Technical Design Document 01: Core Architecture & Stack

Version: 1.0.0
Role: WebMMO Architect

1. Executive Summary

This document defines the non-negotiable technical foundation for the WebMMO project. It serves as the "Common Truth" for all subsequent modules (Game Design, Modding, Ethics), eliminating the need to restate stack decisions in other documents.

2. The Technology Stack (Strict)

2.1 Core Runtime

- **Language:** TypeScript 5.x+ (Strict Mode {"strict": true} is mandatory).
- **Build Tool:** Vite (Chosen for HMR speed and optimized bundling).
- **Package Manager:** Yarn (via Workspaces for Monorepo management).

2.2 Client-Side (The Renderer)

- **Engine:** Phaser 3.8+ (WebGL focus).
- **Constraint:** The Client is a "Dumb Terminal." It handles input and visualization. It **never** calculates authoritative game state.
- **Optimization:** * Use Float32Array for heavy data (fluid grids, pathfinding).
    - Use Phaser's AssetPack loader for dynamic content.

2.3 Server-Side (The Authority)

- **Runtime:** Node.js (Latest LTS).
- **Framework:** Colyseus.
- **Responsibility:** Authoritative simulation, state persistence, and validation.
- **State Sync:** Delta-compression via Colyseus Schema.

3. Directory Structure (Monorepo)

To ensure code sharing between Client and Server without duplication, we utilize a strict Monorepo layout.

```
/project-root
  /shared            <-- SHARED KERNEL
    /types           # TS Interfaces (e.g., IUnitStats, IBuildingConfig)
    /constants       # Game Balance (TILE_SIZE, MAX_HEALTH)
    /schemas         # Colyseus State Definitions (.ts)
```

```
   /utils          # Pure functions (Deterministic RNG, Math)

 /server           <-- AUTHORITATIVE NODE.JS
   /src
     /rooms        # Game Logic Loops
     /systems      # ECS Systems (Movement, Production)
     /sandbox      # isolated-vm implementation for Mods

 /client           <-- PHASER 3 + VITE
   /src
     /renderer     # Custom Shaders, Pipelines
     /scenes       # UI and Input handling
     /managers     # Prediction & Interpolation logic

 /mods             <-- GITIGNORED CONTENT
   /Core           # The "Base Game" is treated as Mod ID 0
   /UserMods       # Sideloaded content
```

4. Coding Standards

4.1 Data-Driven Design

Hardcoding values is strictly prohibited.

- **BAD:** const damage = 10;
- **GOOD:** const damage = ConfigManager.get('unit_stats', 'archer').damage;

4.2 Network Synchronization

We use Colyseus Schema for efficient delta updates.

```
// shared/schemas/Entity.ts
import { Schema, type } from "@colyseus/schema";

export class Entity extends Schema {
   @type("number") x: number;
   @type("number") y: number;
   @type("string") id: string;

   // We do NOT sync unnecessary data like internal timers or sprite colors
   // unless they change frequently. Static data stays in JSON.
}
```

5. Deployment & Scalability

- **Persistence:** MongoDB (for player data/save states).
- **Communication:** Redis (for inter-process communication between game rooms).
- **Sharding:** The world is divided into "Rooms" (Spatial Partitioning). A standard Gateway pattern routes players to the correct node.