**Dev Tycoon — ERD & System Architecture**

Backend: Laravel (API + Simulation Engine)  
Frontend: NextJS (React)  
DB: MySQL + Redis (cache, queue)  
Realtime: Laravel Pusher   
Queue: Redis Queue (Laravel Horizon)

**1. Mục tiêu tài liệu**

Tài liệu này mô tả **ERD (Entity Relationship Diagram)**, các bảng chính, quan hệ giữa chúng, cùng **luồng dữ liệu (sequence flow)** cho các hành động quan trọng (click → income, start project, hire employee, prestige), kiến trúc service-level, cron/queue design và các event broadcast cần có.

Mục tiêu: cung cấp blueprint để dev backend Laravel + frontend NextJS triển khai Phase 1–2 (Freelancer → Studio) với mở rộng cho Phase 3+.

**2. ERD (Mermaid)**

erDiagram

users ||--o{ companies : owns

users ||--o{ game\_states : has

companies ||--o{ employees : employs

companies ||--o{ projects : runs

users ||--o{ projects : creates

projects ||--o{ project\_tasks : contains

employees ||--o{ tasks : performs

companies ||--o{ products : builds

researches }o--|| companies : belongs\_to

market\_events ||--o{ companies : affects

leaderboards ||--o{ users : ranks

USERS {

bigint id PK

varchar name

varchar email

varchar password

int level

int prestige\_points

datetime last\_active

timestamps

}

COMPANIES {

bigint id PK

bigint user\_id FK

varchar name

int company\_level

decimal cash

decimal monthly\_revenue

decimal monthly\_costs

timestamps

}

GAME\_STATES {

bigint id PK

bigint user\_id FK

bigint company\_id FK NULL

decimal money

decimal click\_power

decimal auto\_income

int xp

int level

json upgrades

timestamps

}

EMPLOYEES {

bigint id PK

bigint company\_id FK

varchar name

varchar role

int productivity

int skill\_level

int salary

int energy

enum status

timestamps

}

PROJECTS {

bigint id PK

bigint company\_id FK

bigint user\_id FK

varchar title

text description

int difficulty

decimal reward

int progress

datetime started\_at

datetime deadline

enum status

timestamps

}

PROJECT\_TASKS {

bigint id PK

bigint project\_id FK

varchar title

int estimated\_hours

int progress

bigint assigned\_employee\_id FK NULL

timestamps

}

PRODUCTS {

bigint id PK

bigint company\_id FK

varchar name

varchar type

int active\_users

decimal revenue\_per\_tick

int version

timestamps

}

RESEARCHES {

bigint id PK

bigint company\_id FK

varchar tech\_name

int level

json bonuses

datetime started\_at

datetime completed\_at NULL

timestamps

}

MARKET\_EVENTS {

bigint id PK

varchar event\_type

text description

json effect

datetime start\_time

datetime end\_time

timestamps

}

LEADERBOARDS {

bigint id PK

bigint user\_id FK

int rank

decimal score

timestamps

}

**3. Key Relationships & Notes**

* users ↔ companies: Một user có thể sở hữu nhiều company (tương lai cho multi-company feature), mặc định phase 1 chỉ dùng 1 company.
* game\_states lưu trạng thái game tách biệt để dễ cache/snapshot/restore.
* companies chứa các thông tin kinh tế toàn cục (cash, revenue, costs).
* projects được chạy bên trong companies và có thể chứa nhiều project\_tasks để mô phỏng agile.
* employees liên kết với companies và có thể assigned task.
* researches là tech tree items.
* market\_events là nguồn phát event toàn cầu ảnh hưởng tới doanh thu/costs/skill/market.

**4. Sequence Flows (Mermaid)**

**4.1 Click → Income (user active)**

sequenceDiagram

participant Client

participant API (Laravel)

participant DB

Client->>API: POST /api/game/click {user\_id}

API->>DB: BEGIN TRANSACTION

API->>DB: UPDATE game\_states SET money = money + click\_power

API-->>DB: SAVE

API->>Client: 200 OK {newMoney, clickPower}

API->>Broadcast: Event IncomeUpdated(user, delta)

Broadcast-->>Client: IncomeUpdated via WebSocket

**4.2 Offline Income Calculation (login)**

sequenceDiagram

participant Client

participant API

participant DB

Client->>API: GET /api/game (user logs in)

API->>DB: SELECT last\_active, auto\_income

API->>DB: calculate secondsOffline = now - last\_active

API->>DB: money += secondsOffline \* auto\_income

API-->>DB: UPDATE game\_states (last\_active = now)

API->>Client: 200 OK {game\_state}

**4.3 Start Project → Project Progress (background)**

sequenceDiagram

participant Client

participant API

participant DB

participant QueueWorker

Client->>API: POST /api/projects/start {projectParams}

API->>DB: INSERT project(status: in\_progress)

API->>Queue: dispatch ProcessProjectProgress(project\_id)

Queue->>QueueWorker: pick job

QueueWorker->>DB: update project.progress += computedRate

alt project.completed

QueueWorker->>DB: project.status = completed

QueueWorker->>Broadcast: ProjectCompleted

end

**5. Jobs & Scheduler design**

* **Jobs (queued)**
  + CalculateIdleIncome (every minute) — iterate active game\_states or companies and compute income; write to DB and broadcast.
  + ProcessProjectProgress — triggered on project start and periodically by scheduler to progress tasks.
  + TriggerMarketEvent (every 5–15 min) — create global events affecting companies.
  + AIBehaviorTick — for NPC companies or AI assistant behaviors.
* **Scheduler**
  + Use app/Console/Kernel.php to schedule jobs. Use Laravel Horizon + Redis for visibility.
  + Granularity: every 1 minute for most updates; per-second client UI can animate but persisted changes done per-minute.

**Why per-minute?**

* Reduces DB writes, easier to scale. Client can show smoother per-second update using autoIncome/60.

**6. Realtime events (Broadcast list)**

Use Laravel Echo with laravel-websockets or Pusher.

* IncomeUpdated — payload: {userId, delta, money}
* ProjectUpdated — {projectId, progress, status}
* ProjectCompleted — {projectId, reward}
* EmployeeHired — {companyId, employee}
* EmployeeQuit — {companyId, employeeId}
* MarketEventTriggered — {eventId, effect}
* LeaderboardUpdated — {top: []}

**7. API Design (Important Endpoints)**

* POST /api/auth/register — create user
* POST /api/auth/login — login (JWT / Sanctum)
* GET /api/game — fetch full game state for dashboard
* POST /api/game/click — perform click (returns delta)
* POST /api/game/upgrade — buy upgrade
* POST /api/projects/start — start project
* POST /api/projects/:id/claim — claim completed project reward
* POST /api/employees/hire — hire employee
* POST /api/research/start — start research
* POST /api/prestige — prestige/reset
* GET /api/leaderboard — global leaderboard

**Auth:** Use Laravel Sanctum for SPA token or Passport for full OAuth. For NextJS, Sanctum with cookie-based SPA auth is smooth.

**8. Caching & Performance**

* **Cache heavy-read**: GET /api/game should be cached per user for short TTL (e.g. 5s) in Redis.
* **Atomic updates**: Use DB transactions for money updates; use optimistic locking where needed.
* **Shard/Partition**: As user base grows, partition game\_states and projects by user\_id mod N.
* **Use queues** for all heavy writes/compute (Horizon + Redis).
* **Rate limit**: throttle clicks per second server-side to avoid abuse.

**9. Migrations samples (pseudo-Laravel)**

**create\_game\_states\_table**

Schema::create('game\_states', function (Blueprint $table) {

$table->id();

$table->foreignId('user\_id')->constrained();

$table->foreignId('company\_id')->nullable()->constrained();

$table->decimal('money', 20, 2)->default(0);

$table->decimal('click\_power', 12, 2)->default(1);

$table->decimal('auto\_income', 12, 2)->default(0);

$table->integer('xp')->default(0);

$table->integer('level')->default(1);

$table->json('upgrades')->nullable();

$table->timestamp('last\_active')->nullable();

$table->timestamps();

});

**create\_projects\_table**

Schema::create('projects', function (Blueprint $table) {

$table->id();

$table->foreignId('company\_id')->constrained();

$table->foreignId('user\_id')->constrained();

$table->string('title');

$table->text('description')->nullable();

$table->integer('difficulty')->default(1);

$table->decimal('reward', 20, 2)->default(0);

$table->integer('progress')->default(0);

$table->enum('status', ['queued','in\_progress','completed','failed'])->default('queued');

$table->timestamps();

});

**10. Security & Anti-cheat**

* **Server-side authoritative**: Never trust client for money/calculation.
* **Rate-limiting**: clicks and upgrade requests.
* **HMAC-signed requests**: optional, to make botting harder.
* **Detect anomalies**: background job to flag impossible progress (huge money jumps) and soft-ban.

**11. Deployment & Scaling**

* **Initial**: single Laravel app + Redis + MySQL managed (e.g., DigitalOcean, Linode, or AWS Lightsail)
* **Mid scale**: Move to microservices: split economy-service, project-service, ai-service; use Kubernetes or ECS.
* **Realtime**: Use laravel-websockets or Pusher. Consider horizontal scaling with Redis pub/sub.
* **CI/CD**: GitHub Actions deploy to server(s); use worker pool for queues.

**12. Next steps (recommended immediate deliverables)**

1. Finalize DB schema for Phase 1 (GameState, Users, Projects, Employees).
2. Implement Laravel Sanctum auth + GET /api/game endpoint.
3. Implement POST /api/game/click with DB transaction + broadcast.
4. Create CalculateIdleIncome job and schedule it in Kernel (every minute).
5. Build NextJS dashboard to display game state and call click endpoint.

**13. Appendix — Useful Laravel packages**

* laravel/sanctum (auth)
* beyondcode/laravel-websockets (websockets)
* laravel/horizon (queue dashboard)
* spatie/laravel-permission (roles/permissions if needed)
* spatie/laravel-activitylog (audit)
* spatie/laravel-rate-limited-queue (throttle queues)

Nếu Thống muốn, tôi có thể tiếp tục:

* Viết migration đầy đủ + Eloquent models + sample controllers cho các endpoint quan trọng;
* Tạo repo skeleton Laravel + NextJS (boilerplate) có auth, click endpoint, cron job đơn giản;
* Hoặc phác họa UI mockup cho dashboard (NextJS + Tailwind).

Chọn tiếp: 1) Migrations + Models + Controllers skeleton hoặc 2) Repo skeleton (Laravel + NextJS) hoặc 3) UI mockup.