packages
 │   ├── /database              # Shared Prisma Client &
Migrations
 │   ├── /types                 # Shared TS Interfaces & Zod
Schemas
 │   ├── /ui                    # Atomic Design System
(@printeast/ui)
 │   └── /config                # Shared ESLint, Tailwind, &
TSConfigs
 │
 ├── /docker                    # Local Infrastructure
 │   └── docker-compose.yml     # Postgres (pgvector), Redis,
pgAdmin
 └── turbo.json                 # Pipeline task definitions
```

---

4. Database Schema (15 Core Tables)

Implementation: Prisma Schema (`schema.prisma`).

4.1 Identity & Access Control

- **Tenants:** `id (UUID)` , `name` , `slug (Unique Index)` , `tier (FREE, PRO, ENT)` , `metadata (JSONB)` .
- **Users:** `id` , `tenant_id (FK)` , `email (Indexed)` , `passwordHash` , `status (Enum)` , `createdAt` .
- **Roles:** `id` , `name (7 Types: Admin, Creator, Seller, Buyer, Supplier, Support, Visitor)` , `permissions (JSONB)` .
- **UserRoles:** `user_id (FK)` , `role_id (FK)` . (Composite Primary Key).
- **AuditLogs:** `id` , `tenant_id (FK)` , `user_id (FK)` , `action` , `resource` , `changes (JSONB)` .

4.2 Commerce & Operations

- **Product:** `id`, `tenant_id (FK)`, `name`, `basePrice`, `sku`, `metadata (JSONB)`.
- **Order:** `id`, `tenant_id (FK)`, `buyer_id (FK)`, `status (Enum)`, `totalAmount`, `trackingNumber`.
- **OrderItem:** `id`, `order_id (FK)`, `product_id (FK)`, `design_id (FK)`, `priceAtTime`.
- **Category:** `id`, `name`, `slug`, `parentId (Nullable FK)`.
- **Supplier:** `id`, `name`, `location`, `contactEmail`, `integrationKey`.
- **Wallet:** `id`, `user_id (FK)`, `balance`, `currency`.
- **Transaction:** `id`, `wallet_id (FK)`, `amount`, `type (Debit/Credit)`, `reference_id`.
- **Inventory:** `id`, `product_id (FK)`, `quantity`, `warehouseLocation`.
- **Review:** `id`, `user_id (FK)`, `product_id (FK)`, `rating`, `comment`.
- **Notification:** `id`, `user_id (FK)`, `type`, `content`, `isRead`.

---

5. API Architecture & Security

5.1 Authentication Flow (JWT + Redis)

1. **Access Token:** Stored in memory (Frontend). Signed with RS256. Expiry: 15 minutes.
2. **Refresh Token:** Stored in **httpOnly**, **Secure**, **SameSite=Strict** Cookie. Expiry: 7 days.
3. **Revocation:** All active JWT IDs are whitelisted in **Redis**. On logout, the ID is deleted from Redis, effectively revoking access immediately across all devices.

5.2 Security Protocols

- **Tenant Isolation:** A custom `IsolationMiddleware` intercepts Prisma calls to ensure the `tenant_id` is automatically injected into every `where` clause.

- **Rate Limiting:** Redis-backed. 100 requests per 15 minutes per IP.
- **Cost Management: AWS**-level monitoring with budget alerts and strict usage thresholds to prevent surprise overages.

---

6. Frontend Strategy

6.1 Route Groups & Protected Layouts

- `(marketing)` : Public-facing parallax Landing page with Hero, Inspiration Grid, and Trust Bar.
- `(auth)` : Split-screen Login/Register/Verification. Features an animated Laptop Frame slider showcasing workflow.
- `(dashboard)/[tenant_slug]` : Requires **AuthGuard**.
  - **[role]/**: Requires **RoleGuard** (validates against the 7 roles in the user session).

6.2 Design System Implementation

JavaScript

```javascript
// tailwind.config.js
theme: {
  extend: {
    colors: {
      printeastPink: '#E23E83',
      printeastOrange: '#FF7A39',
      neutralBase: '#F8F9FC',
    }
  }
}
```

---

7. Infrastructure & Deployment (CI/CD)

7.1 GitHub Actions Pipeline

1. **Lint & Format:** Ensures style consistency.
2. **Type Check:** `tsc --noEmit` across all packages.
3. **Build:** Verifies that both apps and packages build correctly via Turbo Repo.
4. **Deploy: - Frontend** → Vercel (Edge-optimized).
   - **Backend** → **AWS** (Persistent Cloud Environment for backend logic).

---

8. Phase 0 Deliverables Checklist

Week 1: Hard Infrastructure & Data

- [ ] Initialize **TurboRepo** & `pnpm` workspaces with shared `@printeast/database`.
- [ ] Setup **Docker Compose** for local PostgreSQL (pgvector) & Redis 7.
- [ ] Define and migrate 15-table **Prisma schema** with Multi-tenant discriminator logic.
- [ ] Implement **Express Auth Engine** (Register/Login/JWT/Redis Whitelisting).
- [ ] Setup **Supabase Storage** and CloudFront CDN for design assets.

Week 2: Shells & Connectivity

- [ ] Launch Parallax Landing Page (Hero, Masonry Inspiration Grid, Trust Bar).
- [ ] Create **7 Role Dashboard Layouts** with "Three-Zone" architecture and state.
- [ ] Implement **Next.js Middleware** for dynamic RBAC redirects.
- [ ] Deploy full stack to staging environments (**AWS** backend + Vercel frontend).
- [ ] Complete **OpenAPI/Swagger** documentation for all Phase 0 endpoints.

Status: READY FOR IMPLEMENTATION