ChatGPT

PurpurMC 1.21.5 API – Capabilities vs Bukkit/Spigot/Paper for an MMO Engine

Overview: Bukkit, Spigot, Paper, and Purpur APIs

Minecraft plugin development builds on the Bukkit API, with Spigot and Paper extending it. **Bukkit** provides the core event system, commands, and basic access to game entities and blocks. **Spigot** is a performance-tuned fork of Bukkit that adds a few convenience APIs (e.g. JSON chat components, item cooldowns) and configuration tweaks. **Paper** is a further fork focused on performance and more API hooks – it includes all Spigot APIs and introduces many new events, async operations, and utilities useful to developers. **Purpur** is a fork of Paper aimed at maximum configurability and new gameplay features [1]. Purpur includes **all** Paper API features and adds its own enhancements (custom events, config options, and Pufferfish performance patches) on top [1]. For a large MMO-style plugin, Purpur's API can be seen as a superset of Bukkit/Spigot/Paper, giving you the broadest range of features and knobs to tweak.

Key Differences: Bukkit/Spigot provide the basics needed for plugins, but complex MMORPG features often require workarounds or direct NMS access. Paper's expanded API (and improvements up through Minecraft 1.21.5) offers cleaner hooks for combat, block interactions, scheduling, etc., reducing the need for hacky solutions. Purpur further adds convenience by exposing server-side toggles (in `purpur.yml`) and a handful of extra events (e.g. for anvil usage, spawners, ridable entities) that can simplify MMO plugin development. Below, we detail each aspect and how the platforms compare.

Custom Combat Systems

Designing custom combat mechanics (custom hit logic, damage calculation, attack speed, etc.) is possible on all platforms, but with varying ease:

- **Hit Registration & Damage Events:** All Bukkit-based servers use the `EntityDamageByEntityEvent` (and `EntityDamageEvent`) as the primary hook for melee and projectile hits. On **Bukkit/Spigot**, you can listen to these events to detect hits and modify damage. You can cancel hits, apply your own damage values (`event#setDamage()`), or add effects. Spigot doesn't add much beyond Bukkit here, aside from more damage cause types and minor API improvements. **Paper** and Purpur, however, provide extra context and control. For example, Paper exposes an `EntityPushedByEntityAttackEvent` that fires when an entity is knocked back by an attack [2] – this event allows an MMO plugin to adjust or cancel the knockback effect separately from the damage. Paper also provides `EntityDamageItemEvent` (for when an item, like armor, takes durability damage) [3], which helps in implementing custom item durability or "armor break" mechanics. Purpur inherits all these Paper events, and additionally offers **Pre-explosion events** that can tie into combat: e.g. `PreEntityExplodeEvent` fires before an entity's explosion is processed [4], letting you modify or cancel explosion damage (useful for custom explosive skills or preventing

terrain damage from boss attacks). In general, **Paper/Purpur give finer-grained combat hooks** than Spigot/Bukkit.

- **Custom Damage Calculations:** In Bukkit, you often calculate "true damage" (ignoring armor/enchantments) by setting damage after the default armor reduction, or by using a custom damage cause. Spigot and Paper still use the vanilla damage pipeline by default, but **Paper 1.19+ introduced a data-driven damage system** aligned with Minecraft's new `DamageType` registry. This means plugins (or datapacks) can define new damage types with specific properties (e.g. ignoring armor or immunity) [5]. Paper's registry events allow registering custom `DamageType` entries in `minecraft:damage_type` [6], so an MMO plugin could, for example, define a "TRUE_DAMAGE" type that bypasses armor and use it when inflicting damage. Without that, one can still simulate true damage by applying damage with a type that ignores armor (such as the void or magic damage types) or by manually calculating final values and using `Entity#setHealth`. **Status effects** (poison, stun, etc.) beyond vanilla potions must be handled by the plugin on all platforms. You might simulate them by repeatedly applying potion effects (Bukkit API `addPotionEffect`) or by tracking custom metadata (e.g. a "stun" tag that your plugin checks in movement events). No platform directly supports adding new potion effect types without a client mod, but Paper's registry system also allows registering custom potion or enchantment types via datapacks if needed. In summary, **all platforms allow overriding damage in events**, but Paper/Purpur (1.21.5) make it cleaner: you get more event hooks and can even inject new damage mechanics via the data/registry API.

- **Attack Speed & Cooldowns:** Minecraft's attack cooldown (post-1.9 "Combat Update") is governed by the player's **attack speed attribute** and client-side timing. **Bukkit/Spigot** let you modify attack speed using the Attributes API (`Attribute.GENERIC_ATTACK_SPEED`) on players or weapons, so you can raise it to remove cooldown or lower it for slower attacks. They do not provide a direct toggle to revert to pre-1.9 combat, so plugins like *OldCombatMechanics* are used to achieve that. Purpur doesn't natively "bring back old hit spam" either (it focuses on adding options without reintroducing removed mechanics [1]), but you could configure a high default attack speed and use plugins to fine-tune old behavior. For **custom attack cooldowns or ability cooldowns**, Spigot introduced an **item cooldown API**: you can call `Player#setCooldown(Material, ticks)` to put a global cooldown on using a specific item type (this shows the little progress overlay on the hotbar icon, just like ender pearl cooldown) [7]. This is great for MMO ability items – e.g., after using a "teleportation orb" item, set a cooldown so the player can't use that item again for N seconds. Paper/Purpur fully support this Spigot API. Additionally, Paper's item component system (see *Item Customization* below) includes a `UseCooldown` data component that can define cooldowns per item more granularly (e.g. attach a cooldown property to a specific item stack rather than all of a material). This is an advanced feature – by default, using Spigot's `setCooldown` per Material is simpler.

- **Custom Effects & Criticals:** Bukkit doesn't expose the logic for critical hits or attack combos – you'd implement custom crits by checking conditions in damage events (e.g. if player is airborne and attacking, multiply damage). Paper doesn't add explicit API for critical hits either (that's mostly client-side), but with its data API you could potentially mark an item or entity to adjust damage outcomes. For visual effects, all platforms let you spawn particles (`World#spawnParticle`) and play sounds on hit, which is important for satisfying combat feedback in an MMO. Paper's chat/Adventure API integration can also send action bar messages on hits (for example, "**+10 Combo!**"), though Spigot can do that via `player.spigot().sendMessage(ChatMessageType.ACTION_BAR, …)` as well.

Purpur doesn't add specific combat effects APIs, but its **configuration** can adjust some mechanics server-wide (for instance, enabling/disabling sweep attacks or tweaking knockback). Purpur's default config is very configurable for PvE/PvP mechanics (e.g. disabling creeper griefing without a plugin, configuring ender pearl cooldown, etc.), which can indirectly simplify an MMO server setup.

Summary: For custom combat, **Bukkit/Spigot require more manual work**, intercepting damage events and using attributes. **Paper** provides extra events (e.g. to handle knockback separately, or monitor item damage) and allows hooking into the new **DamageType** system for finer control [5]. **Purpur** inherits those and adds a few events of its own (e.g. pre-explosion) and config toggles. If your MMO plugin needs complex combat (like custom damage types, armor penetration, or alternate attack mechanics), using Paper/Purpur gives you cleaner APIs and better performance. You can achieve the same on Spigot, but expect to do more low-level tweaking (or even NMS for things like disabling shield blocking or modifying reach).

Mining Systems (Block Interaction & Gathering)

Mining and resource gathering in an MMO context often involves custom block behaviors – e.g. variable mining speeds, regenerating ores, custom drop logic, etc. The API support for these has improved significantly in Paper 1.21.x versus base Spigot:

- **Block Break Events & Custom Drop Logic:** On all platforms, `BlockBreakEvent` is the core event fired when a player breaks a block. Bukkit/Spigot allow you to cancel the break or modify the drops and XP. For example, you can listen for a diamond ore break and drop custom items or prevent the break unless conditions are met. This works, but to implement advanced mining (like progressively breaking a rock or cooperative breaking), Bukkit alone was limiting. **Paper** introduces a specialized event `BlockBreakProgressUpdateEvent`, which fires **whenever the client's block crack progress is updated** during mining [8]. This is a game-changer for custom mining speed mechanics – it lets you detect how far along a block is in being broken. You could use it to *slow down* or *speed up* mining by canceling the break at certain progress and resetting progress, or to allow multiple players to contribute to breaking the same block (since you can track partial progress). For instance, an MMO plugin can listen to `BlockBreakProgressUpdateEvent` and accumulate a custom "mining progress" based on a player's Mining Speed stat, only actually breaking the block when your threshold is met. On Bukkit/Spigot, there is no such event; you'd have to cancel every break immediately and use a repeating task to simulate breaking (sending fake crack particles). Paper's event (added in recent versions) and **Purpur** (which includes it) make that much cleaner [8].

- **Mining Speed Manipulation:** By default, mining speed is determined by tool material, enchantments (Efficiency), and potion effects (Haste). **Bukkit/Spigot** do not let you directly set how fast a block breaks, but you can simulate faster mining by giving players effects or by handling instant breaks manually. For example, an old trick is to monitor `PlayerInteractEvent` for left-clicks and use `player.sendBlockDamage()` (Spigot API) to show crack progress manually [9] [10]. Spigot's `Player#sendBlockDamage` allows you to fake a breaking animation with a progress value (0.0 to 1.0) for a given player [9]. An MMO plugin could use that to visualize a custom mining timer. However, doing this well is complex – you must cancel the actual break and track time yourself. **Paper's approach with block break progress events** plus **asynchronous chunk access** improvements makes it easier. Additionally, **Paper 1.21.5 introduces an experimental Item Data API** that can define tool behavior. Using Paper's `Tool` component, you can attach custom mining rules to an ItemStack. For example, you can declare that your custom pickaxe item instant-breaks

Netherrack but mines **Obsidian** at half speed. The `Tool.Rule` allows specifying a set of block types with a custom speed multiplier [11], and whether the tool can harvest drops from those blocks (simulate correct tool tier) [12]. This means on Paper you could create *new tool tiers* via code – something not possible on Spigot without NMS. Purpur inherits this capability. Under the hood, Paper's system will handle the mining speed and drop logic according to your rules (no need to intercept every event). *Example:* Define a "Mithril Pickaxe" item in your plugin and give it a `Tool` data component that sets `"speed": 2.0` for stone-type blocks (mines twice as fast) and `"speed": 0.5` for ores (half speed), etc. When players use that item, the server adjusts the breaking time accordingly. This level of customization **did not exist in Bukkit/Spigot** – it's brand new in Paper's API (1.19+ and improved by 1.21.5) [13] [11].

- **Regenerating Blocks (Mining Regen):** Many MMO servers have mining areas where blocks regenerate after a delay (like Hypixel SkyBlock's regenerating ores). Bukkit/Spigot provide no built-in support for this; you implement it by storing the broken block's type and scheduling a task (using the Bukkit scheduler) to re-place the block after X seconds. This is straightforward: on `BlockBreakEvent`, if the block is in a special region, cancel drops and schedule `block.setType(originalType)` later. All platforms can do this. The difference comes with **performance and persistence**. With a high player count, scheduling thousands of block regen tasks on Bukkit could cause lag, but using **Paper's async scheduler** (or Folia's region scheduler) can help distribute that work. Paper's Folia (multi-threaded region) technology, which Purpur can support, provides **region-specific schedulers** so that resetting a block in one region won't stall the whole server tick [14]. In practice, even on Spigot, a well-tuned scheduler can handle regen tasks, but Paper gives you the option to offload block placement to a region thread or run it asynchronously if it doesn't need to be exact. Also, Paper's Block API tends to be more thread-safe in certain contexts (with the new RegionScheduler, you can schedule `location.getBlock().setType(...)` on the correct thread easily [15] [16]).

- **Custom Tool Stats (Fortune, etc.):** Changing how tools affect drops (like a custom "double ores" stat or custom enchant) is mostly done via event logic on all platforms. For example, you can catch `BlockBreakEvent`, check the player's custom stat for Mining Fortune, and modify the drops list (Bukkit's event lets you `event#getDrops()`). Spigot/Paper don't have a specific API for "Fortune X2" beyond the vanilla enchant handling, so you'll implement that manually. However, Paper's platform does make it easier to read and write **NBT data** on items, so you could store a custom NBT tag like `MiningFortune:50` on a tool using the **PersistentDataContainer** and have your event logic read that. PersistentDataContainer (PDC) works on **ItemMeta** in Spigot and above, so this is available to all (it's part of the Bukkit API since 1.14). Purpur doesn't change this directly, but one of Purpur's goals is to reduce the need for separate plugins – for example, Purpur has a built-in config option to allow **silk-touch harvesting of spawners** (with customizable tool list and lore) [17] [18], which normally you'd implement via an event in Spigot. Such config options aren't *APIs* for the plugin, but they show Purpur focusing on common custom mining tweaks (like letting players pick up spawners without writing a plugin). In an MMO plugin on Purpur, you could choose to rely on that config for spawner drops (and just require the `purpur.drop.spawners` permission for players) rather than coding it yourself.

Summary: Bukkit/Spigot give the basic hooks for mining (break events, manual crack animations). **Paper** greatly enhances block interaction control with *BlockBreakProgressUpdateEvent* [8] and the new **Tool data components**, letting you create truly custom mining tools and multi-stage breaking without fragile

workarounds. **Purpur** inherits those and adds convenience configs (e.g. easily toggle features like spawner mining [17] or define blocks that can be path-created with a shovel). For a complex mining system (like Hypixel's where mining speed is a player stat, and blocks take variable time to break and then regenerate), Paper/Purpur's capabilities **unlock much cleaner implementations**. Previously difficult tasks – e.g. showing progressive breaking for a 5-second ore – can now be handled by listening to break progress events or by configuring a custom tool that inherently has a slower breaking speed for that ore. This reduces the need for kludgy client-side packet hacks and yields a more robust mining system.

Structure Generation and World Generation Support

Large MMO servers often have custom worlds or dynamic structure generation (for dungeons, housing, etc.). The APIs for world generation differ in flexibility:

- **Bukkit/Spigot World Generation:** Bukkit provides a way to implement a custom **ChunkGenerator** (via `WorldCreator` and your own generator class) and **BlockPopulator** for post-processing chunks. This is how you create completely custom worlds (terrain) in pure Java. However, the Bukkit API for world gen is fairly low-level and static – many servers instead used third-party worldgen plugins or mods (like TerrainControl, now OpenTerrainGenerator) because fine-tuning terrain via Bukkit alone was limited. Spigot didn't add much to worldgen APIs besides some bugfixes; you're still basically writing noise functions or using the vanilla generator and modifying it. For **structure placement**, Bukkit has an API to load and paste **structure templates** (structures saved by a structure block as `.nbt`). You can use `StructureManager` and `Structure` API to load saved schematics and paste them at runtime. This works on Spigot and above, which is useful for scripted generation of buildings or dungeons. Without mods, **all platforms require you to pre-define structures or code the generation algorithm manually**.

- **Paper/Purpur Enhancements:** Paper has focused on performance and *some* worldgen extensibility. In 1.18+, Paper introduced asynchronous chunk generation improvements (to lessen lag spikes when generating new terrain). By 1.21.5, Paper has integrated experimental changes from their *"generators"* branch [19], meaning the server can handle chunk generation on multiple threads or without blocking the main thread as much. For a plugin developer, this means that if you trigger world generation (e.g. using `World#loadChunk` or flying around to pregenerate), Paper will handle it more smoothly – you don't get new API calls, but your actions cause less lag. **Custom Structures via Datapack:** A cutting-edge approach is to leverage Minecraft's built-in datapack system. Minecraft allows custom biomes, carvers, structures, etc., via datapacks (JSON files). Paper's API exposes hooks to datapack loading [20] and registry events [21]. For example, Paper's `RegistryEvent` can let a plugin modify the server's tag or registry entries before the world loads. An MMO plugin could inject a datapack (via the API or file) that defines new structures (like custom buildings that spawn in certain biomes) and use Paper's registry events to ensure it's loaded. This is fairly advanced and not strictly needed if you generate structures manually, but it's an option. **Purpur** doesn't add new worldgen APIs beyond Paper, but it does have extra toggles: e.g. controlling whether spawners can be deactivated by redstone [22], whether Nether bedrock ceiling is enabled (not sure if Purpur has that toggle, but it has many world options), or enabling certain discontinued world features. These configurations can indirectly support world design for MMO (for instance, if you want skyblock-like void worlds, Purpur can ensure no phantom chunks spawn water animals incorrectly [23] via a fix toggle).

- **Procedural/Scripted Generation:** If your MMO needs to generate structures on the fly (say, a dungeon when a player enters, or player housing plots), you will mostly use a combination of **WorldEdit's API or manual block placement**. None of these platforms provide a high-level "generate castle" function – you'd either pre-design schematics and paste them (with WorldEdit API or Bukkit's structure API), or algorithmically place blocks with loops. The difference is performance and thread safety. On Bukkit/Spigot, **any block changes must occur on the main server thread** (or you risk concurrency issues). Paper's Folia (region thread) model would allow you to schedule block changes on a region thread asynchronously [15] [16], which is beneficial if you're generating a large structure while players are around. Also, Paper's asynchronous chunk loading means you can safely check or load chunks without halting the tick loop, which helps if your generation algorithm needs to scan an area. Purpur, being based on Paper, inherits these advantages.

- **New MC Features:** By 1.20/1.21, Mojang has added commands like `/place structure <name>` and the concept of **template worlds** – while not directly an API, an MMO plugin could invoke these commands (via the API `dispatchCommand`) to place vanilla structures or use the **StructureTemplate** API to handle .NBT templates. All platforms can do that, but Paper's team often ensures their API is up-to-date with new vanilla features. For example, if 1.21 introduced a new type of structure element, Paper's API likely provides a wrapper for it sooner.

Comparing Difficulty: Bukkit/Spigot can accomplish custom world generation, but it often means heavy custom code or relying on external libraries. **Paper** doesn't magically generate worlds for you, but it *improves performance and adds hooks* – e.g. you can listen on `PaperWorldGenerationInitEvent` (hypothetical name) or use async chunk calls to generate without lag. In **Purpur**, while no significant new worldgen API is added, you benefit from all Paper improvements plus granular configs for world behavior (which can save you from writing small utility plugins). In summary, for generating and managing an MMO world with custom structures, Paper/Purpur give you a smoother experience and more integration points (including the ability to integrate custom datapack content). The fundamental work of designing the generation logic remains the same as Spigot, but you have more tools to avoid performance pitfalls and to integrate with Minecraft's native structure systems.

Custom Stats and Player Attributes

MMO servers typically introduce a host of custom stats (e.g. **Strength, Agility, Defense, Crit Chance, "True Defense"**, etc.) and often need to modify player attributes like health or speed. Implementing these involves both **storing/tracking the stats** and **applying their effects in-game**. Here's how the platforms compare:

- **Storing Custom Stats:** All platforms allow you to maintain custom data for players. The traditional approach is using a HashMap or database in your plugin to keep track of stats. Modern APIs offer **Persistent Data Containers** (PDC) for attaching custom data to entities. In Spigot (and thus Paper/Purpur), `Player` **implements** `PersistentDataHolder` [24], so you can store, say, an integer for "Strength" on the player's NBT which persists across server restarts (saved in the player `.dat` file). This is a powerful built-in mechanism to persist stats without external files – all three (Spigot/Paper/Purpur) support it. Bukkit (pre-1.14) didn't have PDC, but by 1.21.5 any server you use will have it available. **Scoreboards** are another way to store/display stats: you can create a scoreboard objective for a stat and update it as needed, which also persists and can show on the sidebar or below names. Bukkit allowed this, and Spigot/Paper continue to support it. **Paper** even adds *number formatting* for

scoreboards – e.g., using Paper's `FixedFormat` or `StyledFormat`, you can display a score as a formatted text (like adding thousands separators or custom color) instead of a plain integer [25]. This could be used to show HP or Mana in a fancy way on the scoreboard. In general, storing the raw values is similar across platforms (PDC, config files, or a database via JDBC), but **Paper/Purpur give you nicer ways to integrate with Minecraft's systems** (like scoreboard enhancements and PDC on more objects).

- **Modifying Attributes (Health, Speed, etc.):** Bukkit introduced the **Attribute API** (around 1.9), which Spigot/Paper fully support. This lets you get or set base values for attributes like `GENERIC_MAX_HEALTH`, `GENERIC_MOVEMENT_SPEED`, etc. For example, to give a player 50 health (25 hearts), you do `player.getAttribute(GENERIC_MAX_HEALTH).setBaseValue(50)`. All platforms can do this. This is crucial for MMO servers where you might have gear that increases max health or buffs that increase movement speed. Spigot's API covers applying **AttributeModifiers** to ItemStacks too (so an item when worn can add +5 Attack Damage, etc.). **Paper** didn't need to change this API since it's already sufficient, but Paper's item data components can also define attributes on items in a more data-driven way (e.g. the `ItemAttributeModifiers` component can embed attribute bonuses in an item's NBT easily [26]). Purpur doesn't add to this beyond inheriting it. One minor difference: Spigot had some limitations with updating health UI – after changing max health, you might need to refresh the player's health to see the hearts update (often by setting health to a lower value then back up). Paper tends to fix or smooth out such issues more quickly, but that's not an official API difference, just quality of implementation.

- **Custom "Derived" Stats:** Stats like *Defense* or *Crit Chance* usually don't map one-to-one to a Bukkit API concept. For defense (which could reduce damage taken), you handle it in the damage event: listen to `EntityDamageEvent`, calculate reduced damage based on the player's Defense stat, and use `event#setDamage` to apply it. This is the same in Spigot and Paper. Paper's advantage might come from the DamageType system if you want certain damage to bypass defense (you could mark true damage differently), but largely it's custom logic. For *Crit Chance*, you'd perhaps listen to EntityDamageByEntity, check if the attacker is a player and roll a random chance, then multiply damage or apply effects. All that is manual on any platform. There's no built-in "critical hit chance" attribute (vanilla critical hits are random if player is sprinting or falling, etc.), so your plugin implements it. The **scheduler** and **metadata** tools can help manage cooldowns or temporary stat boosts. For example, if a player uses a skill that increases strength by +50 for 10 seconds, you might set a metadata flag or PDC value and use a delayed task to remove it. This is the same pattern on Bukkit and Paper. Paper's *RegionScheduler* (from Folia) could be used to handle these timed tasks in a thread-safe way if you have many (so that scheduling 1000 buff expirations doesn't all hit the main thread at once) [14], but a normal Bukkit scheduler would also suffice unless performance is a concern.

- **Displaying Stats to Players:** Part of an MMO experience is showing players their stats (e.g. in a GUI, scoreboard, or action bar). All platforms let you create custom GUIs using the Inventory API (design an inventory with icons representing stats). There are community libraries to help with GUI (for instance, **SmartInvs** or **GuiAPI** – these work on any Bukkit-based server). Scoreboard sidebars are commonly used for quick stat displays ("❤ Health: 100"). **Paper's scoreboard component formatting** can let you replace the numeric score with a heart icon, etc., by using chat components [25]. This is unique to Paper/Purpur; on Spigot you'd have to continually update a text line on the scoreboard like "Health: 100" (which is fine, just not as fancy). For more dynamic displays, plugins

often use the **BossBar** API (since 1.9) to show things like a Mana bar at the top. All platforms support creating and updating BossBars. **Action bars** (the text above the hotbar) can be used for short stat info (like "+5 Strength!"). Spigot doesn't have a one-liner API for action bar, but you can use Spigot's ChatComponent API. Paper integrates the Kyori **Adventure** API, which provides a more modern way to send action bar messages and has nice formatting. In Purpur, you can of course use Adventure as well (since it's part of Paper). These differences are more about convenience – *any* combat stat or player stat system is achievable on base Spigot, but Paper's extras (Adventure, scoreboard formatting, etc.) let you polish the presentation and handle edge cases (like extremely large numbers, colored symbols) more easily.

Summary: Managing custom stats involves a lot of plugin-side logic and possibly databases; the **core API features (PDC, Attributes, Scoreboards)** are available on Spigot and improved slightly on Paper. Where **Paper/Purpur shine** is in the quality-of-life features: persistent data on almost anything, Adventure for formatted text, and performance considerations for scheduling many stat-related tasks. Purpur also adds a **Language API** that could be useful if your MMO supports multiple languages – you can fetch Minecraft translation keys in a target language (for item names, etc.) via `PurpurLanguage` class [27], which can help localize stat names or messages. In conclusion, **Spigot is capable but bare-bones**, while **Paper/Purpur provide a richer toolkit** to integrate custom stats seamlessly with the game (and with less worry about things like scoreboard limits or data saving).

Abilities and Skills (Mana, Cooldowns, AoE, Targeted Abilities)

An MMO plugin typically introduces special abilities or skills: spells with area damage, targeted attacks, mana or energy systems, and cooldown management. Implementing these relies on event handling, scheduling, and sometimes packet tricks – areas where Paper/Purpur can significantly ease development:

- **Mana Systems:** There's no native "mana" in Minecraft, so plugins create one. Common implementations include repurposing the XP bar or boss bar as a mana bar. **Bukkit/Spigot** allow setting a player's XP level/experience, which changes the XP bar visually – many MMO plugins show mana as the XP level (e.g. 100 mana = level 100, and they subtract as mana is used, possibly hiding actual XP gain). Alternatively, you can maintain mana internally and display it via a **BossBar** (the API for boss bars lets you set a custom bar with a name "Mana" and a progress value). All platforms support BossBar API (added in Spigot 1.9+). **Paper** doesn't add a custom "mana bar" object per se, but using the Adventure API you can have more control over boss bar styles and text. Purpur doesn't add anything special for mana either – it's implemented at the plugin level. One advantage of Purpur for mana might be performance: Purpur/Paper can handle frequent bossbar or action bar updates efficiently (since they optimize packet handling), which is useful if you're updating a mana bar 20 times a second.

- **Ability Cooldowns:** We touched on item cooldowns earlier (Player#setCooldown). For skill cooldowns not tied to an item (e.g. a class skill invoked by command or an empty-hand action), you'll implement your own timing – e.g. store a timestamp in a `cooldowns` map when a skill is used and prevent reuse until enough time passes. This is universal across platforms. However, if the skill *is* tied to an item (say a "Magic Wand" item), you can leverage the Spigot item cooldown API to show the cooldown visually (the item grays out). For example, after a player casts a spell with a wand, call `setCooldown(wandMaterial, 100 ticks)` to show a 5-second cooldown on that item [7]. Paper/Purpur fully support that and also have the `UseCooldown` item component which could let

you define cooldown behavior on the item itself (so the server *automatically* applies a cooldown after each use). Paper's component might allow specifying different cooldown durations for different actions or stacking multiple cooldowns, but using the simple API is often enough. **Purpur** doesn't introduce distinct cooldown mechanics beyond what Paper has.

- **AoE (Area of Effect) Abilities:** Usually, an AoE ability might create an explosion or affect entities in a radius. Bukkit/Spigot let you get entities in a radius via `World#getNearbyEntities(BoundingBox or Location,radius,...)` and apply effects (damage, potion, knockback). For visual effects, you spawn particles (explosions, spell particles) and sounds. This is standard and doesn't differ much on Paper. Where **Paper helps** is if you want more fine control or performance: Paper has an `AreaEffectCloud` entity API (vanilla lingering potion clouds) that you could spawn as a means to apply potion effects in an area automatically. Also, Paper's improved event system can handle large numbers of entities better – for instance, if your AoE hits 100 entities, doing that on Bukkit vs Paper might not change the code, but Paper's server performance with large entity lists is generally better (due to overall optimizations). If you wanted to simulate a *damage-over-time zone*, you might create a repeating task that checks all players in an area and damages them. Folia's region scheduler could confine that logic to that region thread, potentially improving concurrency.

- **Targeted Abilities (Raycasts and Projectiles):** Many skills target a single entity or a point (like shooting a fire beam or teleporting to a clicked location). Implementing this often requires ray tracing from the player's POV to find what they're aiming at. **Bukkit/Spigot** provide a simple method `player.getTargetBlockExact(distance)` but it only gets the block. If you want to hit entities, you'd use vector math or the `World#rayTraceEntities` / `rayTrace` methods (introduced in later Spigot versions). **Paper** enhances this with a **Raytracing API** (`io.papermc.paper.raytracing` package) which gives more powerful and flexible ray trace options, including filtering, ignoring certain entities, or getting precise hit positions. This can help for line-of-sight abilities and custom projectiles. Alternatively, launching actual projectiles (like arrows, snowballs, etc.) can be done via the Bukkit API (`World#spawnArrow` etc.), and then you might give them custom behavior by listening to events (e.g. `ProjectileHitEvent`). Paper doesn't need to change much there, though it does fix some inconsistencies and adds projectile-specific events (for example, `ProjectileCollideEvent` fires on Paper when a projectile is about to collide with an entity, allowing you to intercept it mid-air). Purpur adds a few niche events like `PlayerLaunchProjectileEvent` (not sure if Purpur has that specifically, but it has others like the spawner egg events). For targeted abilities, one often needs to send **packets** to create custom visuals (like a laser beam). Without external libraries, you might use the particle API to draw a line of particles. If you go deeper (custom entity models as projectiles, etc.), you'd likely use **ProtocolLib** (a library to intercept and send custom packets) – which is compatible with all platforms. However, note that **Paper has some packet events** now (e.g. you can listen for chunk packet send, or client tick packet) [28], but they haven't provided a general intercept-all-packets event (to avoid performance issues). For MMO devs, ProtocolLib remains a common friend for advanced visuals or interactions, and it works fine on Purpur/Paper.

- **Scheduler and Concurrency for Skills:** Many abilities involve delays (charge-up times, duration of effects, etc.). The **Bukkit scheduler** (and Java threads) are used across all platforms. Where you must be careful on Spigot is to not block the main thread. For example, a skill that affects blocks over time – you'd schedule small batches. **Paper's async and region schedulers** (Folia) allow more structured

concurrency. For instance, if you have a **world event** that runs every tick dealing damage in an area, on Folia you could attach that to the region's tick loop which spreads out work. Also, if your MMO does *a lot* of ability-related tasks (timers, projectile tracking, etc.), using Paper's scheduled tasks might scale better. Purpur (as of 1.21.5) is fully compatible with Folia's scheduler API – i.e., you can call `Server#getGlobalRegionScheduler()` or `getRegionScheduler()` on Purpur just as on Paper [29] [30]. If the server is running normal Paper, those calls are handled on the main thread for compatibility [31], and if running Folia, they truly run on multiple threads. This gives your plugin forward-compatibility with multi-threaded servers. In summary, ability logic often relies on **timers and events**, and Paper's advanced scheduler APIs make it easier to write thread-safe code for that.

Summary: Creating a rich abilities system is primarily the plugin's responsibility – **Bukkit/Spigot provide the basic tools** (events, particles, scheduler). **Paper** streamlines some of this (ray tracing utilities for targeting, projectile events, integration with Adventure for sending cool-down messages or sounds easily, better threading for timers). **Purpur** doesn't add new ability APIs, but it brings all of Paper's benefits. One Purpur-specific feature that *could* be handy is the **ridable entities system**: Purpur lets you make nearly any mob mountable and WASD-controllable via config [32]. If your MMO includes mounts or mount-based skills, Purpur saves you from coding a custom control system – you can toggle, say, "skeleton horses controllable" and then just use Minecraft's built-in movement. Purpur even has events like `RidableMoveEvent` when a custom ridable mob is moved by a player [4]. While not directly a combat ability, this shows Purpur's penchant for *gameplay feature APIs* that can complement an MMO (imagine a skill that temporarily lets you ride a dragon – Purpur makes the riding part trivial to enable). Overall, for ability systems, **Paper/Purpur's contributions are in making it easier and more performant** to handle complex sequences, whereas on Spigot you might have to do more manual work (and possibly deal with more packet-level hacks for things like lasers or shield effects).

Item Customization and Crafting

Custom items are the bread and butter of MMO servers – whether it's unique weapons with special effects, custom textured items, or new crafting recipes. All these platforms support extensive item customization, but Paper (and thus Purpur) have introduced powerful new APIs to treat items in a data-driven way:

- **NBT Tags and Persistent Data:** For years, plugins have used item NBT tags to store custom info (like an item's rarity, owner, custom stats). In older Bukkit, this required reflection or NMS access. Modern Spigot introduced `PersistentDataContainer` **on ItemMeta**, allowing plugins to attach arbitrary namespaced data to items in a officially supported way. This means on Spigot/Paper/Purpur you can do `itemMeta.getPersistentDataContainer().set(key, PersistentDataType.INTEGER, 42)` to tag an item. This data travels with the item, is saved, and can be read by your plugin later. It's ideal for marking custom items (e.g. a "flame sword" with an internal ID). So base Spigot already solves a lot of what used to require an **NBT API library**. That said, libraries like *ItemNBTAPI* (by tr7zw) still exist to manipulate NBT in bulk or for unsupported cases and work on all these servers. But with PDC available, you might not need them for 1.21+ development.

- **Custom Models and Textures:** Minecraft allows custom item models via the CustomModelData tag in item NBT, combined with a resource pack. Spigot's API exposes this: `ItemMeta#setCustomModelData(Integer)`. So you can assign model IDs to items to make them appear as entirely new items (client-side) with your resource pack. All three platforms support this

equally – it's part of Bukkit. Purpur doesn't change anything here specifically. You just ensure the player has the resource pack; you might use the Spigot `Player#setResourcePack(url,hash)` to prompt them. One note: Purpur has a config to allow large custom resource packs to be used (like increasing the pack size limit or load timeout), which can help if your MMO has a big resource pack.

- **Item Interaction & Behavior:** By default, to give an item special behavior, you handle events like `PlayerInteractEvent` (for right-click actions), `EntityDamageByEntityEvent` (if hitting something with the item), `PlayerItemConsumeEvent`, etc. Bukkit provides these hooks. **Paper's big innovation in 1.19+ is the *DataComponent API for items*** [33] [34]. This is a suite of components that attach to an ItemStack's NBT to give it properties that the server will recognize and act upon. Some highlights:

  - `BlocksAttacks`: Makes an item function like a shield (blocking attacks) with customizable block angle, sounds, and cooldowns [35] [36]. This means you could turn, say, a sword into a parrying dagger that can block with a smaller window. Without this, you'd script blocking logic yourself.
  - `Weapon`: Allows setting a custom base damage for an item and whether it disables shield blocking for a time (like axes do) [37] [38]. Using this, you can define weapons that deal a specific damage amount independent of their type – effectively custom weapon classes.
  - `Tool`: We discussed above – it lets you define what blocks the item can break and at what speed, even if it's not normally a tool [39] [11]. For example, make a gold pickaxe actually mine faster than diamond on certain blocks, or let a special item act as shears + axe combo.
  - `Consumable`: Define custom food/potion behavior for an item (hunger, saturation, status effects when consumed) without turning it into a vanilla food item [40] [41]. Useful for custom buff items or potions.
  - **Many others**: e.g. `ItemLore` for automatically adding lore lines, `Enchantment` for storing enchant data, `Durability` (not listed above but likely managed through existing API), `DeathProtection` (item that prevents death and breaks instead) [42], etc.

These DataComponents essentially allow **declarative item design**. Instead of writing listeners to check "if item is X, cancel damage and do Y", you can attach a component and the server handles it. This is extremely powerful for an MMO with hundreds of custom items – it moves logic to data where possible. Note that this is **Paper/Purpur only**; Spigot has nothing like it. Purpur includes the same API (since it's in Paper's namespace) – as a developer you'd shade in Paper's API to use it. It's marked experimental, but it's available in 1.21.5 snapshots. For example, to create a custom **"Thunder Sword"** that strikes lightning on hit, you might still need an event (no built-in component for that specific effect), but you *could* use the Weapon component to give it a base damage and blocking disable, and then just add a small event handler for the lightning part. Without components, you'd handle everything in the event.

- **Custom Crafting and Recipes:** Bukkit has long provided a Recipes API. You can create **Shaped** and **Shapeless recipes**, Furnace recipes, etc., and add them to the server. This allows players to craft your custom items. On Spigot, you can even require certain item meta by using `RecipeChoice.ExactChoice` (to require a specific ItemStack as an ingredient). That means, for instance, you can set a recipe that requires an item with a specific PDC tag or custom model. This was a more recent addition – older versions couldn't differentiate items by NBT in recipes easily. **Paper** extends the recipes system with minor improvements; for example, Paper has an event `PrepareResultEvent` for smithing table and other new station recipes (Spigot might have that by now too). Purpur adds very specific things like the events for anvil and grindstone usage [43] [44],

which can be used to implement custom "crafting" via those interfaces. For instance, Purpur's `AnvilUpdateResultEvent` fires whenever the output slot of an anvil is recalculated [44] – your plugin can intercept this to set a custom result (and Purpur allows bigger max anvil stack sizes via config, etc.). This is extremely useful if you want, say, an anvil to combine items into a custom outcome (e.g. put a special rune on a sword by combining in an anvil). Without that event, you'd have to monitor inventory clicks and simulate the anvil logic. Purpur also has `GrindstoneTakeResultEvent` [43] so you could reward custom items when players use a grindstone in a certain way. These are **Purpur-unique hooks** that are not in Paper by default.

- **Item Metadata and Display:** All platforms let you customize item name and lore (ItemMeta API). Spigot added the concept of **localizedName** and extra custom tags, but those are rarely used. Paper does integrate with the Adventure API, so you can set item lore with gradient colors or hex colors easily (Spigot now supports hex in legacy color codes too). Purpur doesn't add anything here except perhaps removing default chat color limits in config. Also notable: **Paper/Purpur support the hoverable tooltips for items in chat** via Adventure – on Spigot you use `player.spigot().sendMessage(BaseComponent…)` with a hover event.

Summary: In item customization, **Spigot gives you the toolbox (ItemMeta, recipes, events)** to do anything, but **Paper/Purpur provide power tools** that can save a ton of time and complexity. The new Paper item data API essentially lets you implement mod-like item behavior (custom shields, tools, etc.) *without writing low-level code for each*. Purpur builds on that by offering event hooks for vanilla UIs like anvil/grindstone, and config options for common item tweaks (e.g. **better-enchanted mending logic** to target most-damaged item first [45], or toggling whether shovels can create grass paths on various blocks [46] – these can shape gameplay but are transparent to your plugin). For a large MMO plugin, leveraging Paper's item API and Purpur's ready-made options means you can focus more on game design rather than reinventing mechanics. On Spigot, everything is doable, but you'll end up writing more event handlers (for custom crafting results, for item abilities, etc.) that Paper/Purpur might handle inherently.

Player Data Management and Persistence

Persistent player data (saving player stats, quest progress, etc.) and runtime metadata (storing transient flags like "in combat" or "combo counter") are critical in MMO plugins. The platforms largely share methods here, but there are nuances:

- **Persistent Storage of Player Data:** Typically done via external storage (YAML files, SQLite/MySQL, etc.) – this is outside the scope of the Minecraft API itself. All servers support reading/writing files from your plugin. However, as mentioned earlier, the **PersistentDataContainer** (PDC) on players can be used to save small pieces of data directly in the world save. This is suitable for things tightly coupled to the player in-game (like "hasCompletedTutorial" flag or intrinsic stats). All three support this, but Paper/Purpur's advantage might be that they keep up with any improvements to NBT limits or offer better integration. For example, if Mojang increases the capacity of player NBT, Paper will support it immediately. Purpur doesn't modify how data is saved except offering perhaps more configuration for saving intervals and such.

- **Metadata vs PDC: Bukkit's Metadata API** (`player.setMetadata(key, new FixedMetadataValue(...))`) allows attaching data to entities during runtime. This data is not

persisted to disk – it's just stored in memory for the life of that entity. It's useful for temporary flags (like marking a player as "bleeding" for 5 seconds). This works on all Bukkit platforms. However, many consider Metadata API clunky and instead use their own maps or PDC for most cases. PDC can serve for both persistent and non-persistent (you can always remove it later). One thing to note: Spigot's metadata and PDC calls are all thread-safe only if used on main thread. If you use async tasks, you must be careful (which is where Paper's Folia can ensure tasks run on correct threads).

- **Scoreboards and Stat Tracking:** We discussed using scoreboards to display stats, but they can also store data. For example, you might create a scoreboard objective "playtime" and increment it – this is persisted in the world and can be accessed via the API. All platforms allow that. Paper doesn't change scoreboard functionality (aside from formatting), and Purpur doesn't either. One difference: Purpur's config can alter some scoreboard constraints (like allowing longer team prefixes/suffixes which are normally limited to 16 chars in vanilla). This can help if you use scoreboard teams to denote factions or titles. It's a minor thing but can be convenient.

- **Player Metadata and Tags:** Minecraft itself has the concept of **"scoreboard tags"** on entities (strings you can attach via `/tag` command). The Bukkit API exposes this via `Entity#getScoreboardTags()`. These tags persist with the entity (in NBT) and are often used for marking entities for commands or other plugins. You could use them, for example, to mark NPCs with a specific role ("QuestGiver") so that other systems or command blocks can identify them. All platforms support this equally; Paper/Purpur don't particularly expand this.

- **External Libraries for Data:** Many large plugins integrate with databases. While not part of the Bukkit API, it's worth noting: if using an ORM or database pool, Paper and Purpur are built on Java 17+ and have no issues with those libraries. Purpur includes some quality-of-life like a built-in Hikari connection pool in the server (Pufferfish patches, possibly), but that's more on the server internals side. From the API perspective, they all allow you to manage your data as needed. One thing – if you use something like **LuckPerms API** or **PlaceholderAPI** to integrate data (like storing player stats as permissions or exposing them as placeholders), all of that works regardless of Paper/Purpur, since those are plugin-level integrations.

- **Data Versioning and Conversion:** Paper tends to be quicker in supporting the newest data formats (e.g. player data version upgrades). If your MMO plugin retains data between MC updates, running on Paper might mean slightly smoother transitions, but generally it's the same.

Summary: In terms of API, **there's no dramatic difference in how you manage player data** – it's largely up to your plugin's architecture. **Paper/Purpur's advantages** lie in performance (you can schedule data save tasks asynchronously easily, and Folia's thread model can prevent blocking when saving lots of data) and some extended capabilities like using PDC freely. Purpur aims to **"make small plugins unnecessary"**, so it offers config for things like auto-saving player data more frequently or not at all (if you want to control saving manually). As an MMO developer, you'll still design your data storage system, but you can trust that on Paper/Purpur you have all the same tools plus some extra reliability. For example, if you attach a large NBT structure to a player via PDC (say a custom quest log in JSON), Paper can handle large NBT better and might increase limits where Spigot could not (this is hypothetical, but Paper has been known to raise certain hard limits if safe). In conclusion, **Spigot vs Paper vs Purpur in player data is mostly "no difference in capability, just in convenience and performance."** Using scoreboards, PDC, metadata, or external DB

works everywhere – just remember that Paper's environment might handle lots of concurrent tasks more gracefully, which matters if your MMO is tracking and saving data for hundreds of players constantly.

Mob and Boss Design (AI & Custom Behaviors)

Creating custom mobs and bosses – with special AI, unique attacks, attributes – is one of the hardest parts of a Minecraft MMO. Much of Minecraft's mob AI isn't exposed in the API, but the platforms have some tools and differences:

- **Mob Attributes and Simple Tweaks:** Changing mob stats (health, damage, speed) is straightforward on all platforms via Attribute API (just like players). You can spawn entities and adjust their `GENERIC_MAX_HEALTH`, etc., or apply potion effects. Spigot added methods like `LivingEntity#setAI(boolean)` to disable the AI of a mob (making it stationary, good for NPCs) – available on all modern servers. Purpur gives config options to **set default attributes for mobs** in `purpur.yml`. For example, you can globally scale mob health or damage. In Purpur's config, you'll find per-mob settings like setting a zombie's base attack damage or whether a mob has AI at all [47]. This is done without any plugin – so if you want all zombies to be faster and stronger in your MMO, Purpur can do that by config, whereas on Spigot you'd write a plugin to buff them on spawn. Purpur's config can also turn normally passive mobs hostile or vice versa (e.g. make cows aggressive) and other quirky options. Those are unique to Purpur and can be valuable if you want to drastically change vanilla behavior globally.

- **Custom AI Goals:** This is where **Bukkit/Spigot have no official API**. If you want to give a mob a new behavior (like a zombie that fires a ranged attack, or a boss that teleports), you usually have to resort to NMS (accessing the internal PathfinderGoal system) or use a library. There are libraries and frameworks – e.g. **Citizens NPC** can create controllable NPCs, and **MythicMobs** is a plugin that allows scriptable custom mob skills and AI (via its own scripting language). Many MMO servers use MythicMobs or similar because doing it from scratch is cumbersome on Bukkit. **Paper/Purpur unfortunately do not (yet) provide a high-level custom AI API**. They have not abstracted the pathfinding system to an API, likely because it's complex and changes with Minecraft versions. However, Paper has made some improvements: it has an `EntityMoveEvent` [48] which fires when any living entity moves – allowing you to monitor or cancel movement. For instance, you could use that to implement a "root" effect (cancel movement if the entity has a rooted metadata). Paper also has `EntityToggleSitEvent` for wolves/pets, `EntityPathfindEvent` in some forks (Pufferfish maybe) to detect path calculations – but those are minor. **Purpur** doesn't provide new AI controls either; what it does is offer toggles and events for specific vanilla behaviors. For example, Purpur has events like `BeePollinateEvents` (if you cared to hook into a bee's pollination) [49], or `GoatRamEvent`, which are fun but rarely needed in an MMO (unless you really want to, say, prevent goats from ramming players by cancelling that event). Purpur's riding system (mentioned earlier) effectively *injects AI for mounts* – behind the scenes it gives mobs the ability to be controlled by players, which is something not in Spigot.

- **Boss Mechanics:** Complex boss mechanics (multi-phase fights, summoning minions, special attacks) will mostly be your plugin logic using a combination of scheduled tasks and event manipulation. For instance, you might listen to a boss's `EntityDamageEvent` to detect when health drops below X and then trigger a "phase change" (maybe spawn lightning and minions). All that can be done on any

platform. The differences are subtle: Paper's better event set can help (e.g., Paper has `WardenAngerChangeEvent` [50] which could be handy if you have custom Warden-based boss and want to manipulate its target anger). Also, Paper's ability to **spawn custom particles or entities without as much lag** (due to performance improvements) means you can have fancier boss effects. Purpur, with Pufferfish patches, further improves server performance under load, which is indirectly beneficial if your boss fights involve many entities or projectiles.

- **Mob Persistence and Tracking:** For an MMO, you might keep custom data on mobs (like an "elite" flag or level). You can again use PersistentDataContainer on Entities to store custom NBT on mobs in Spigot/Paper. That's helpful if a mob unloads and reloads – it retains your data. All platforms support that (since 1.14). Paper extended PDC to also be available on TileEntities and a broader range of objects (Spigot covered entities and items though, so it's similar). Purpur doesn't change that.

- **Integration with Pathfinding Systems:** If you do go down to NMS, Paper/Purpur usually maintain better mappings and slightly cleaner NMS. Purpur might also accept some NMS-based PRs that Paper wouldn't (since Paper tries to stick to API or performance, Purpur sometimes includes "fun" patches). For example, Purpur had in the past some patch to allow controlling mob spawning more granularly, but that might be config only. Speaking of spawning: Purpur's config can fine-tune mob spawning (e.g. spawn limits, per-biome counts, etc.), which can help ensure your custom mobs don't get overwhelmed by vanilla spawns or vice versa.

- **Library and Plugin Ecosystem:** Not strictly API, but noteworthy: If you use **MythicMobs** or **Citizens**, those are tested on Spigot/Paper and generally also work on Purpur (Purpur is a drop-in replacement API-wise). Paper's community often provides addons that take advantage of Paper features (like a MythicMobs module that uses Paper for certain effects). When building an MMO, sometimes using these existing frameworks can accelerate development (e.g., let MythicMobs handle custom AI of mobs and just integrate with it). Paper/Purpur's benefit here is just that they are 100% compatible and the performance overhead is lower.

Summary: Bukkit/Spigot give only basic mob controls (spawn, set attribute, listen for damage/death). **Paper** adds some useful events (movement, special cases) and overall makes the server more robust to handle complex mob logic written in the plugin (due to async tasks, etc.). **Purpur** provides config-level control to change vanilla mob behavior (rideable, hostile toggle, attribute defaults) which can jump-start certain features without coding [32]. However, truly custom AI still largely requires custom coding or external libraries on any platform. In an MMO plugin, you might use a mix: script simple behaviors via events (possible on Spigot), and rely on platform optimizations for heavy lifting (Paper's region thread to handle a large boss arena perhaps). One thing Purpur uniquely offers is the ability to **fine-tune almost every mob rule** in the config – from creeper explosion radii to whether shulkers can spawn new shulkers. This means with Purpur you can sometimes achieve a design goal by config instead of code (for example, turning off enderman griefing or making villagers follow players, etc.). Those fine details can save development time.

In conclusion, designing advanced mobs/bosses remains one of the more challenging tasks and isn't *magically* solved by Paper or Purpur, but running your MMO on Paper gives you confidence that you can push the server with custom entity logic and not hit as many performance issues or missing hooks. Purpur goes further by offering a server config that is practically an MMO server config out-of-the-box (since many

options are inspired by gameplay customization needs), plus a few extra events to plug into those mechanics.

Unique PurpurMC Features for MMO Development

Beyond the comparisons above, it's worth highlighting **Purpur-specific enhancements** that Paper doesn't have, which can be valuable for an MMO server:

- **Extensive Configuration Options:** Purpur's philosophy is to provide toggles for many gameplay mechanics so you don't need a plugin. While not "API" in the coding sense, as a developer-admin you can use these to shape your world. For example, **drop spawners with Silk Touch** is a one-line config change [17], **baby mobs rideable** toggle for fun mount mechanics [51], separate creeper/ghast griefing controls, disabling certain damage types (like dragon or wither block damage), enabling PvP in specific worlds without plugin, etc. This can simplify your plugin: you focus on MMO-specific logic and let Purpur's config handle generic tweaks. It also reduces your maintenance, since Purpur keeps those toggles updated across MC versions.

- **Additional Event Hooks:** Purpur introduces a handful of events that Paper lacks, usually to complement its features. We mentioned `AnvilUpdateResultEvent` and `GrindstoneTakeResultEvent` for custom crafting via those stations [43]. There's also `PlayerAFKEvent` (Purpur can detect AFK status natively – you might use this to, say, pause certain regen or kick idlers) and `ExecuteCommandEvent` which fires whenever a command is run [52] (you could use this as a catch-all to implement custom command logging or restrictions in your MMO). Purpur's `PreBlockExplodeEvent` and `PreEntityExplodeEvent` [53] [4] allow you to finely control explosion physics globally (maybe your MMO has a mechanic where certain explosions don't destroy terrain – Purpur event makes it easy to cancel or alter the block list affected). These events are unique to Purpur; on Paper/Spigot you'd sometimes achieve similar outcome by other means (e.g. Paper has an Explosion API where you can modify the block list in `EntityExplodeEvent`, which Spigot also has to a degree, but Purpur gives a *pre* event before calculation).

- **Language and Permissions Helpers:** Purpur adds a `Language` API [27] that can translate Minecraft's locale strings. If you want to display item names or messages in the client's language (for a global MMO audience), this is useful. Purpur also has a `PurpurPermissions` utility (which likely registers its custom permission nodes like the spawner mining permission) – not directly MMO-related, but indicates Purpur is mindful of permission control for features.

- **Pufferfish and Tuinity Patches:** Purpur includes performance enhancements from forks like Pufferfish (and formerly Tuinity) [1]. This means better mob pathfinding performance, micro-optimizations in entity ticking, etc. For an MMO with possibly many entities (players, mobs, NPCs), these patches can provide a smoother experience beyond stock Paper. It doesn't change how you code your plugin, but it raises the ceiling for how much you can do *with* your plugin without TPS drops.

- **Community and Ecosystem:** While not an API feature, Purpur's community is often focused on creative server setups. This means you'll find examples and help for using Purpur features in unconventional ways. For instance, someone may have used Purpur's ridable bats to create flying

mount mini-games, or used the `storedEntity` (Purpur's API to store an entity inside a block, perhaps for custom spawners) to implement a capture system. Being on Purpur gives you access to those niche ideas that you can incorporate with less custom code.

Finally, it's important to mention **new and upcoming features in 1.21.5** that unlock previously difficult tasks. We've covered many: Paper's item DataComponent API (which is new and vastly improves custom item creation) [33] [34], the registry/datapack API (to define custom damage types or perhaps even custom worldgen, making things like "true damage" easier to implement cleanly), and Folia's region scheduler (multi-threaded tick tasks) that allows you to scale your MMO tasks across threads without breaking things [14]. These are cutting-edge in 1.21.5 and did not exist in older versions – for example, Hypixel SkyBlock's developers in earlier versions had to rely on heavy NMS for things like mining speed and custom items, but a modern 1.21.5 server with Paper/Purpur can utilize official APIs to achieve the same: - Hypixel's *custom mining (multi-hit ores)* can be implemented via Paper's BlockBreakProgressUpdateEvent plus the Tool component for items, rather than a spaghetti of packet listeners. - Hypixel's *"True Defense"* (reducing true damage) could be implemented by introducing a new DamageType (with no armor reduction) and handling a player stat to subtract from that damage in the event – something now achievable by hooking the new damage system, whereas before one had to intercept specific damage causes individually. - Hypixel's *forge (item combining with time delay)* can be done with a combination of Purpur's anvil events (if using an anvil-like GUI) or a custom GUI, and using the scheduler to handle timed crafting. The new schedulers ensure even if the server restarts or lags, you can potentially reschedule tasks on reboot, or use persistent data to remember finish times. While there isn't a single API call for "schedule item craft in 2 hours", the building blocks (persistent data + async scheduler) are robust and available. Additionally, something like the **GlobalRegionScheduler** on Paper could be used to run long-term tasks that aren't tied to any player's region (like a global countdown for a forge) without ticking on the main thread [29] [54].

To wrap up, here's a **brief comparison table** of the platforms for key MMO-related capabilities:

Feature Area	**Bukkit** (baseline)	**Spigot** (+ Bukkit)	**Paper** (+ Spigot)	**Purpur** (+ Paper)
Combat Hooks	Basic damage events, attributes	Same as Bukkit (minor additions)	Extra events (knockback, item damage) and DamageType registry [5]	Inherits Paper; plus pre-explosion events [53], config for mechanics
Mining & Blocks	Break/click events; no progress feedback	+ `Player.sendBlockDamage` visual [9]	BreakProgressEvent for mining [8]; Tool API for custom mining speed [11]	Inherits Paper; config for silk spawners, etc. [17]

Feature Area	**Bukkit** (baseline)	**Spigot** (+ Bukkit)	**Paper** (+ Spigot)	**Purpur** (+ Paper)
World Generation	Custom ChunkGenerator API (limited); structure API	(same as Bukkit)	Async/region generation, datapack/registry events for custom world data	Inherits Paper; config toggles for world rules (mob spawning, griefing, etc.)
Custom Stats	Store in files or scoreboard; attributes for HP/speed	(same; has item cooldown API)	PDC for persistent stats on entities; Adventure API for display; new scheduler for concurrent tasks [14]	Inherits Paper; Language API for locale [27]; extensive server configs for stat-related rules (e.g. hunger, regen rates)
Abilities & Skills	Events and schedulers; no built-in mana/ cooldown UI	+Item cooldown UI [7]	Adventure for action bar/bossbar; Raytrace API for targeting; Folia schedulers for async skill tasks	Inherits Paper; rideable-any-mob feature [32]; a few extra events (command exec, etc.)
Items & Crafting	ItemMeta (name/lore/ Enchant); custom recipes; events for use	+JSON text in lore, ExactChoice recipe matching	**Item Component API** (custom tool/ weapon behavior) [39] [37]; custom model data support; Paper has grindstone/ smithing events too (since MC added)	Inherits Paper; extra Anvil/ Grindstone events [43]; config for mechanics like better mending [45]

Feature Area	**Bukkit** (baseline)	**Spigot** (+ Bukkit)	**Paper** (+ Spigot)	**Purpur** (+ Paper)
Player Data	Metadata, scoreboard, config files; PDC (1.14+)	(same as Bukkit)	No special API changes (just more efficient handling); Folia threads avoid data save lag	Inherits Paper; built-in config options for auto-saving, etc. (minor)
Mobs & Bosses	Attributes, potion effects; no AI API (use NMS or MythicMobs)	(same; +setAI(false) method)	EntityMoveEvent, etc. for control; overall better performance for many entities	Inherits Paper; **massive config for mob AI/ behavior** toggles [47] [32]; niche events (bee, llama caravan) for custom interactions

(Table legend: Purpur "config" refers to purpur.yml options that alter gameplay without coding.)

Conclusion

Bukkit/Spigot provide the fundamental API needed to create an MMO plugin, but many advanced features require custom work and sometimes workarounds. **PaperMC 1.21.5** builds on that with numerous additions: new event hooks (for finer control of combat, block breaking, etc.), the ability to tie into Minecraft's data-driven systems (registries for damage types, custom worldgen data), and performance-oriented utilities like regionized schedulers. **PurpurMC 1.21.5** includes all of Paper's capabilities and layers on a vast array of configuration options and a few targeted API additions that cater to server gameplay customization. For developing a large MMO-like plugin engine, Purpur offers the **widest feature set and flexibility**, letting you achieve things via config or minor code that would otherwise require significant development.

In practice, if you develop against Purpur/Paper's API, you can still maintain compatibility with Spigot (falling back gracefully when a Paper-specific event or method isn't present). This is a strategy to consider – you get the benefits on Purpur/Paper, while not excluding Spigot entirely. But many MMO servers will choose Purpur for the **enhanced experience**: better performance under load and many quality-of-life features for both developers and server owners [55] [56]. With Minecraft 1.21.5's new API features, systems that were once hacks (e.g. Hypixel's mining speed, custom item abilities, complex crafting mechanics) can

now be implemented in a cleaner, more supported way. This ultimately leads to a more maintainable codebase and a smoother gameplay experience.

References:

- Bukkit/Spigot 1.21.5 API Javadocs (Entity, Player, etc.) [9] [7]
- PaperMC 1.21.5 API Javadocs (events and new item components) [8] [11] [37]
- PurpurMC Documentation and Javadocs (unique events and config examples) [53] [43] [17]
- PaperMC Developer Docs on Folia (scheduler improvements) [14]
- Reddit – Purpur vs Paper overview (performance and features) [1] [55]

---

[1] [55] [56] What are the advantages of Purpurmc over Papermc? : r/admincraft
https://www.reddit.com/r/admincraft/comments/rz2483/what_are_the_advantages_of_purpurmc_over_papermc/

[2] [3] [48] [50] io.papermc.paper.event.entity (Scissors-API 1.20.1-R0.1-SNAPSHOT API)
https://javadoc.scissors.gg/1.20.1/io/papermc/paper/event/entity/package-summary.html

[4] [49] org.purpurmc.purpur.event.entity (purpur-api 1.21.4-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/org/purpurmc/purpur/event/entity/package-summary.html

[5] DamageType (paper-api 1.21.1-R0.1-SNAPSHOT API)
https://jd.papermc.io/paper/1.21.1/org/bukkit/damage/DamageType.html

[6] Uses of Interface io.papermc.paper.registry.data ... - Purpur
https://purpurmc.org/javadoc/io/papermc/paper/registry/data/class-use/DamageTypeRegistryEntry.Builder.html

[7] [24] HumanEntity (Spigot-API 1.21.5-R0.1-SNAPSHOT API)
https://hub.spigotmc.org/javadocs/bukkit/org/bukkit/entity/HumanEntity.html

[8] Spigot Event List
https://spigot-event-list.s7a.dev/?tags=paper-purpur

[9] [10] Player (Spigot-API 1.21.5-R0.1-SNAPSHOT API)
https://hub.spigotmc.org/javadocs/bukkit/org/bukkit/entity/Player.html

[11] [12] [13] [34] [39] Tool (purpur-api 1.21.5-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/io/papermc/paper/datacomponent/item/Tool.html

[14] [15] [16] [30] [31] [54] Supporting Paper and Folia | PaperMC Docs
https://docs.papermc.io/paper/dev/folia-support/

[17] [18] [22] [23] [32] [45] [46] [47] [51] Configuration - PurpurMC Documentation
https://purpurmc.org/docs/purpur/configuration/

[19] Pufferfish-1.20 #48 Changes [Jenkins] - Dashboard [Jenkins]
https://ci.pufferfish.host/job/Pufferfish-1.20/48/changes

[20] [21] Overview (purpur-api 1.21.5-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/

[25] io.papermc.paper.scoreboard.numbers (purpur-api 1.21.5-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/io/papermc/paper/scoreboard/numbers/package-summary.html

26 33 40 41 42 io.papermc.paper.datacomponent.item (purpur-api 1.21.5-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/io/papermc/paper/datacomponent/item/package-summary.html

27 org.purpurmc.purpur.language (purpur-api 1.21.5-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/org/purpurmc/purpur/language/package-summary.html

28 io.papermc.paper.event.packet (purpur-api 1.21.5-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/io/papermc/paper/event/packet/package-summary.html

29 Uses of Interface io.papermc.paper.threadedregions.scheduler ...
https://purpurmc.org/javadoc/io/papermc/paper/threadedregions/scheduler/class-use/RegionScheduler.html

35 36 BlocksAttacks (purpur-api 1.21.5-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/io/papermc/paper/datacomponent/item/BlocksAttacks.html

37 38 Weapon (purpur-api 1.21.5-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/io/papermc/paper/datacomponent/item/Weapon.html

43 44 org.purpurmc.purpur.event.inventory (purpur-api 1.21.4-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/org/purpurmc/purpur/event/inventory/package-summary.html

52 53 org.purpurmc.purpur.event (purpur-api 1.21.4-R0.1-SNAPSHOT API)
https://purpurmc.org/javadoc/org/purpurmc/purpur/event/package-summary.html