**Documentatie Gabazon**

**Parti:**

1. **Modul de Minecraft**
   1. **Tehnologii utilizate**
   2. **Modulul client**
   3. **Modulul generator de date**
   4. **Modulul de blocuri**
2. **Serverul de baza de date**
3. **Modul de Minecraft**

**I.a) Tehnologii utilizate**

Modul este structurat sub forma unui proiect Gradle, care vine la inceput cu cateva modificari, in principal in build.gradle pentru a elibera programatorul de nevoia de a face manual tweak-uri.

Principala tehnologie utilizata este Minecraft Forge, care este un framework ce permite conectarea la diferite bus-uri ale jocului pentru a introduce noi date legate de lume, entitati, comenzi, blocuri etc. intr-un mod mult mai simplu. Acesta pune la dispozitie atat metode si structure de date proprii pentru a realiza lucrul acesta, dar, de curand, si metode folosite de cei de la Mojang pentru a oferi o flexibilitate cat mai mare in timpul programarii.

La final ar fi pachetul de socketuri oferit de Java, prin care se realizeaza comunicarea cu modulul server.

**I.b) Modulul client** (GabazonClient.java)

Modulul client, asa cum am mentionat si mai sus, foloseste socketurile din Java si are o structura extrem de simpla deoarece utilizatorul nu ajunge niciodata sa-l foloseasca in mod direct, ci este apelat doar de un singur bloc atunci cand se plaseaza o comanda. Acesta trimite catre server un string care reprezinta comanda si un integer care reprezinta bugetul utilizatorului si intoarce un string care contine raspunsul de la server, urmand ca acesta sa fie prelucrat de catre blocul care a facut apelul.

Nu prea putem sa prelucram raspunsul in partea asta deoarece trebuie sa generam o noua entitate de tip carte scrisa pe care sa i-o oferim utilizatorului.

**I.c) Modulul generator de date** (Pachetul data)

Modulul generator de date este unul care are o importanta ceva mai mica in cazul meu din cauza marimii mici a modului. Acesta permite crearea diferitelor proprietati:loottables,tags,retete etc. sub forma de cod, care vor fi transformate ulterior in fisiere de tip json din care Minecraft-ul poate sa citeasca. Acest modul este foarte important in cazul modurilor mari deoarece scapa programatorul de nevoia de a scrie de mana sute de fisiere json.

Modulul acesta poate fi rulat independent folosind task-ul de runData configurat in build.gradle.

**I.d) Modulul de blocuri** (Pachetele block, setup, tools)

Crearea blocurilor se face in 2 etape: definirea claselor si inregistrarea lor. Pentru blocuri simple, care nu au functionalitate, se poate sari peste prima etapa si sa fie inregistrate sub forma de blocuri normale.

Un bloc poate avea de la o singura componenta (in cazul blocurilor simple), pana la 4 (cele care ofera si o interfata grafica pentru user). Este clasa block, care poate sa aiba un tile entity (sau un block entity din ultimele versiuni), acesta ocupandu-se de salvarea proprietatilor mai complexe prin care se da functionalitate unui bloc. In cazul meu, prin aceste entitati am reusit sa modelez comportamentul energiei, utilizarii ei, ce iteme accepta fiecare bloc si ce sa faca atunci cand primesc un item valid.

In clasa block am facut cateva verificari in functie de bloc deoarece diferite iteme au diferite comportamente in functie de itemul cu care au fost atinse. Monitorul atunci cand este atins cu un portofel va returna de la server o lista cu toate itemele, cantitatile si preturile (aceasta este singura metoda prin care aplicatia face o comanda de tip “list”, ceea ce imi permite sa nu pun prea multe verificari pe partea de server), statia cand este atinsa cu portofelul permite adaugarea de fonduri pentru cumparaturi. Cheia poate fi folosita pentru a inchide generatorul sau pentru a obtine o carte cu informatii de la statie. Ampermetrul permite verificarea nivelelor de energie a tuturor blocurilor. Aceste verificari puteau (si trebuiau cel mai probabil) sa fie facute in iteme, doar ca pe versiunea de 1.16.5 (pentru care am scris modul) multe metode au fost schimbate, ceea ce a incetinit un pic procesul de programare, si nu am apucat sa extind conceptul de item.

Blocurile nu pot sa comunice direct cu entitatile lor, ci fac lucrul acesta prin intermediul lumii sau prin intermediul unui container, care este cea de a 3-a componenta. Container-ul pregateste tot ce are nevoie screen-ul, ultima componenta, si contine informatii referitoare la inventarul jucatorului, lumea curenta si o referinta catre tile entity.

La baza am facut cate o clasa abstracta pentru fiecare din aceste componente si apoi le-am derivat in subpachetele blocurilor, suprascriind in functie de situatie metodele sau schimband valorile campurilor.

De inregistrare se ocupa pachetul setup, unde imi fac cate o copie a fiecarui bloc si item, cu scopul de a le adauga intr-un container specific pentru clasa, impreuna cu domeniul in care se afla (id-ul modului), urmand apoi sa fie adaugate in bus-ul de incarcare a jocului prin intermediul Forge-ului.

1. **Serverul de date**

Ca tehnologii, serverul de date concurent combina socketurile din java impreuna cu pachetul jdbc pentru a putea utiliza o baza de date in care sa stochez itemele.

Am ales sa folosesc jdbc deoarece avand o singura tabela de la care trebuie sa accesez date, este mult mai usor doar sa am o clasa DAO prin intermediul careia iau datele cand am nevoie, iar de asemenea, a fost mult mai usor de integrat cu proiectul.

La fel ca modulul client din mod, partea de server este si ea relativ simplificata deoarece intr-o anumita masura stim cum se desfasoara interactiunea. Nu avem nevoie sa punem un timeout deoarece in momentul in care clientul se conecteaza trimite si comanda, output-ul este prelucrat de unul din cele 3 cazuri, trimis, iar conexiunea este incheiata.

Comanda de listare este destul de straightforward, se face un apel catre baza de date prin intermediul clasei ItemDAO de forma “SELECT \* FROM items” de la care primim o lista cu toate itemele din server, pe care o prelucram sub forma de string pe care il intoarcem. String-ul acesta este procesat in client si transformat intr-o carte pe care jucatorul poate s-o consulte pentru nume, stoc sau preturi.

Comanda care are ceva mai multe verificari este cea de cumparare deoarece aici pot interveni destul de multe erori din partea utilizatorului, de exemplu: scrierea unui nume de item gresit, utilizarea delimitatorului incorect (ceea ce ar duce la o exceptie de tip IndexOutOfBounds) sau utilizarea unui simbol ce nu poate fi tradus intr-un integer. Aici verificam cu fiecare item daca user-ul isi permite sa fie adaugat item-ul in cos, iar daca nu, iesim din bucla si intoarcem doar cosul alcatuit pana in momentul respectiv. Daca nu s-a adaugat niciun item, ii intoarcem jucatorului un mesaj prin care il anuntam ca nu are suficienti bani. Dupa ce ne asiguram ca un item poate sa fie cumparat, ii facem un update in baza de date, pentru a ne asigura ca nu sunt cumparate mai multe lucruri decat avem.

Dupa ce ii intoarcem jucatorului raspunsul, inchidem conexiunea cu acesta si ne ocupam de ceilalti client daca sunt, iar daca nu, asteptam sa vina altul.