Quantum Mechanic: Modular Implementation Framework

Session-Persistent Development System

🎯 Overview

This document provides a **session-resumable implementation strategy** for building all 10 major feature systems. Each module is designed as a **self-contained artifact** that can be built independently across multiple sessions without context loss.

📋 Module Completion Tracker

✅ COMPLETED MODULES (Session 1)

- [x] **Core-01**: NetworkIdentity.cs
- [x] **Core-02**: PacketProcessor.cs
- [x] **Core-03**: ServerHost.cs
- [x] **Core-04**: ClientManager.cs
- [x] **Core-05**: SaveSystem.cs
- [x] **Core-06**: EconomyManager.cs
- [x] **Core-07**: ProjectBootstrapper.cs

🔄 IN-PROGRESS MODULES

- [ ] None currently

⏳ PENDING MODULES (Prioritized Queue)

Phase 1: Visual Foundation (Critical Path)

- [ ] **Visual-01**: ProceduralModelFactory.cs
- [ ] **Visual-02**: MeshGenerationLibrary.cs (Humanoid, Weapon, Armor primitives)
- [ ] **Visual-03**: MaterialGeneratorURP.cs (Cyberpunk, Fantasy, Organic styles)

Phase 2: Character Identity

- [ ] **Character-01**: CharacterCreationSystem.cs
- [ ] **Character-02**: RaceDefinitions.cs (8 races with stats/abilities)
- [ ] **Character-03**: ClassDefinitions.cs (6 classes with skill trees)
- [ ] **Character-04**: StatAllocationSystem.cs (27-point buy)
- [ ] **Character-05**: BackgroundSystem.cs

Phase 3: Combat Foundation

- [ ] **Combat-01**: WeaponDatabase.cs (100+ weapon definitions)
- [ ] **Combat-02**: WeaponStatCalculator.cs
- [ ] **Combat-03**: DamageSystem.cs (types, resistances, crits)
- [ ] **Combat-04**: CombatController.cs (attack resolution)

Phase 4: Ranged & Physics

- [ ] **Physics-01**: ProjectilePhysicsSystem.cs
- [ ] **Physics-02**: ProjectileFactory.cs (bullet, arrow, plasma, psi)
- [ ] **Physics-03**: HitDetectionSystem.cs (headshots, limb damage)
- [ ] **Physics-04**: BallisticsCalculator.cs (gravity, drag, penetration)

Phase 5: Magic System

- [ ] **Magic-01**: MagicSystem.cs (mana management)
- [ ] **Magic-02**: SpellDatabase.cs (50+ spells across 6 schools)
- [ ] **Magic-03**: SpellCastingController.cs
- [ ] **Magic-04**: SpellEffectManager.cs (VFX sync)
- [ ] **Magic-05**: SpellComboSystem.cs

Phase 6: Psionic System

- [ ] **Psionic-01**: PsionicSystem.cs (Psi Points)
- [ ] **Psionic-02**: PsionicPowerDatabase.cs (30+ powers)
- [ ] **Psionic-03**: TelekinesisController.cs
- [ ] **Psionic-04**: TelepathyController.cs
- [ ] **Psionic-05**: AstralProjectionController.cs

Phase 7: Augmentation System

- [ ] **Aug-01**: AugmentationManager.cs
- [ ] **Aug-02**: AugmentationDatabase.cs (20+ augmentations)
- [ ] **Aug-03**: BioelectricEnergySystem.cs
- [ ] **Aug-04**: AugmentationSlotController.cs
- [ ] **Aug-05**: AugmentationVFXSystem.cs

Phase 8: Auction House

- [ ] **Economy-01**: AuctionHouseSystem.cs
- [ ] **Economy-02**: AuctionListingManager.cs
- [ ] **Economy-03**: BiddingController.cs
- [ ] **Economy-04**: MailSystem.cs
- [ ] **Economy-05**: AuctionHouseUI.cs

Phase 9: Dungeon Generation

- [ ] **Dungeon-01**: DungeonGenerator.cs (BSP algorithm)
- [ ] **Dungeon-02**: RoomFactory.cs (combat, puzzle, treasure, boss)
- [ ] **Dungeon-03**: EnemySpawner.cs
- [ ] **Dungeon-04**: LootDistributor.cs
- [ ] **Dungeon-05**: DungeonNetworkSync.cs

Phase 10: Integration & UI

- [ ] **Integration-01**: GameNetworkManager.cs (v2 with all systems)
- [ ] **Integration-02**: PlayerController.cs (v2 with abilities)
- [ ] **UI-01**: CharacterCreationUI.cs
- [ ] **UI-02**: InventoryUI.cs
- [ ] **UI-03**: SpellbookUI.cs
- [ ] **UI-04**: AuctionHouseUI.cs
- [ ] **UI-05**: AugmentationUI.cs

🔄 Session Resume Protocol

Starting a New Session

Copy-paste this prompt to resume work:

QUANTUM MECHANIC RESUME - SESSION [NUMBER]

I'm continuing implementation of the Quantum Mechanic Mini-MORPG.

COMPLETED MODULES:
[List all checked modules from tracker above]

CURRENT OBJECTIVE:
Implement the next module in the priority queue: [MODULE-ID]

REQUIREMENTS:
1. Full, production-ready C# code (no snippets)
2. Integrate with existing NetworkIdentity, PacketProcessor, SaveSystem, EconomyManager
3. Follow "Call-Chain Rule" - all methods must be invoked in game loop
4. Use ONLY Unity built-in APIs + System.* namespaces
5. Include XML documentation for agentic readability
6. Create as artifact for easy copy-paste

CONTEXT FROM PREVIOUS SESSION:
[Paste relevant technical details from notes below]

Please provide the complete implementation for [MODULE-ID].

📝 Technical Context Notes (Session-Persistent)

Architecture Decisions

Networking:

- TCP with 4-byte size-prefixed framing
- JsonUtility for serialization (no Newtonsoft)

- Multi-threaded listeners with main-thread action queues
- Packet types: enum PacketType (bytes 1-15 used, 16+ available)

Save System:

- AES-256-CBC encryption with fixed keys (production: use per-user keys)
- Atomic writes via temp file + rename
- Auto-save every 60 seconds
- Path: Application.persistentDataPath

Economy:

- Event-driven (OnCurrencyChanged, OnItemAdded, etc.)
- Stack-based inventory (configurable max stack sizes)
- Item database: Dictionary<string, Item>
- Currency stored in PlayerData

Materials:

- URP Lit shader (fallback to Standard if URP missing)
- Programmatic generation in ProjectBootstrapper
- Naming: PlayerMaterial, GroundMaterial, etc.

Prefabs:

- Player: Capsule + NetworkIdentity + CharacterController + PlayerController
- Procedurally generated via ProjectBootstrapper
- Saved to: Assets/_QuantumMechanic/Prefabs/

Folder Structure:

```
Assets/_QuantumMechanic/
  ├── Scripts/
  │   ├── Core/
  │   ├── Networking/
  │   ├── Economy/
  │   ├── Persistence/
  │   ├── Player/
  │   ├── Combat/      [NEW]
  │   ├── Magic/       [NEW]
  │   ├── Psionics/    [NEW]
  │   └── Augmentation/[NEW]
  ├── Prefabs/
  ├── Materials/
  └── Scenes/
```

Naming Conventions

Events:

- Prefix: On[Action][Result]
- Example: OnCurrencyChanged, OnItemAdded

Managers:

- Suffix: Manager, System, Controller
- Singleton pattern via static Instance property

Network Packets:

- Suffix: Data

- Example: TransformData, SpellCastData

Enums:

- PascalCase, descriptive
- Example: PacketType, WeaponType, DamageType

Integration Points

PlayerData Extensions Needed:

```csharp
// Add to SaveSystem.PlayerData
public string characterRace;
public string characterClass;
public int[] baseStats; // STR, DEX, CON, INT, WIS, CHA (6 values)
public string[] installedAugmentations;
public string[] learnedSpells;
public string[] psionicPowers;
public string equippedWeaponPrimary;
public string equippedWeaponSecondary;
public string[] equippedArmor; // Head, Chest, Legs, Feet
```

New Packet Types Needed:

```csharp
```

```
public enum PacketType : byte
{
    // Existing: 1-9
    Augmentation = 10,
    SpellCast = 11,
    ProjectileSpawn = 12,
    AuctionBid = 13,
    AuctionListing = 14,
    PsionicEffect = 15,
    DungeonSeed = 16,
    WeaponEquip = 17,
    DamageDealt = 18,
    CharacterCreated = 19
}
```

NetworkIdentity Extensions:

csharp

```
// Add to NetworkIdentity.cs
private int _currentHealth;
private int _maxHealth;
private float _currentMana;
private float _maxMana;
private float _currentPsi;
private float _maxPsi;

public event Action<int, int> OnHealthChanged; // current, max
public event Action<float, float> OnManaChanged;
public event Action<float, float> OnPsiChanged;
```

🎯 Module Implementation Templates

Template A: Database System

Use for: WeaponDatabase, SpellDatabase, AugmentationDatabase, RaceDefinitions, ClassDefinitions

```csharp
```

```
using System;
using System.Collections.Generic;
using UnityEngine;

namespace QuantumMechanic.[System]
{
    [Serializable]
    public class [Item]Definition
    {
        public string id;
        public string displayName;
        // ... specific fields
    }

    public class [System]Database : MonoBehaviour
    {
        private static [System]Database _instance;
        private Dictionary<string, [Item]Definition> _database;

        public static [System]Database Instance => _instance;

        private void Awake()
        {
            if (_instance != null && _instance != this)
            {
                Destroy(gameObject);
                return;
            }
            _instance = this;
            DontDestroyOnLoad(gameObject);
            InitializeDatabase();
        }
```

```
    private void InitializeDatabase()
    {
        _database = new Dictionary<string, [Item]Definition>();
        RegisterDefault[Items]();
    }

    private void RegisterDefault[Items]()
    {
        // Register 10+ items inline
    }

    public [Item]Definition Get[Item](string id)
    {
        return _database.TryGetValue(id, out var item) ? item : null;
    }

    public void Register[Item]([Item]Definition item)
    {
        _database[item.id] = item;
    }
  }
}
```

Template B: Manager System

Use for: MagicSystem, PsionicSystem, AugmentationManager, DungeonGenerator

```
csharp
```

```
using System;
using UnityEngine;

namespace QuantumMechanic.[System]
{
    public class [System]Manager : MonoBehaviour
    {
        [Header("Configuration")]
        [SerializeField] private float _resourceMax = 100f;

        private static [System]Manager _instance;
        private float _currentResource;

        public static [System]Manager Instance => _instance;
        public float CurrentResource => _currentResource;

        // Events
        public event Action<float, float> OnResourceChanged; // current, max
        public event Action<string> On[Action]Performed;

        private void Awake()
        {
            if (_instance != null && _instance != this)
            {
                Destroy(gameObject);
                return;
            }
            _instance = this;
            DontDestroyOnLoad(gameObject);
        }

        private void Update()
        {
```

```
        // Regeneration, cooldowns, etc.
    }

    public bool TryPerform[Action](string actionId, float cost)
    {
        if (_currentResource < cost) return false;

        _currentResource -= cost;
        OnResourceChanged?.Invoke(_currentResource, _resourceMax);
        On[Action]Performed?.Invoke(actionId);
        return true;
    }
  }
}
```

Template C: Network Controller

Use for: SpellCastingController, ProjectileFactory, AuctionListingManager

```
csharp
```

```
using UnityEngine;
using QuantumMechanic.Networking;

namespace QuantumMechanic.[System]
{
    public class [System]Controller : MonoBehaviour
    {
        private ClientManager _client;
        private ServerHost _server;

        private void Start()
        {
            _client = FindObjectOfType<ClientManager>();
            _server = FindObjectOfType<ServerHost>();

            if (_client != null)
            {
                _client.OnPacketReceived += HandleClientPacket;
            }

            if (_server != null)
            {
                _server.OnPacketReceived += HandleServerPacket;
            }
        }

        private void HandleClientPacket(NetworkPacket packet)
        {
            if ((PacketType)packet.packetType != PacketType.[YourType])
                return;

            // Process packet
        }
```

```
    private void HandleServerPacket(uint clientId, NetworkPacket packet)
    {
        if ((PacketType)packet.packetType != PacketType.[YourType])
            return;

        // Validate and broadcast
    }

    public void Send[Action]Request(/* params */)
    {
        [Data] data = new [Data](/* params */);
        string payload = JsonUtility.ToJson(data);
        NetworkPacket packet = new NetworkPacket(PacketType.[YourType], _client.LocalClientId,
        _client.Send(packet);
    }
  }
}
```

📦 Artifact Delivery Format

Each module should be delivered as a **single, complete artifact** with:

1. **File header comment** with module ID and dependencies
2. **Full namespace and imports**
3. **Complete class implementation** (no truncation)
4. **XML documentation** on all public methods
5. **Integration hooks** for existing systems

Example Header:

```csharp
// MODULE: Visual-01
// FILE: ProceduralModelFactory.cs
// DEPENDENCIES: None (pure Unity + System.*)
// INTEGRATES WITH: All systems (generates visual models)
// NETWORK PACKETS: None (client-side only)
```

🔢 Progress Tracking

Completion Metrics

- **Total Modules**: 55
- **Completed**: 7 (12.7%)
- **Remaining**: 48 (87.3%)

Estimated Sessions

- **Modules per session**: 3-5 (depending on complexity)
- **Total sessions needed**: 10-16
- **Current session**: 1

Critical Path

The fastest path to playable game:

1. Visual-01, Visual-02 (model generation)

2. Character-01, Character-02, Character-03 (creation system)
3. Combat-01, Combat-02, Combat-03 (weapons + damage)
4. Integration-01, Integration-02 (hook everything together)
5. UI-01, UI-02 (player-facing interfaces)

Playable Milestone: After ~15 modules (27% completion)

💾 Context Preservation Strategy

What to Save Between Sessions

Per Module Completed:

- [x] Check the box in completion tracker
- [ ] Add any new packet types to "New Packet Types Needed" section
- [ ] Note any PlayerData fields added
- [ ] Update "Integration Points" if manager events added
- [ ] Record folder structure changes

After Each Session:

- Copy updated "Technical Context Notes" to external file
- Save completion tracker state
- Note any architectural decisions made

What NOT to Include in Resume Prompt

- Full code from previous modules (bloats context)
- Detailed roadmap (already established)

- UI mockups or visual designs
- Lore/narrative content

🚀 Quick Start Next Session

Session 2 Recommendation: Start with **Visual-01** (ProceduralModelFactory.cs) because:

1. It's the foundation for all visual content
2. Requires no dependencies beyond Unity primitives
3. Can be tested immediately (generate a few models in editor)
4. Unblocks character creation, weapons, augmentations

Estimated Scope: 300-500 lines, single artifact

🎯 Success Criteria Per Module

Each module must:

- [ ] Compile without errors
- [ ] Follow "Call-Chain Rule" (all methods invoked)
- [ ] Use only permitted dependencies
- [ ] Include 3+ XML doc comments
- [ ] Integrate with at least 1 existing system
- [ ] Be copy-pasteable as-is into Unity project

📋 Session Checklist Template

Copy this for each session:

```
SESSION [NUMBER] - [DATE]

MODULES TARGETED:
- [ ] [MODULE-ID-1]
- [ ] [MODULE-ID-2]
- [ ] [MODULE-ID-3]

ARTIFACTS CREATED:
- [ ] [Filename].cs - [Lines of code] - [Status: Complete/Partial]

INTEGRATION POINTS ADDED:
- [ ] New packet types: [list]
- [ ] PlayerData fields: [list]
- [ ] Events subscribed: [list]

BLOCKERS/ISSUES:
- [None / List issues]

NEXT SESSION START:
Begin with module: [MODULE-ID]
Context needed: [Specific technical details]
```

🔧 Bootstrapper Integration Queue

As modules complete, add to ProjectBootstrapper:

```
csharp
```

```
[MenuItem("Project/Initialize Phase 2 - Characters")]
public static void InitializePhase2()
{
    // Create character creation UI prefab
    // Generate race/class definition assets
    // Setup stat allocation scene
}

[MenuItem("Project/Initialize Phase 3 - Combat")]
public static void InitializePhase3()
{
    // Generate weapon database JSON
    // Create projectile prefabs
    // Setup combat test scene
}
```

Each phase adds new menu items for incremental setup.

This framework ensures every session is productive, resumable, and measurably advances toward the complete implementation of all 10 feature systems without context loss or token waste.