This document compiles all the design decisions for the **Code Nexus** game into a single, comprehensive reference, structured by core systems: **Monetization, Language, Logic,** and **Implementation**.

1. Game & Monetization Model (Freemium with VIP)

Feature	Freelancer (Free-to-Play)	Architect (VIP Subscription)
Core Game Access	All features, commands, and knowledge are accessible. Progression is gated by time, efficiency, and manual effort.	All features are accessible. Progression is accelerated and simplified by automation tools.
Time Sink / Constraint	Lower **VC Resource Caps** (CPU/RAM). Slower **Hashcrack** and network operation times. **No Module Automation.**	**Quantum Core VC Upgrade** (Higher CPU/RAM). **Background Module Execution** (scripts run while offline).
VIP Exclusive Tools	N/A	**Automated Command Synthesis (A-CS)** (Simple interface UI). **Real-Time Module Debugger** (Step-by-step trace). **vc.auto_defend on** (Offline defense).
Monetization Goal	**Encourages Learning:** Forces players to write highly efficient, optimized code to overcome resource limits.	**Sells Efficiency:** Saves the player real-world time and effort on mundane or repetitive tasks.

2. NexusScript Language Design

The language is designed for mobile-friendly input, using spaces for separation and minimal special characters.

Core Syntax Rules

Concept	NexusScript Syntax	Example
Variable	Must start with **$**	set $target = new IP 10 0 0 1
Object Method	Dot notation with space-separated args	$router.connect("admin" $pass)
Control Flow	Uses simple brackets	if target.port is open (21) { run ftp login }

Command Database (Example Man Pages)

Command	Progression Tier	Syntax / Example Usage	Function
set	Core	set $var = [value]	Declare a variable or instantiate a new

Command	Progression Tier	Syntax / Example Usage	Function
			object.
scan	Core	scan $target_ip --service	Performs network discovery and port enumeration.
run	Core	run my brute force v1 ($server_ip)	Executes a player-created **NexusScript** module.
hashcrack	Level 5+	hashcrack $captured_hash $wordlist	Cracks a password hash using available resources.
pivot	Level 10+	pivot $compromised_server scan 10 10 0 0	Routes an attack through a compromised asset to reach an internal network.
thread spawn	Level 15+	set $t1 = thread spawn "module a"	Executes a module as a separate, parallel process (resource dependent).
raw	Level 15+	raw send $target_ip $data_packet	Allows sending custom, low-level data packets (advanced).

3. Game Logic and Progression

Progression is gated by a **"Hack-to-Learn"** system, where knowledge itself is the primary resource.

Knowledge Acquisition (K-Map System)

Stage	Logic Gate & Server Check	Player Action Required
LOCKED	Command fails with "not found" error.	N/A.
DISCOVERED	**Boolean Check: \text{KNOWLEDGE_FRAGMENTS_COLLECTED}**	Find and copy \ge 3 unique **Knowledge Fragments** from enemy/mission filesystems.
SYNTHESIS	**Boolean Check: \text{SYNTHESIS_MODULE_CREATED}**	Write a **NexusScript** module that correctly compiles the 3 fragments. The man page unlocks.
INTEGRATED (Functional)	**Boolean Check: \text{CONCEPT_VALIDATED}**	Successfully complete a high-level mission that requires the *concept* of the command (e.g., simultaneous actions for thread spawn). Command is fully executable.

Mission Logic (PvP & CTF)

Missions are defined by a server-side **Boolean Goal**.

Mission Type	Goal / Server Logic	Strategic Requirement
PvP: Resource Acquisition	\text{WINNER} = (\text{ACQUIRED_FILES} \ge 3)	Teams must hack opponents to steal 3 critical files OR defend/counter-attack to recover their own.
PvE/PvP: CTF	\text{WINNER} = (\text{TARGET_ROOT} \land \text{FLAG_EXFILTRATED})	First team to breach the final network layer (which is dynamically patched between rounds) and copy the flag wins.
Team Collaboration	\text{K-Map Update} = \text{SUCCESS}	Players can use the **share $fragment @team** command to instantly integrate knowledge fragments among teammates.

4. Implementation Framework (Backend & Client)

The architecture is designed for cross-platform mobile and web play, with all logic residing on the server.

Backend Server (Logic & State)

Component	Technology Recommendation	Primary Function
Framework	Node.js (Express) or Python (Django/Flask)	Hosts the core game logic and API endpoints.
Database	(MongoDB or PostgreSQL)	Stores all persistent data: Player state, K-Maps, Missions, and Module files.
APIs	**REST:** For state management (login, save, load). **WebSocket:** For real-time PvP, log streams, and live command execution feedback.	Ensures the single source of truth for all game logic is the server.
Execution Flow	Client \rightarrow WebSocket \rightarrow **Server-Side VC Logic** \rightarrow Server Response \rightarrow Client.	The server executes all NexusScript and game logic; the client is a display terminal.

Cross-Platform Client (Interface)

Component	Technology Recommendation	Primary Function
Game Engine	**Phaser 3** (or similar framework like PixiJS)	Creates a single, responsive interface that works on web and mobile (via wrappers like Capacitor).
Interface	**Simple CLI/Terminal UI:**	Provides the visual experience,

Component	Technology Recommendation	Primary Function
	Fixed-size terminal area, simple input component optimized for mobile keyboards, and a static sidebar for VC status (RAM/CPU/Credits).	focusing on text and minimalist design for low bandwidth and high readability.