RFC 001: Phase 0 – Core MVP Foundation (Final Blueprint)

Project: Printeast (The AI-Native POD OS)

Phase: 0 (Weeks 1–2)

Status: `READY FOR IMPLEMENTATION`

By: Soumyadyuti Dey

Date: January 20, 2026

1. Executive Summary

Phase 0 is the "Construction of the Factory." We are not building features (like the Design Studio) yet; we are building the infrastructure that makes those features possible.

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, Login, be assigned a Role (RBAC), and land on a secured Dashboard.

Success Criteria:

1. **CI/CD:** Automated pipelines for linting, type-checking, and staging deployment.
2. **Auth:** Zero-trust JWT flow with secure cookie handling and Redis session whitelisting.
3. **Data:** 15-table relational schema with strictly typed Prisma Client.
4. **UX:** Landing Page + Role-based Dashboards (Admin, Creator, Seller) live.

2. Technical Stack (Finalized)

2.1 Core Architecture

- **Monorepo Manager: TurboRepo**. Orchestrates builds across apps/packages.
- **Package Manager:** `pnpm`. Utilizes content-addressable storage for fast, disk-efficient dependency management.
- **Language: TypeScript v5.x**. Strict mode, `noImplicitAny`, and `exactOptionalPropertyTypes` enabled.

2.2 Frontend (The Interface)

- **Framework: Next.js 14** (App Router). Uses React Server Components (RSC) for optimized data fetching.
- **Styling: Tailwind CSS**. Custom theme extension for Printeast Pink (#E23E83) and Orange (#FF7A39).
- **State Management: Zustand** (Client state) + **TanStack Query v5** (Server state/caching).
- **Forms: React Hook Form** + **Zod** (Schema-based client-side validation).

2.3 Backend (The Brain)

- **Runtime: Node.js (LTS)**.
- **Framework: Express.js v4.x** with TypeScript wrappers.
- **Security:** `helmet` for header security, `cors` for origin control, `express-rate-limit` for DDoS protection.
- **Logging: Pino** (JSON logging for cloud-native observability).

2.4 Data & Infrastructure

- **Database: PostgreSQL 16**. Hosted on Supabase (Managed) for MVP speed.
- **ORM: Prisma**. Used for migrations and as a type-safe query builder.

- **Caching: Redis**. Used for JWT refresh-token whitelisting and rate-limit tracking.
- **File Storage: Supabase Storage** (S3-compatible). Handles all design assets and mockups.
- **Hosting:**
    - **Frontend:** Vercel (Edge-optimized).
    - **Backend:** Render (Web Service) with auto-scaling enabled.

---

3. Monorepo File Structure

Plaintext

```
/printeast-monorepo
├── /apps
│   ├── /web                       # Next.js 14 Application
│   │   ├── /app                   # App Router (Routes &
Layouts)
│   │   ├── /components            # UI Library (Atomic Design:
atoms, molecules)
│   │   ├── /hooks                 # Custom React Hooks (useAuth,
useOrders)
│   │   └── /services              # API Client (Axios/Fetch
wrappers)
│   │
│   └── /backend                   # Node.js Express API
│       ├── /src
│       │   ├── /controllers       # Request/Response
orchestration
│       │   ├── /middleware        # AuthGuard, RoleGuard,
ErrorHandler
│       │   ├── /models            # Zod schemas for
Request/Response validation
│       │   ├── /routes            # Versioned API routes
(/v1/auth, /v1/users)
│       │   └── /services          # Business logic & DB
interaction
```

```
│       └── server.ts          # Entry point
│
├── /packages
│   ├── /database              # Shared Prisma Client &
Migrations
│   ├── /types                 # Shared TS Interfaces & Zod
Schemas
│   ├── /config-eslint         # Shared ESLint configurations
│   └── /config-tailwind       # Shared Tailwind Theme &
Tokens
│
├── /docker                    # Local Development
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

- **User:** `id (UUID)` , `email` , `passwordHash` , `role (Enum)` , `isVerified` , `createdAt` .
- **Session:** `id` , `userId (FK)` , `token` , `expiresAt` , `userAgent` , `ipAddress` .

4.2 Commerce Engine

- **Product:** `id` , `name` , `basePrice` , `sku` , `supplierId (FK)` , `metadata (JSONB)` .
- **Order:** `id` , `userId (FK)` , `status (Enum)` , `totalAmount` , `shippingAddress (JSONB)` , `trackingNumber` .
- **OrderItem:** `id` , `orderId (FK)` , `productId (FK)` , `designId (FK)` , `quantity` , `priceAtTime` .

- **Category:** `id`, `name`, `slug`, `parentId (Nullable FK)`.

4.3 Creator & Designer Assets

- **Design:** `id`, `userId (FK)`, `imageUrl`, `promptText`, `vectorEmbedding (pgvector)`, `status (Enum)`.
- **Creator:** `id`, `userId (FK)`, `bio`, `portfolioUrl`, `commissionRate`.

4.4 Operations & Finance

- **Supplier:** `id`, `name`, `location`, `contactEmail`, `integrationKey`.
- **Wallet:** `id`, `userId (FK)`, `balance`, `currency`.
- **Transaction:** `id`, `walletId (FK)`, `amount`, `type (Credit/Debit)`, `description`.
- **Inventory:** `id`, `productId (FK)`, `quantity`, `warehouseLocation`.
- **Review:** `id`, `userId (FK)`, `productId (FK)`, `rating`, `comment`.
- **Notification:** `id`, `userId (FK)`, `type`, `content`, `isRead`.
- **Payment:** `id`, `orderId (FK)`, `amount`, `status`, `gatewayRef`.

---

5. API Architecture & Security

5.1 Authentication Flow (JWT + Redis)

1. **Access Token:** Stored in memory (Frontend). Expiry: 15 minutes.
2. **Refresh Token:** Stored in `httpOnly`, `Secure`, `SameSite=Strict` Cookie. Expiry: 7 days.
3. **Whitelisting:** All active Refresh Tokens are stored in **Redis**. On logout, the token is deleted from Redis, effectively revoking access.

5.2 Security Protocols

- **Rate Limiting:** 100 requests per 15 minutes per IP for standard routes; 5 per minute for Auth routes.
- **Validation: Zod** is used for both Input (Request Body) and Output (Response) parsing to ensure no sensitive data leaks.

- **Error Handling:** Standardized JSON responses.

JSON

```
{
  "success": false,
  "error": "UNAUTHORIZED",
  "message": "Invalid or expired token",
  "details": null
}
```

---

6. Frontend Strategy

6.1 Route Groups & Protected Layouts

Using Next.js Route Groups to enforce layouts:

- `(marketing)` : Public-facing.
- `(auth)` : Login/Register only.
- `(dashboard)` : Requires `AuthGuard` .
  - `admin/` : Requires `RoleGuard(['ADMIN'])` .
  - `creator/` : Requires `RoleGuard(['CREATOR'])` .

6.2 Design System Implementation

Tailwind configuration will include:

JavaScript

```
theme: {
  extend: {
    colors: {
      printeastPink: '#E23E83',
      printeastOrange: '#FF7A39',
      neutralBase: '#F8F9FC',
    }
  }
}
```

---

7. Infrastructure & Deployment (CI/CD)

7.1 GitHub Actions Pipeline

1. **Lint & Format:** Ensures code style consistency.
2. **Type Check:** `tsc --noEmit` to catch logic errors.
3. **Build:** Verifies that both apps and packages build correctly.
4. **Deploy:** - Frontend → Vercel Preview/Production.
    - Backend → Render Staging/Production.

---

8. Phase 0 Deliverables Checklist

Week 1: Infrastructure & Data

- [ ] Initialize TurboRepo & `pnpm` workspaces.
- [ ] Setup Docker Compose for local Postgres & Redis.
- [ ] Define and migrate 15-table Prisma schema.
- [ ] Implement Express Auth module (Register/Login/JWT).
- [ ] Setup Supabase Storage Buckets.

Week 2: Shells & Connectivity

- [ ] Launch Parallax Landing Page (Static).
- [ ] Create Dashboard Layouts with Sidebar navigation for all roles.
- [ ] Implement Next.js Middleware for RBAC.
- [ ] Deploy full stack to Staging environments.
- [ ] Complete documentation for API endpoints (Swagger/OpenAPI).

Status: `READY FOR IMPLEMENTATION`