Chciałbym przygotować system kolejkowy do zapisu budowli, które crawler wykona. Chcę to zapisywać w bazie danych dla kazdego miasta budynki jakie maja byc w budowie. Juz mam przygotowany entity "villages"

```
import { Entity, Column, PrimaryColumn, CreateDateColumn, UpdateDateColumn } from 'typeor

@Entity('villages')
export class VillageEntity {
    @PrimaryColumn()
    id: number; // Game ID, not auto-generated

    @Column()
    name: string;

    @Column()
    coordinates: string; // Format: "XXX|XXX"

    @Column({ default: false })
    isAutoBuildEnabled: boolean;

    @Column({ default: false })
    isAutoScavengingEnabled: boolean;

    @CreateDateColumn()
    createdAt: Date;

    @UpdateDateColumn()
    updatedAt: Date;
}
```

Teraz chciałbym przygotować coś takiego, że mogę dodać budynek do kolejki

Dodawanie budynku do kolejki

1. Algorytm sprawdza czy istnieje wioska, korzystając z metody "findByName" z "villages.service", jezeli nie istnieje to za pomoca loggera wyswietla blad.
2. Jezeli wioska istnieje sprawdza, czy dany budynek moze byc wybudowany, to znaczy ze sprawdza za pomoca scrappera czy w UI budynek jest wybudowany o poziom nizej niz uzytkownik chce teraz i porownuje (to powinna być osobna metoda, bo moze tutaj dojsc wiecej warunków). Dodatkowo ze stałej "TRIBAL_WARS_BUILDINGS" sprawdzac budynek na ktory poziom chcemy wybudować nie jest wyższy niz maxLevel ustawiony w tej stałej
3. Jezeli jest wszystko poprawnie algorytm dodaje budynek do bazy danych (nie przejmuj się tym jak potem te budynki beda odczytywane, zeby realnie coś zbudować, na razie tylko zapisujemy liste w bazie)
4. W bazie powinno być zapisywane takie pola jak:

```
    @IsString()
    @IsNotEmpty()
    buildingId: string; -> ze stałej TRIBAL_WARS_BUILDINGS pole "id"

    @IsString()
    @IsNotEmpty()
    buildingId: string; -> ze stałej TRIBAL_WARS_BUILDINGS pole "name"

    /**
     * Target level to upgrade the building to
     * @example 3
     */
    @IsInt()
    @Min(1)
    level: number;

    /**
     * Village ID or village name where to build
     * @example '12345' or 'Moja Wioska'
     */
    @IsString()
    @IsNotEmpty()
    village: string; -> powiazanie kluczem z tabeli "VillageEntity"
```

Przygotuj proszę dobry plik prd.md pod to co opisałem. Chcę to przekazać innemu modelowi, który zajmie się implementacją. Napisz na koncu, bo to bedzie wiadomosć do bota, że jest w trybie "ask" i zrobimy 2 rundy pytań jakie ma do mnie w celu wyjaśnienia szczegółów

Jasne, oto propozycja pliku `prd.md` (Product Requirements Document) opisującego system kolejki budowy. Dokument jest przygotowany w sposób, który może być bezpośrednio przekazany do modelu AI w celu implementacji.

PRD: System kolejki budowy budynków

1. Przegląd

Celem jest stworzenie systemu kolejkowania zadań budowy dla bota gry "Plemiona". System umożliwi dodawanie zleceń budowy lub rozbudowy budynków w określonych wioskach. Zlecenia będą zapisywane w bazie danych w celu ich późniejszego przetwarzania przez moduł wykonawczy.

Ten dokument opisuje logikę i wymagania dotyczące **dodawania nowego zlecenia budowy do kolejki**.

2. Wymagania funkcjonalne

2.1. Nowa encja w bazie danych: `BuildQueueEntity`

Należy stworzyć nową tabelę w bazie danych do przechowywania zleceń budowy.

Nazwa tabeli: `build_queue`

Definicja encji (TypeORM):

```
import { Entity, Column, PrimaryGeneratedColumn, CreateDateColumn, UpdateDateColumn, Many
import { VillageEntity } from './village.entity';

export enum BuildStatus {
    QUEUED = 'queued',
    IN_PROGRESS = 'in_progress',
    COMPLETED = 'completed',
    FAILED = 'failed',
}

@Entity('build_queue')
export class BuildQueueEntity {
    @PrimaryGeneratedColumn('uuid')
    id: string;

    @Column()
    buildingId: string; // Identyfikator budynku ze stałej TRIBAL_WARS_BUILDINGS, np. 'ma

    @Column()
    buildingName: string; // Nazwa budynku ze stałej TRIBAL_WARS_BUILDINGS, np. 'Ratusz'

    @Column()
    level: number; // Docelowy poziom do rozbudowy

    @Column({
        type: 'enum',
        enum: BuildStatus,
        default: BuildStatus.QUEUED,
    })
    status: BuildStatus;

    @ManyToOne(() => VillageEntity, village => village.id)
    village: VillageEntity; // Relacja do encji VillageEntity

    @CreateDateColumn()
    createdAt: Date;

    @UpdateDateColumn()
    updatedAt: Date;
}
```

2.2. Obiekt transferu danych (DTO) dla dodawania zlecenia

Należy zdefiniować DTO, które będzie używane do walidacji danych wejściowych.

Definicja DTO (np. dla NestJS):

```
import { IsString, IsNotEmpty, IsInt, Min } from 'class-validator';

export class AddToBuildQueueDto {
    /**
     * Identyfikator budynku z TRIBAL_WARS_BUILDINGS (pole "id")
     * @example 'main'
     */
    @IsString()
    @IsNotEmpty()
    buildingId: string;

    /**
     * Docelowy poziom rozbudowy budynku
     * @example 3
     */
    @IsInt()
    @Min(1)
    level: number;

    /**
     * Identyfikator (ID) wioski, w której ma nastąpić budowa
     * @example 12345
     */
    @IsInt()
    @IsNotEmpty()
    villageId: number;
}
```

2.3. Logika biznesowa serwisu

Należy zaimplementować metodę w nowym serwisie (np. `build-queue.service.ts`), która będzie realizować następujący algorytm:

Metoda: `addToQueue(dto: AddToBuildQueueDto): Promise<BuildQueueEntity>`

Algorytm:

1. **Weryfikacja wioski:**
    - Użyj metody `findOneById` (lub podobnej) z `villages.service`, aby sprawdzić, czy wioska o podanym `villageId` istnieje w bazie danych.
    - Jeśli wioska nie istnieje, zarejestruj błąd za pomocą loggera (np. `logger.error(...)`) i rzuć wyjątek (np. `NotFoundException`).
2. **Walidacja zlecenia budowy:**
    - Ta logika powinna zostać umieszczona w osobnej, prywatnej metodzie serwisu, np. `validateBuildOrder(buildingId, level, village)`.

- **Warunek 1: Sprawdzenie aktualnego poziomu budynku.**
  - Użyj modułu scrapującego, aby odczytać z interfejsu gry aktualny poziom budynku (`currentLevel`) o zadanym `buildingId` w danej wiosce.
  - Zlecenie jest prawidłowe tylko wtedy, gdy `currentLevel === level - 1`.
- **Warunek 2: Sprawdzenie maksymalnego poziomu.**
  - Odszukaj budynek w stałej `TRIBAL_WARS_BUILDINGS` na podstawie `buildingId`.
  - Sprawdź, czy docelowy `level` nie jest wyższy niż `maxLevel` zdefiniowany dla tego budynku w stałej.
- Jeśli którykolwiek z warunków walidacji nie zostanie spełniony, rzuć wyjątek (np. `BadRequestException` lub `ConflictException`) z odpowiednim komunikatem.

3. **Dodanie zlecenia do bazy danych:**
   - Jeśli walidacja zakończy się sukcesem, utwórz nową instancję `BuildQueueEntity`.
   - Wypełnij pola: `buildingId`, `buildingName` (pobrane ze stałej), `level` oraz `village` (powiązana encja wioski).
   - Zapisz nową encję w bazie danych.
   - Zwróć zapisaną encję.

3. Zależności i założenia

- `VillageEntity`: Istnieje encja `VillageEntity` i serwis `villages.service` z metodą do wyszukiwania wiosek.
- `TRIBAL_WARS_BUILDINGS`: Istnieje globalna stała (lub plik konfiguracyjny) o strukturze podobnej do poniższej:

```
export const TRIBAL_WARS_BUILDINGS = [
    { id: 'main', name: 'Ratusz', maxLevel: 30 },
    { id: 'barracks', name: 'Koszary', maxLevel: 25 },
    // ...inne budynki
];
```

- **Scrapper**: Dostępny jest moduł scrapujący, który potrafi odczytać z interfejsu gry aktualny poziom danego budynku w konkretnej wiosce.
- **Logger**: Aplikacja posiada skonfigurowany mechanizm logowania zdarzeń.

4. Co jest poza zakresem?

- Implementacja mechanizmu, który odczytuje zlecenia z tabeli `build_queue` i faktycznie je wykonuje w grze.
- Obsługa błędów wykonania zlecenia (np. brak surowców) i aktualizacja pola `status` na `FAILED`.
- Interfejs użytkownika do zarządzania kolejką.

Znajdujesz się w trybie "ask". Zanim zaczniesz implementację, przeprowadzimy dwie rundy pytań, aby wyjaśnić wszelkie szczegóły i wątpliwości. Proszę, zadaj swoje pytania.