

Backend Development Hands-on Test

Objective

The purpose of this task is to evaluate:

- Your programming skills.
- Your ability to implement solutions effectively within a limited timeframe.

You don't need to complete all the requirements. Focus on demonstrating your strongest skills and areas of expertise.

Task Overview

This task requires you to implement a backend service and a battle engine. You are free to choose any programming language and framework; however, we strongly recommend using one you are proficient in. This will help you focus on showcasing your skills without being hindered by unfamiliar tools.

We recommend using **Redis** as your main database, but use whichever DB you're familiar with.

Goal:

Develop a backend service to handle player registration, leaderboards, and battle requests, and an engine to simulate and manage battles programmatically.

What We're Looking For

- Clean, maintainable, and efficient code.
 - Thoughtful design decisions and clear documentation.
 - Your ability to prioritize and focus on key features within the provided timeframe.
-

Backend Service Requirements

Database Modeling

Define and store the following data in the primary database:

1. Player:

- **Identifier** (Unique ID).
- **Name** (Max: 20 characters, unique).
- **Description** (Max: 1,000 characters).
- **Gold** (Max: 1 billion).
- **Silver** (Max: 1 billion).
- **Attack Value**, **Defense Value** and **Hit Points**.

Endpoints/Handlers

You need to implement the following endpoints:

1. **Create Player:**
 - Validate inputs and store player data in the database.
 2. **Submit Battle:**
 - Allow players to initiate battles by specifying an opponent.
 - Add battle requests to a processing queue.
 3. **Retrieve Leaderboard:**
 - Return a ranked list of players.
 - Each entry should include:
 - Rank/Position.
 - Score.
 - Player Identifier.
-

Battle Processor Requirements

Processor Functionality

1. **Process Battles:**
 - Handle battles in the order they are submitted.
 - Ensure no battle is processed more than once, and none are skipped.
 - Aim for immediate processing (no strict real-time requirement).
 2. **Battle Processing Steps:**
 - Execute the battle logic (see **Battle Engine Logic** for details).
 - Generate a detailed battle report of events and outcomes to be presented to the player.
 - Update player resources:
 - Deduct resources from the losing player.
 - Add resources to the winning player.
 - Submit the total resources stolen as score to the leaderboard.
-

Battle Engine Logic

Battle System Rules

1. **Turn-Based:**
 - The player who initiated the battle attacks first.
 - Roles switch after each turn.
2. **Hit or Miss:**
 - Use the defender's **Defense Value** to determine if an attack misses.
3. **Damage Calculation:**
 - **Damage** is equal to the attacker's current **Attack Value**.

- If the attacker successfully hits their opponent, the defender's hit points are reduced by the attacker's attack value.
- The attacker's **Attack Value** decreases as their health drops (minimum cap: 50% of the base value).
 - Example: If an attacker starts with 100 health and 70 attack, losing 10% of health reduces attack by 10%. Attack will never drop below 35, which is 50% of the base value.

4. Victory Condition:

- The battle ends when one player's **Hit Points** reach zero.
-

Resource Management

1. Resources Stolen:

- Winners steal **5–10%** of each resource type (e.g., gold, silver) individually from the losing player.
 - Resource values should always be rounded up to ensure fairness.
- Example:
 - If a player has **500 gold** and **120 silver**, and **7%** is stolen, the winner takes **35 gold** and **9 silver**.
- This ensures the total stolen is proportional across all resource types and falls within an overall range of **10–20%**.

2. Battle Report:

- Generate a detailed report of all actions, including damage, misses, outcomes and resources stolen.
-

Concurrency

- Support simultaneous processing of battles that don't involve overlapping players.
-

Security Requirements

- Protect all endpoints from unauthorized access.
-

Notes for Candidates

- Focus on clean, maintainable code.
- Provide documentation for:
 - Setup and running instructions.
 - Key design decisions.
- Write meaningful tests to verify functionality.
- Highlight any assumptions or trade-offs you made.