Tyracoon RP

Présentation du projet

Architecture — Serveur RP Minecraft Moddé 1.20.1 (Forge)

Contexte du projet

Serveur semi-RP **Tyracoon RP** — Minecraft 1.20.1 Forge avec :

- Une capitale centrale gérée par une team admin, protégée du griefing
- 7 classes (spécialisations) restreignant l'accès à certains mods/items
- Un système d'économie en **Tyracoins** avec hôtel des ventes
- Une interface joueur custom (GUI) pour remplacer les commandes texte
- Deux mods distincts : un client, un serveur
- Accès libre via whitelist automatique Discord → RCON (sans validation manuelle)
- Capacité cible au lancement : **64 joueurs**

Classes (spécialisations)

Les 7 classes

Classe	Mods restreints	Ambiance
Sentinelles	Simply Swords, ArmorPlus	Combat pur, armes et armures avancées
Arcanistes	Iron's Spells and Spellbooks, Relics	Magie offensive via reliques et sorts
Sylvanes	Feywild, Botania	Nature, féérique, magie florale
Dévoyés	Occultism, Born in Chaos	Invocation démonique, rituels occultes
Artificiers	Create, Scorched Guns 2	Technique, machines, armes à feu
Harmonistes	Paimon, Touhou Little Maid, Ars Nouveau	Soutien, soins, compagnons, magie de soin/buff/debuff

Classe	Mods restreints	Ambiance
Exsangues	Mahou Tsukai, Blood Magic	Magie par le sacrifice, puissance au prix de la vie

Mods sans restriction

Tout mod absent de toutes les listes de classes est accessible à tous par défaut. Le tri sera fait à partir de la liste complète des mods du modpack.

Restrictions spéciales

- **Harmonistes / Ars Nouveau** — sorts offensifs directs bridés via `SpellCastEvent`. Sorts de buff alliés et debuff ennemis autorisés. Seuls les glyphes d'attaque directe sont bloqués. Liste configurable dans `config/class_mods.toml`

Joueur sans classe (NULL)

Un joueur sans classe est en mode "salle d'attente" jusqu'au choix de spécialisation. Restrictions actives :

- **Bloqué au spawn/zone d'accueil** — téléporté de force si il quitte la zone
- **Aucune interaction avec les mods restreints** — tous les mods de classes sont bloqués
- **Accès aux zones de classes interdit**
- **Équipement vanilla uniquement**

La GUI de choix de classe s'ouvre automatiquement au premier login et ne peut pas être fermée sans choisir.

Changement de classe

- Définitif pour le joueur
- Commande admin uniquement : `/class set <joueur> <classe>`

Système de restrictions des mods

Approche souple — un joueur peut ramasser et revendre tout item, mais ne peut pas équiper/utiliser les items des mods attribués à d'autres classes. Ça anime l'économie via l'HDV.

Un seul fichier de config — par défaut tous les mods sont accessibles à tous. On définit uniquement les attributions par classe. Une classe voit automatiquement ce qui lui est interdit en regardant les attributions des autres.

```
# config/class_mods.toml

[sentinelles]
mods = ["simplyswords", "armorplus"]

[arcanistes]
mods = ["irons_spellbooks", "relics"]

[sylvanes]
mods = ["feywild", "botania"]

[devoyés]
mods = ["occultism", "born_in_chaos"]

[artificiers]
mods = ["create", "scorchedguns"]

[harmonistes]
mods = ["paimon", "touhou_little_maid", "ars_nouveau"]

[exsangues]
mods = ["mahou_tsukai", "blood_magic"]
```

Un mod absent de toutes les listes est accessible à tout le monde automatiquement (ex: Critters and Companions).

Logique de vérification à l'équipement :

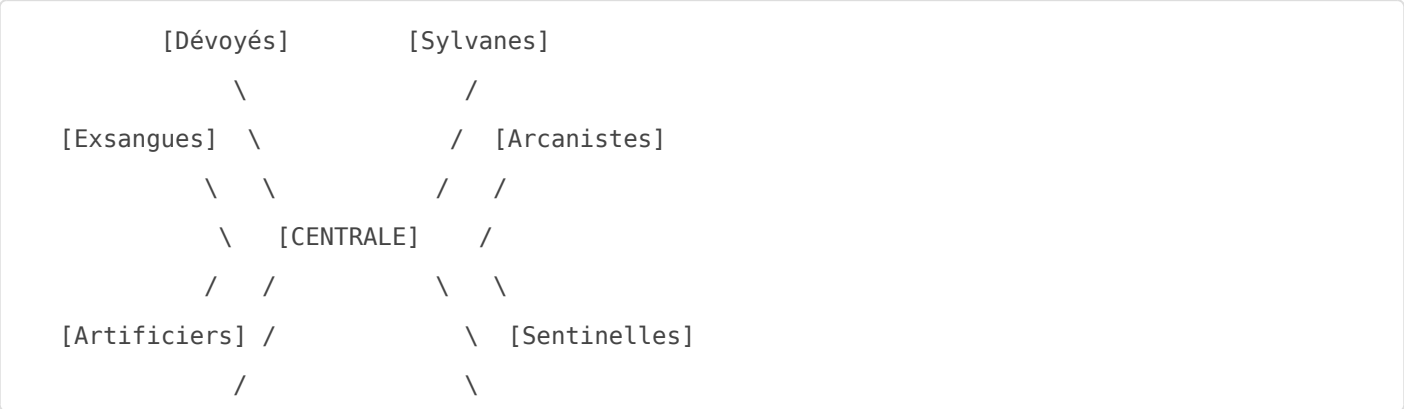
```
public static boolean canEquip(String classe, ItemStack item) {
    String modid = ForgeRegistries.ITEMS
        .getKey(item.getItem())
        .getNamespace();

    // Récupère tous les mods attribués à d'autres classes
    Set<String> forbiddenMods = ClasseConfig.getAllMods()
        .entrySet().stream()
        .filter(e -> !e.getKey().equals(classe))
        .flatMap(e -> e.getValue().stream())
        .collect(Collectors.toSet());
```

```
    return !forbiddenMods.contains(modid);
}
```

Restriction spéciale — Harmonistes / Ars Nouveau : sorts offensifs bridés via `SpellCastEvent`. Les glyphes bloqués sont listés dans `config/class_mods.toml` sous `[harmonistes.ars_nouveau_blocked_glyphs]`.

Game design — La Capitale

Structure urbaine



Zone Centrale (Harmonistes)

- Grande place avec fontaine
- PNJs marchands (loots mobs à prix fixe, ressources basiques)
- Hôtel des ventes (HDV)
- Hôtel de ville
- Guilde mère des Harmonistes + PNJ quêtes
- **5 plots** réservés aux guildes ayant accompli des exploits

Zones périphériques (×6)

Chaque zone dispose de :

- Guilde mère de la classe + PNJ quêtes
- Boutique fixe (catalogue TOML, prix fixes, stock illimité)
- **5 plots** achetables par les guildes de joueurs

Règles des plots de guildes

- Achat via interaction avec un panneau (`SignBlockEntity`) — le panneau s'actualise avec le nom de la guilde
- **1 plot maximum par guilde**
- Prix d'achat en Tyracoins défini par l'admin
- Volume autorisé : `y_base - 8` (fondations) → `y_base + 20` (bâtiment)
- Bâtiments pré-construits par la team admin, les guildes les personnalisent

Plots centraux et exploits

Les 5 plots de la zone centrale sont attribués **manuellement par les admins** via les commandes `/plot` selon leur jugement. Pas de système automatique — c'est une récompense discrétionnaire pour des accomplissements notables.

Progression de classe

Aucune — tout est accessible dès le choix de la classe. Pas de niveaux, pas de déblocages progressifs.

Économie — Tyracoins

Sources de revenus pour les joueurs

1. Vente de loots de mobs (prix fixes, PNJ central)

Configuré dans `config/mob_prices.toml` :

```
[mob_loot_prices]
"minecraft:bone" = 2
"minecraft:rotten_flesh" = 1
"minecraft:spider_eye" = 4
"minecraft:ender_pearl" = 4
"minecraft:string" = 1
```

2. Quêtes journalières — rotation à minuit, 5 quêtes par zone (35 quêtes/jour au total)

Configuré dans `config/quests/<classe>.toml` :

```
[[quests]]
id = "sentinelles_daily_01"
```

```
name = "Collecte de matériaux"
item = "minecraft:iron_sword"
quantity = 5
reward = 50
repeatable = false
```

Fonctionnement de la remise de quête : Le joueur retourne voir le PNJ de quête avec les items dans son inventaire. Le mod vérifie la présence et la quantité, supprime les items et crédite la récompense en Tyracoins. Pas de suivi de progression en temps réel — la validation se fait uniquement à la remise.

PNJ custom

Base villageois Minecraft avec texture personnalisée. Pas de mod NPC tiers — le villageois est retexturé et son comportement est overridé via le mod pour afficher les GUIs custom (catalogue boutique, liste de quêtes, rachat de loots).

3. Hôtel des ventes — échanges joueur à joueur en Tyracoins

Boutiques fixes par zone

Catalogue défini dans `config/shops/<zone>.toml`, prix fixes, stock illimité :

```
[[items]]
item = "create:wrench"
price = 15

[[items]]
item = "create:cogwheel"
price = 5
```

Commandes admin :

```
/shop reload <zone>    → recharge le catalogue d'une zone
/shop reload all       → recharge tous les catalogues
```

Structure du projet — Gradle multi-modules

Un seul projet Gradle produisant deux JARs distincts :

```
tyracoonrp/
├── build.gradle              # configuration Gradle commune
├── settings.gradle           # déclaration des modules
├── common/                   # code partagé — pas de JAR, compilé dans les deux mods
│   └── src/
├── tyracoonrp-client/        # mod client (GUIs, intégrations JEI/Patchouli)
│   └── src/
└── tyracoonrp-server/        # mod serveur (BDD, logique, commandes)
    └── src/
```

`settings.gradle` :

```
rootProject.name = 'tyracoonrp'

include 'common'
include 'tyracoonrp-client'
include 'tyracoonrp-server'
```

Dépendance `common` dans chaque module :

```
// tyracoonrp-client/build.gradle et tyracoonrp-server/build.gradle
dependencies {
    implementation project(':common')
}
```

JARs produits :

- `tyracoonrp-client-1.0.0.jar` → distribué dans la modpack
- `tyracoonrp-server-1.0.0.jar` → installé sur le serveur uniquement

Élément	common	client	server
Packets réseau (structure)	✅		
Modèles de données (Guild, Player...)	✅		
GUIs, Screens		✅	
Intégration JEI		✅	
Logique métier (quêtes, économie, PvP)			✅
Connexion MySQL			✅

Élément	common	client	server
Commandes admin			□
Gestion des events Forge			□
Envoi/réception packets	□		

Mods FTB (protection de base)

La suite FTB gère la protection globale de la capitale, sans développement custom :

Mod	Rôle
FTB Teams	Gestion des groupes et de la team admin
FTB Chunks	Claim de la capitale entière par la team admin
FTB Ranks	Permissions fines par rang (interaction PNJ, boutons, etc.)
FTB Essentials	Commandes utiles (/home, /tpa, etc.)

La protection globale de la capitale est entièrement déléguée à FTB. Le mod custom intervient **par dessus** pour la logique RP (guildes, plots, économie).

Mod Serveur

Responsabilités

- Gestion des spécialisations joueurs
- Système de plots de guildes dans la capitale (par dessus FTB Chunks)
- Système d'économie et hôtel des ventes
- Connexion MySQL
- Fichiers de configuration (règles par spécialisation, paramètres économiques)

Hiérarchie des permissions dans la capitale

```
FTB Chunks (couche 1 - protection globale)
└── Mod custom (couche 2 - logique guildes)
    ├── Admin team     → tout faire partout
    ├── Guilde X       → droits dans son plot uniquement
```

```
    └── Joueur lambda  → interactions publiques uniquement
```

Protection des plots de guildes

- Vérification dans `BlockBreakEvent` / `BlockPlaceEvent` si le bloc est dans un plot
- Si plot → vérifie que le joueur est membre de la guilde propriétaire
- Normalisation des coordonnées à la création (`Math.min/max`) pour éviter les plots invalides

```
GuildPlot plot = PlotManager.getPlotAt(pos);
if (plot != null) {
    int yMin = plot.getYBase() - plot.getYOffsetDown();
    int yMax = plot.getYBase() + plot.getYOffsetUp();
    boolean inVolume = pos.getY() >= yMin && pos.getY() <= yMax;
    if (!inVolume || !plot.isMember(player.getUUID())) {
        event.setCanceled(true);
    }
}
```

Fichiers de configuration (TOML)

- Liste des spécialisations et leurs restrictions associées
- Paramètres économiques (taxes, durée des annonces, etc.)
- Valeurs par défaut des plots (y_offset_down, y_offset_up)

Mod Client

Responsabilités

- Affichage des GUIs custom (menu joueur, hôtel des ventes, choix de spécialisation)
- Intégration JEI
- Dépendance Patchouli (livre de guide)

Interface joueur

- Commande `/menu` → le serveur envoie un packet → le client ouvre un `Screen` custom
- Les actions disponibles dans le menu varient selon la spécialisation du joueur
- Chaque action dans la GUI envoie un packet au serveur pour exécution
- Évite aux joueurs de mémoriser des commandes texte, plus immersif en RP

GUIs prévues

- Écran de choix de spécialisation (premier login)
- Menu principal joueur (solde, spécialisation, accès aux fonctions)
- Interface hôtel des ventes (liste d'annonces, achat, création d'annonce)
- Panel admin (gestion des plots, guildes, reset de spécialisation)

Module Common (partagé)

Contient tout ce qui est utilisé par les deux mods :

```
common/packets/
├── OpenMenuPacket.java        # serveur → client : ouvre une GUI spécifique
├── PlayerActionPacket.java    # client → serveur : action depuis la GUI
├── SyncPlayerDataPacket.java  # serveur → client : sync spé, solde, etc.
└── AuctionPacket.java         # actions liées à l'hôtel des ventes
```

Base de données — MySQL

MySQL choisi pour :

- Transactions ACID (achats concurrents à l'hôtel des ventes fiables)
- Gestion de la concurrence multi-joueurs
- Requêtes complexes (filtres sur les annonces, historique, stats)
- Centralisation possible si multi-serveurs à terme

Schéma SQL

```
-- Joueurs
CREATE TABLE players (
    uuid          VARCHAR(36) PRIMARY KEY,
    username      VARCHAR(16) NOT NULL,
    specialisation VARCHAR(64),
    balance       DECIMAL(15, 2) DEFAULT 0.00,
    created_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
-- Hôtel des ventes
CREATE TABLE auction_listings (
    id                    INT AUTO_INCREMENT PRIMARY KEY,
    seller_uuid           VARCHAR(36) NOT NULL,
    item_id               VARCHAR(255) NOT NULL,
    item_nbt              TEXT,                          -- NBT sérialisé pour items moddés
    quantity              INT NOT NULL,
    price                 DECIMAL(15, 2) NOT NULL,
    specialisation_required VARCHAR(64),                 -- null = accessible à tous
    created_at            TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expires_at            TIMESTAMP,
    status                ENUM('active', 'sold', 'expired', 'cancelled') DEFAULT 'active',
    FOREIGN KEY (seller_uuid) REFERENCES players(uuid)
);

-- Historique des transactions (immuable)
CREATE TABLE transactions (
    id          INT AUTO_INCREMENT PRIMARY KEY,
    buyer_uuid  VARCHAR(36) NOT NULL,
    seller_uuid VARCHAR(36) NOT NULL,
    listing_id  INT NOT NULL,
    amount      DECIMAL(15, 2) NOT NULL,
    timestamp   TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (buyer_uuid)  REFERENCES players(uuid),
    FOREIGN KEY (seller_uuid) REFERENCES players(uuid),
    FOREIGN KEY (listing_id)  REFERENCES auction_listings(id)
);

-- Guildes
CREATE TABLE guilds (
    id          INT AUTO_INCREMENT PRIMARY KEY,
    name        VARCHAR(64) NOT NULL,
    owner_uuid  VARCHAR(36) NOT NULL,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (owner_uuid) REFERENCES players(uuid)
);

-- Membres de guilde
CREATE TABLE guild_members (
    guild_id    INT NOT NULL,
```

```
    player_uuid VARCHAR(36) NOT NULL,
    role        ENUM('OWNER', 'OFFICER', 'MEMBER') NOT NULL,
    PRIMARY KEY (guild_id, player_uuid),
    FOREIGN KEY (guild_id)    REFERENCES guilds(id),
    FOREIGN KEY (player_uuid) REFERENCES players(uuid)
);

-- Plots de guildes dans la capitale
-- Protection globale déléguée à FTB Chunks
-- Ce système gère uniquement les sous-zones de guildes
CREATE TABLE guild_plots (
    id              INT AUTO_INCREMENT PRIMARY KEY,
    name            VARCHAR(64) NOT NULL,
    x1 INT, z1 INT,
    x2 INT, z2 INT,
    y_base          INT NOT NULL,
    y_offset_down   INT DEFAULT 8,
    y_offset_up     INT DEFAULT 20,
    dimension       VARCHAR(128) DEFAULT 'minecraft:overworld',
    status          ENUM('libre', 'occupé') DEFAULT 'libre',
    purchase_price  DECIMAL(10,2) DEFAULT 0,   -- prix en Tyracoins
    sign_x INT, sign_y INT, sign_z INT,         -- position du panneau d'achat
    guild_id        INT,
    assigned_at     TIMESTAMP,
    FOREIGN KEY (guild_id) REFERENCES guilds(id)
);
-- Suivi des quotas journaliers de vente aux PNJs
CREATE TABLE daily_sell_quotas (
    player_uuid   VARCHAR(36) NOT NULL,
    item_id       VARCHAR(255) NOT NULL,
    quantity_sold INT DEFAULT 0,
    quota_date    DATE NOT NULL,
    PRIMARY KEY (player_uuid, item_id, quota_date),
    FOREIGN KEY (player_uuid) REFERENCES players(uuid)
);

-- Catalogue des items rachetables par les PNJs
CREATE TABLE npc_buy_offers (
    id          INT AUTO_INCREMENT PRIMARY KEY,
    item_id     VARCHAR(255) NOT NULL,
```

```
    price       DECIMAL(10,2) NOT NULL,
    daily_quota INT NOT NULL,
    active      BOOLEAN DEFAULT TRUE
);

-- Quêtes disponibles (régénérées à minuit)
CREATE TABLE active_quests (
    id             INT AUTO_INCREMENT PRIMARY KEY,
    quest_id       VARCHAR(64) NOT NULL,
    specialisation VARCHAR(64) NOT NULL,
    item_id        VARCHAR(255) NOT NULL,
    quantity       INT NOT NULL,
    reward         DECIMAL(10,2) NOT NULL,
    available_date DATE NOT NULL
);

-- Quêtes acceptées/complétées par joueur
CREATE TABLE player_quests (
    id            INT AUTO_INCREMENT PRIMARY KEY,
    player_uuid   VARCHAR(36) NOT NULL,
    quest_id      VARCHAR(64) NOT NULL,
    progress      INT DEFAULT 0,
    status        ENUM('active', 'completed', 'expired') DEFAULT 'active',
    accepted_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at  TIMESTAMP,
    rotation_date DATE NOT NULL,
    FOREIGN KEY (player_uuid) REFERENCES players(uuid)
);
```

Implémentées via **Brigadier** (système de commandes natif Forge 1.20.1), niveau OP requis (`hasPermission(3)`).

```
/plot add <guild> <x1> <z1> <x2> <z2> <y> <name>   → créer un plot et l'assigner à une guilde
/plot remove <guild> <name>                          → supprimer un plot
/plot list [guild]                                   → lister les plots (d'une guilde ou
tous)
/plot info <name>                                    → détails d'un plot
```

Points importants :

- Les coordonnées sont normalisées automatiquement (`Math.min/max`) peu importe l'ordre de saisie
- Toutes les actions sont loguées dans le chat admin et les logs serveur (`sendSuccess(..., true)`)
- `y_offset_down` (défaut : 8) et `y_offset_up` (défaut : 20) sont ajustables plot par plot

---

Accès joueurs — Whitelist via Discord

Accès libre sans validation manuelle. Le joueur rejoint le Discord et tape `/link <pseudo_minecraft>` dans un channel dédié. Le bot Discord ajoute automatiquement le joueur en whitelist via RCON.

Ce que ça apporte :

- Zéro friction pour rejoindre
- Lien Discord ↔ UUID Minecraft → traçabilité pour les sanctions
- La communauté vit naturellement sur le Discord

Côté `server.properties` :

```
enable-rcon=true
rcon.port=25575
rcon.password=motdepassesécurisé
whitelist=true
```

Le bot envoie via RCON :

```
whitelist add <pseudo>
```

Infrastructure

Architecture globale

```
Internet
    │
    ├── Serveur Dédié
    │   └── Minecraft Forge 1.20.1 (:25565 → public)
    │
```

```
    ├── VPS
    │   ├── Wireguard VPN (:51820/udp → public)
    │   ├── Bot Discord (websocket sortant uniquement)
    │   ├── Bookstack (documentation équipe — déjà en place)
    │   └── Zabbix (monitoring)
    │
    └── OVH Object Storage
        └── Réplication backups (script + cron)
```

Flux d'administration

```
Toi/Collègue → Wireguard VPN → VPS → SSH rebond → Dédié
                                   → RCON        → Minecraft
                                   → MySQL        → Base de données
```

Firewall et VPN gérés intégralement par l'équipe — non documentés ici.

JEI (Just Enough Items)

- Enregistrement des items custom avec descriptions au survol
- Catégories de recettes custom par spécialisation
- Filtrage potentiel des recettes affichées selon la spécialisation du joueur connecté
- Mod **requis côté client**

Patchouli

- Livre de guide in-game entièrement en JSON (`resources/`), zéro code Java
- Contenu prévu :
  - Lore et contexte RP du serveur
  - Description de chaque spécialisation (avantages, restrictions)
  - Tutoriels illustrés (économie, hôtel des ventes)
  - Entrées déverrouillables progressivement (mécanique RP)
- Donné automatiquement au joueur lors du premier login
- Mod **requis côté client**

Dépendances `mods.toml`

Mod client

```
[[dependencies.clientmod]]
    modId = "forge"
    mandatory = true
    versionRange = "[47,)"

[[dependencies.clientmod]]
    modId = "jei"
    mandatory = true

[[dependencies.clientmod]]
    modId = "patchouli"
    mandatory = true
```

Mods requis sur le serveur (non développés)

- `ftbteams` — gestion des groupes
- `ftbchunks` — claim de la capitale
- `ftbranks` — permissions par rang
- `ftbessentials` — commandes utiles
- `ftbbackups2` — sauvegardes automatiques du monde

Points techniques à ne pas oublier

- **Java 17** requis pour Forge 1.20.1 (Java 21 incompatible)
- **Linux** (Ubuntu Server 24.04 LTS ou Debian 12) — évite 15-20% d'overhead vs Windows
- **JVM** : G1GC avec Aikar's Flags, `-Xms12G -Xmx20G` pour 32 GB RAM / 200 mods
- **JDBC MySQL** (HikariCP recommandé pour le pool de connexions) embarqué dans le mod serveur
- **item_nbt** stocké en texte sérialisé — indispensable pour conserver les metadata des items moddés
- Le mod serveur tourne **server-side only** — les joueurs n'ont pas à l'installer
- Le mod client est distribué dans la **modpack** avec JEI et Patchouli
- Prévoir une **migration SQL** versionnée dès le début (Flyway ou scripts numérotés à la main)
- **Synchronisation client/serveur** — checks effectués côté serveur à chaque échange sur le scope complet (équipement, quête, achat, PvP). Le client reçoit les données mises à jour via `SyncPlayerDataPacket` après chaque action validée

- **Redémarrage planifié** — annonce Discord + message in-game à J-15min/J-5min/J-1min via bot, sauvegarde propre via RCON (`save-all`), arrêt propre, redémarrage automatique via systemd

Stratégie de backup

Monde Minecraft — FTB Backup 2

Géré nativement par le mod, rotation automatique incluse. Rien à développer.

MySQL + configs — script cron + réplication OVH Object Storage

Dump MySQL et configs sauvegardés via script cron, réplication et rétention gérées par script vers **OVH Object Storage**.

```
#!/bin/bash
# /opt/backup.sh — MySQL + configs uniquement (monde géré par FTB Backup 2)
# Les secrets sont chargés depuis /etc/tyracoon/secrets.env
source /etc/tyracoon/secrets.env

# Dump MySQL
mysqldump -u "$MYSQL_USER" -p"$MYSQL_PASSWORD" --all-databases \
    | gzip > /tmp/mysql_$(date +%Y%m%d).sql.gz

# Réplication vers OVH Object Storage
restic backup /tmp/mysql_$(date +%Y%m%d).sql.gz \
              /opt/minecraft/config \
              /opt/minecraft/mods.toml

rm /tmp/mysql_$(date +%Y%m%d).sql.gz

# Rétention
restic forget --keep-daily 7 --keep-weekly 4 --keep-monthly 3 --prune
```

Crontab — exécution chaque nuit à 4h :

```
0 4 * * * /opt/backup.sh >> /var/log/backup.log 2>&1
```

Rétention

Type	Durée
Quotidien	7 jours
Hebdomadaire	4 semaines
Mensuel	3 mois

Monitoring — Zabbix + JMX

Zabbix déployé sur le VPS pour surveiller les deux machines.

Métriques système

- CPU, RAM, disque sur le dédié et le VPS
- Process Minecraft — détection de crash
- Process MySQL — détection d'arrêt
- TPS Minecraft via RCON (`/tps`) — alerte si TPS < 18
- Espace disque NVMe — alerte à 80% d'occupation

JMX — métriques JVM internes

JMX activé en local only sur le dédié, scrapé par l'agent Zabbix local. Zéro exposition réseau.

Arguments JVM à ajouter :

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=9010
-Dcom.sun.management.jmxremote.local.only=true
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

Métriques JMX surveillées :

- **Heap usage** — détection de fuites mémoire progressives
- **GC activity** — détecter si G1GC passe trop de temps à nettoyer
- **Thread count** — détecter deadlocks ou accumulation de threads

---

Sécurité complémentaire

- **Fail2ban** sur le VPS — protection contre les tentatives de brute force SSH résiduelles
- **logrotate** sur les logs Minecraft et MySQL — évite que le NVMe se remplisse silencieusement
- **Redémarrage automatique** via systemd en cas de crash Minecraft + notification Discord via webhook
- **MySQL** — compte `root` remote désactivé, utilisateur dédié avec permissions minimales pour le mod
- **Secrets** — mots de passe RCON, MySQL, Restic stockés dans `/etc/tyracoon/secrets.env` en `chmod 600`, chargés par les scripts et services

---

Bot Discord

Projet séparé, hébergé sur le VPS. Discord dédié au serveur Tyracoon RP.

Fonctionnalités principales :

- `/link <pseudo_minecraft>` — whitelist automatique via RCON
- Modération (kicks, bans, logs)
- Annonces automatiques (redémarrage serveur, nouvelles quêtes, etc.)
- Lien Discord ↔ UUID Minecraft pour la traçabilité des sanctions

---

Système de guildes et relations

Intégration FTB Teams

FTB Teams gère les groupes de joueurs. Le mod custom ajoute les **relations entre guildes** par dessus via l'API FTB Teams :

```
TeamAPI.INSTANCE.getTeamOf(player);
TeamAPI.INSTANCE.areInSameTeam(player1, player2);
```

Règles PvP

Relation	Hors claim	Dans claim	Zone centrale
Neutre	✅ PvP autorisé	❌ Protégé	❌ Protégé

Relation	Hors claim	Dans claim	Zone centrale
En guerre	⚔️ PvP autorisé	⚔️ PvP autorisé	🛡️ Protégé
Alliés	🚫 PvP bloqué	🚫 PvP bloqué	🛡️ Protégé

Le grief reste interdit dans tous les cas, y compris en guerre.

Déclaration de guerre

- Les **deux guildes doivent se déclarer mutuellement** — pas de guerre unilatérale
- **Délai de 24h** après la double déclaration avant activation — laisse le temps de se préparer
- **Seul le leader** peut déclarer guerre, alliance ou paix
- **Paix et alliance** nécessitent l'accord des deux parties

Commandes joueurs

```
/guild war <nom_guilde>           → déclarer la guerre
/guild ally <nom_guilde>          → proposer une alliance
/guild ally accept <nom_guilde>   → accepter une alliance
/guild peace <nom_guilde>         → demander la paix
/guild relations                  → voir ses relations actuelles
```

Schéma SQL

```
CREATE TABLE guild_relations (
    guild_id_a      INT NOT NULL,
    guild_id_b      INT NOT NULL,
    relation        ENUM('neutre', 'allié', 'en_guerre') DEFAULT 'neutre',
    declared_by_a   BOOLEAN DEFAULT FALSE,
    declared_by_b   BOOLEAN DEFAULT FALSE,
    active_since    TIMESTAMP,   -- NULL tant que pas mutuellement déclarée + délai écoulé
    declared_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (guild_id_a, guild_id_b),
    FOREIGN KEY (guild_id_a) REFERENCES guilds(id),
    FOREIGN KEY (guild_id_b) REFERENCES guilds(id)
);
```

Modèle économique — Dons communautaires

Philosophie

Accès totalement gratuit et égal pour tous les joueurs — aucun avantage gameplay monnayable. Les dons servent uniquement à couvrir les coûts d'infrastructure.

Transparence des coûts

Poste	Coût estimé/mois
Serveur dédié	~60-80 €
VPS (bot + VPN + Zabbix + Bookstack)	~5-10 €
Object Storage OVH (backups)	~1 €
Domaine	~1 €
Total	**~70-90 €/mois**

Les dons sont plafonnés aux coûts réels et affichés publiquement — aucun bénéfice personnel.

Plateformes recommandées

- **Hello Asso** — gratuit, sans commission, propre légalement pour les projets communautaires en France
- **Tipeee** — alternatif, bien connu de la communauté gaming FR
- **Ko-fi** — si besoin d'une audience internationale

Ce que les donateurs reçoivent (cosmétique uniquement)

- Rôle Discord dédié (cosmétique)
- Titre affiché en jeu via FTB Ranks (cosmétique, zéro avantage gameplay)
- Accès à un channel Discord "soutiens"
- Nom affiché dans un mur des soutiens (Discord + livre Patchouli in-game)

Point légal

En France, les dons reçus par un particulier sont techniquement imposables au-delà d'un certain seuil. Pour rester dans les clous :

- Plafonner les dons aux coûts réels d'infrastructure
- Publier une transparence mensuelle des recettes et dépenses
- Ne jamais dépasser les frais réels — pas d'enrichissement personnel

---

Communication et recrutement

Plateformes de référencement

- **Serveurs-Minecraft.org** — annuaire FR de référence
- **Minecraft-server-list.com** — audience internationale
- **TopG.org** — multi-jeux, bonne visibilité Minecraft
- **Serveurs Discord Minecraft FR** — annonces de recrutement

Réseaux sociaux

- **TikTok / YouTube Shorts** — timelapse de la capitale, présentation des classes, format court très efficace pour Minecraft en ce moment
- **Reddit** — r/feedthebeast (modpacks), r/MinecraftServer (recrutement), r/admincraft (visibilité communauté admin)
- **Twitter/X** — communauté Minecraft FR active avec les bons hashtags

Créateurs de contenu

Levier le plus puissant. Cibler des créateurs **mid-tier (5k-50k abonnés)** spécialisés Minecraft moddé — plus accessibles que les gros, audience plus qualifiée. Un seul streamer/youtubeur peut ramener 50 joueurs d'un coup.

Construction de la communauté

Avant le lancement

- Ouvrir le Discord en avance — teasing screenshots de la capitale en construction, présentation des classes une par une
- Programme bêta (10-20 joueurs) pour tester et créer du bouche-à-oreille

- Publier du lore et du storytelling sur les réseaux — crée de l'attachement avant même de jouer

Au lancement

- Date annoncée à l'avance avec compte à rebours sur Discord
- Événement d'ouverture in-game — cérémonie, quêtes d'introduction guidant vers la capitale
- Annoncer publiquement les premières guildes à obtenir un plot central — crée de l'émulation

Sur le long terme

- Événements réguliers — tournois PvP entre guildes en guerre, marchés éphémères, quêtes saisonnières
- Changelog public à chaque mise à jour sur Discord — montre que le projet est vivant
- Channel Discord dédié aux suggestions joueurs — les joueurs écoutés restent

Argument principal

La **capitale construite à la main** est un argument rare et différenciant. Des screenshots et vidéos de qualité de la capitale avant l'ouverture font une grande partie du travail de recrutement.

Licence

Les mods `tyracoonrp-client` et `tyracoonrp-server` sont la propriété exclusive de la Tyracoon RP Team.

Sélectionner **None** à la création du dépôt Gitea et ajouter manuellement un fichier `LICENSE.txt` à la racine du projet :

```
Copyright (c) 2026 Tyracoon RP Team
All Rights Reserved.

Unauthorized copying, distribution, or modification of this software
is strictly prohibited without prior written permission from Tyracoon RP Team.
```

Le dépôt doit être en **privé**. Aucune licence open source de la liste (MIT, GPL, Apache, etc.) ne convient — elles impliquent toutes une forme de partage ou redistribution du code.

Intégration chat externe

? Salut l'agent ! Briefing — Mod Matrix pour Minecraft 1.20.1 Forge

Ce document est un condensé de la conception réalisée avec un autre agent Claude. Bonne lecture et bon courage pour le dev ! □□

? Objectif du projet

Créer un **mod Forge 1.20.1** qui intègre un client **Matrix** directement dans Minecraft :

- Remplace/enrichit le chat vanilla MC avec des salons Matrix
- Interface GUI dédiée (keybind `M`) pour switcher de salon
- Le salon Matrix actif s'affiche dans le chat vanilla dynamiquement
- HUD minimaliste indiquant le salon actif (ex: `[#général]`)

Le mod sera **publié sur CurseForge**, usage sur un serveur Minecraft privé (whitelist).

? Fonctionnalités retenues

Feature	Statut
Authentification Matrix (token)	□ À implémenter
Sync temps réel des rooms	□ À implémenter
Affichage messages dans chat MC	□ À implémenter
Envoi de messages	□ À implémenter
Switch de salon via GUI	□ À implémenter
Filtrage des rooms par Space Matrix	□ À implémenter

Feature	Statut
Notifications in-game (toast/HUD)	□ À implémenter
VoIP / Appels	□ Hors scope
Envoi de fichiers/images	□ Hors scope

?? Architecture globale

Points clés de sécurité

- Le **token d'enregistrement Synapse n'est jamais dans le mod** (mod public sur CurseForge)
- La création de compte Matrix est déléguée au **bot Discord** (point d'entrée unique)
- Le bot Discord détient seul les droits admin Synapse

?? Serveur Matrix Synapse

- **Hébergement** : VPS Ubuntu 22.04/24.04 dédié (4 cœurs, 8 Go RAM, 75 Go NVMe)
- **Domaine** : `matrix.protosrv.fr`
- **Reverse proxy** : Nginx + Let's Encrypt (Certbot)
- **Base de données** : PostgreSQL (pas SQLite, anticipation de la charge)
- **Fédération** : désactivée (serveur privé, port 8448 non exposé)

- **Enregistrement** : `registration_requires_token: true` — tokens gérés par le bot Discord
- **Synapse Admin** : déployé dans Docker, accessible sur `https://matrix.protosrv.fr:8443` (accès restreint par IP)
- **Compte admin** : `@protomehka-admin:matrix.protosrv.fr` — seul compte admin du serveur

Config homeserver.yaml retenue

```
pid_file: "/var/run/matrix-synapse.pid"
server_name: "matrix.mondomaine.fr"

listeners:
  - bind_addresses:
      - "127.0.0.1"
    port: 8008
    type: http
    tls: false
    x_forwarded: true
    resources:
      - names: [client]
        compress: false

database:
  name: psycopg2
  args:
    user: synapse_user
    password: CHANGEME
    dbname: synapse
    host: localhost
    cp_min: 5
    cp_max: 10

log_config: "/etc/matrix-synapse/log.yaml"
signing_key_path: "/etc/matrix-synapse/homeserver.signing.key"
media_store_path: /var/lib/matrix-synapse/media
max_upload_size: 1M

federation_domain_whitelist: []
allow_public_rooms_over_federation: false
allow_public_rooms_without_auth: false
```

```
enable_registration: true
enable_registration_without_verification: true
registration_requires_token: true

trusted_key_servers: []
suppress_key_server_warning: true

event_cache_size: 10K
caches:
  global_factor: 1.0

enable_metrics: false
report_stats: false
user_directory:
  enabled: false
url_preview_enabled: false

rc_message:
  per_second: 0.5
  burst_count: 10
rc_login:
  address:
    per_second: 0.1
    burst_count: 3
  account:
    per_second: 0.1
    burst_count: 3
  failed_attempts:
    per_second: 0.1
    burst_count: 3
```

? Bot Discord — Rôle et responsabilités

Le bot Discord est le **chef d'orchestre** du système :

1. Commande `/whitelist <pseudo_minecraft>` libre (accessible à tous)
2. Vérifie et ajoute le joueur à la whitelist Minecraft
3. Crée automatiquement un compte Matrix via l'API Admin Synapse :

```
POST /_synapse/admin/v2/users/@pseudo:matrix.protosrv.fr
Authorization: Bearer <access_token_admin>
```

4. Stocke la liaison `Discord ID ↔ UUID Minecraft ↔ compte Matrix`
5. Génère/fournit les credentials Matrix au mod au premier lancement

> ℹ Le token d'accès admin s'obtient via :
>
> ```
> curl -X POST "https://matrix.protosrv.fr/_matrix/client/v3/login" \
>   -H "Content-Type: application/json" \
>   -d '{"type": "m.login.password", "user": "protomehka-admin", "password": "***"}'
> ```
>
> La réponse contient le champ `access_token` à stocker dans les variables d'environnement du bot.

> ℹ Le bot est déjà en cours de développement dans une autre session avec ProtoMehka — **à coordonner** pour l'ajout de la partie création de compte Matrix.

? Authentification du joueur dans le mod

Méthode retenue : Option A — Mot de passe envoyé par MP Discord

Le bot génère un mot de passe aléatoire à la création du compte Matrix et l'envoie en MP Discord au joueur. Le mod construit automatiquement l'identifiant Matrix à partir du pseudo Minecraft, le joueur n'a qu'à saisir son mot de passe au premier lancement.

Flow d'authentification :

```
1. Mod détecte qu'aucun token n'est stocké localement
2. Écran de config s'ouvre automatiquement :

   ┌──────────────────────────────────────┐
   │ Bienvenue ProtoMehka !               │
   │                                      │
```

```
   | Mot de passe Matrix :               |
   | [_____________________________]     |
   |                                     |
   |           [Se connecter]            |
   └─────────────────────────────────────┘
```

3. Le mod récupère le pseudo MC automatiquement :
   `String mcUsername = Minecraft.getInstance().getUser().getName();`
   `String matrixUserId = "@" + mcUsername.toLowerCase() + ":matrix.protosrv.fr";`
4. Appel `POST /_matrix/client/v3/login` avec userId + password
5. Token d'accès Matrix stocké localement → plus jamais redemandé

MP Discord envoyé par le bot :

```
 Tu es whitelisté sur ProtoSRV !

Ton mot de passe Matrix : xK9#mP2$nL
Entre-le dans Minecraft au premier lancement du mod.
```

> **i** Le pseudo Minecraft = le pseudo Matrix (sans le @...protosrv.fr). Le bot doit créer le compte Matrix avec exactement le même username que le pseudo Minecraft du joueur.

Le mod est configuré pour utiliser un **Space Matrix dédié** (l'équivalent d'un serveur Discord) plutôt que d'afficher toutes les rooms du compte joueur. Seules les rooms appartenant au Space du serveur Minecraft sont visibles dans le mod.

Structure du Space :

```
Space: "Mon Serveur Minecraft"
├── #général          — chat ouvert à tous
├── #annonces         — read-only pour les joueurs
├── #aide             — support
├── #off-topic        — détente
└── #staff            — privé, réservé aux admins
```

Avantages :

- Les DM perso ou autres rooms Matrix du joueur ne se mélangent pas dans l'interface MC
- Ajouter un nouveau salon dans le Space côté Synapse le fait apparaître automatiquement dans le mod pour tous les joueurs, sans mise à jour du mod
- Rooms publiques et privées gérées nativement par Matrix

Config du mod :

```
{
  "homeserver": "matrix.protosrv.fr",
  "space_id": "!xxxxx:matrix.protosrv.fr"
}
```

> ℹ Le Space et ses rooms sont à précréer sur Synapse avant le premier lancement du mod. Les rooms privées (ex: #staff) ne seront visibles que pour les joueurs qui y ont été invités.

```
Forge 1.20.1
├── OkHttp          — HTTP + WebSocket (connexion Matrix /sync)
├── Gson            — parsing JSON des events Matrix
├── GUI Minecraft   — Screen natif Forge pour l'interface
└── Thread dédié    — sync Matrix hors thread principal MC
```

Dépendances Gradle à ajouter

```
dependencies {
    implementation fg.deobf("...")  // Forge standard

    // OkHttp
    implementation 'com.squareup.okhttp3:okhttp:4.12.0'
    // Gson (déjà présent dans MC, vérifier version)
    implementation 'com.google.code.gson:gson:2.10.1'
}
```

> ⚠ Attention aux conflits de dépendances avec le classpath Forge/MC — shading d'OkHttp probablement nécessaire.

? Architecture des packages suggérée

```
fr.protomehka.matrixmod/
├── MatrixMod.java              — Entry point Forge
├── config/
│   └── MatrixConfig.java       — Homeserver URL, credentials stockés
├── matrix/
│   ├── MatrixClient.java       — Gestion auth + /sync polling
│   ├── MatrixRoom.java         — Modèle d'une room
│   └── MatrixMessage.java      — Modèle d'un message
├── gui/
│   ├── MatrixScreen.java       — GUI principal (liste rooms + messages)
│   └── RoomListWidget.java     — Widget liste des salons
├── chat/
│   ├── ChatInterceptor.java    — Intercepte envoi/réception chat MC
│   └── MatrixChatRenderer.java — Injection messages Matrix dans chat vanilla
├── events/
│   └── KeyBindHandler.java     — Keybind M pour ouvrir le GUI
└── notification/
    └── ToastNotifier.java      — Notifications toast in-game
```

? Flow utilisateur complet

1. Joueur fait /whitelist sur Discord
2. Bot crée compte Matrix (@pseudoMC:matrix.protosrv.fr) + génère mot de passe aléatoire
3. Bot envoie le mot de passe en MP Discord au joueur
4. Bot invite le joueur au Space Matrix
5. Joueur rejoint le serveur Minecraft
6. Mod détecte qu'aucun token n'est stocké → écran de saisie mot de passe
7. Joueur entre son mot de passe → mod fait /login → token stocké localement
8. MatrixClient se connecte, thread de sync démarre
9. Messages du salon actif s'affichent dans le chat MC
10. Joueur tape dans le chat → envoyé dans le salon Matrix actif
11. Appui sur M → GUI s'ouvre → switch de salon possible
12. Notification toast si message reçu dans un autre salon

? Points ouverts à définir

☑ ~~Comment le mod récupère les credentials au premier lancement ?~~ → Mot de passe envoyé par MP Discord, pseudo MC = pseudo Matrix

☐ Chiffrement E2EE ? (complexifie, via libolm — optionnel)

☐ Rooms Matrix précréées (recommandé) ou créées à la volée ?

☐ Le Space ID est-il hardcodé dans le mod ou configurable par fichier ?

☐ Gestion des rooms privées (DM entre joueurs) ?

Briefing généré depuis une session de conception avec Claude — Mars 2026