Neo Take-Home Assignment

Problem Statement

You work at a company that operates in the video game industry, and have recently joined the team responsible for bootstrapping new role-playing games. This team makes sure that the groundwork for new games is solid, so more people can contribute as new features are added.

Your team has been tasked with producing a proof-of-concept for a new game. It's your role on the team to **produce an API** for managing characters. Other members of the team will be working on the database and user interfaces, so you don't need to worry about those. Your deliverable is the backend system for the game functions detailed below, and basic documentation for accessing your endpoints.

Submission Requirements

- Store all state in memory - do not use a database.
- Include instructions on how to run your solution.
- Please keep your solution private.
- Please write unit tests where requested.

Submit your assignment via `git bundle`:

```
# Make sure everything is included
git add .
git commit -m "..."

# Create the bundle
git bundle create submission-<your name>.bundle master
```

Send this .bundle file to your Neo hiring contact.

Game Features

F1 - Character Creation

Requires unit tests

A job is a character class, dictating some specialities in battle. New characters have initial stats based on their selected job. These stats can grow during the game, however you are not expected to handle this in this assignment. Possible jobs will be shown to the player so they can make an informed choice.

Information needed to create a new character is `name` and `job`.

`name` must contain letters or `_` (underscore) characters and have a length between 4 and 15 characters inclusive. `job` must be one of `Warrior` `Thief` or `Mage`.

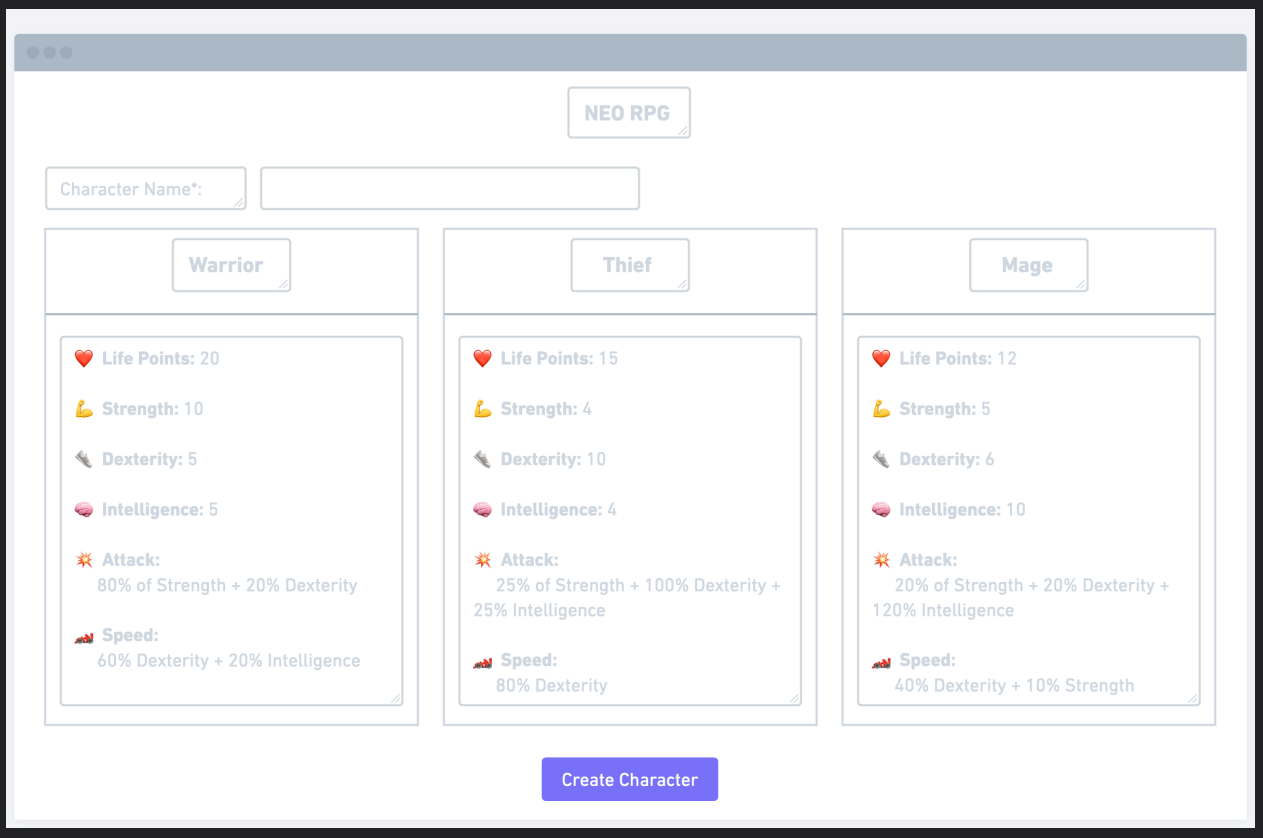
Design considerations

- A character can only have a single job at a time.
- A character should be designed with some future features in mind, which don't need to be implemented at this point in the project:
  - A character will be able to level up, at which point their core attributes will change (**health, strength, dexterity,** and **intelligence**)
  - A character will be able to change their job, resulting in calculations involving their modifiers to reflect their new job (**attack modifier** and **speed modifier**)

Currently available jobs

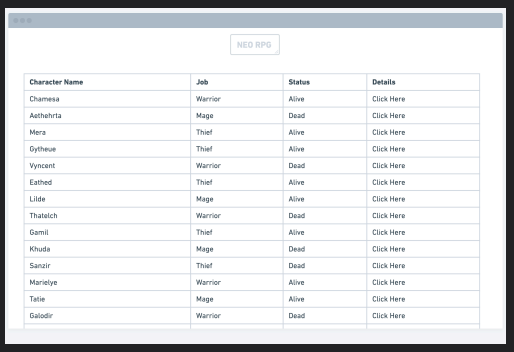
Job name	Health Points (HP)	Strength	Dexterity	Intelligence	Attack modifier	Speed modifier
Warrior	20	10	5	5	80% of strength + 20% of dexterity	60% of dexterity + 20% of intelligence
Thief	15	4	10	4	25% of strength + 100% of dexterity + 25% of intelligence	80% of dexterity
Mage	12	5	6	10	20% of strength + 20% of dexterity + 120% of intelligence	40% of dexterity + 10% of strength

F1 - User Interface



F2 - Character List

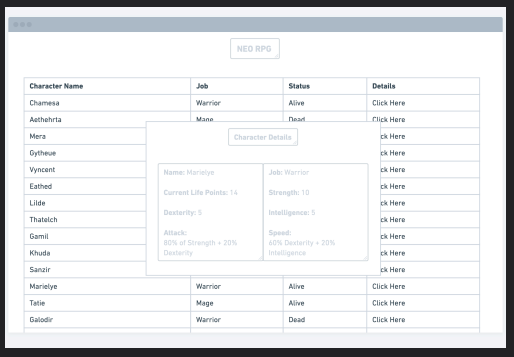
> *We want to have a way to show every character on the screen.*



Allow the player to see all characters by name and job, and if they are alive or dead.

F3 - Character Details

> *From the page on F2, we want to have a way to show the details of a selected character.*



Allow the player to see details about a specific character.

- Name
- Job
- Current life points
- Maximum life points
- Stats
- Battle modifiers

F4 - Battle

Requires unit tests

> *We need to create the algorithm that will control the battle between two characters. A battle is fought until one of the characters is dead.*

A battle is a series of rounds, with each round having 1 or 2 turns. Characters use their turn to attack the other. At the end of the battle, a battle log is printed along with the winner and loser. Here's how the battle works:

A round begins by determining which character goes first. To do this, generate a random integer between zero and each character's calculated speed modifier. The character with the highest speed value will take the first turn in the round. Repeat this process in the case of a draw. The battle log should only show the result that was not a draw.

A turn is comprised of one character attacking the other. To deal damage, generate a random integer between zero and the attacker's calculated attack modifier. Subtract the damage from the defender's HP stat. If the defender's health points reach 0, they are dead and the battle is over (health points cannot fall below 0). Otherwise, the defending character takes their turn.

The defeated character will appear dead in F2 and F3. The victorious character keeps their remaining HP after the battle.

Battle log

The battle log will have the following format:

```
Battle between CharacterX (<job>) - <HP> HP and CharacterY (<job>) - <HP> HP begins!
CharacterX <speed> speed was faster than CharacterY <speed> speed and will begin this round.
CharacterX attacks CharacterY for <damage>, CharacterY has <HP> HP remaining.
CharacterY attacks CharacterX for <damage>, CharacterX has <HP> HP remaining.
CharacterY <speed> speed was faster than CharacterX <speed> speed and will begin this round.
…
CharacterX wins the battle! CharacterX still has <HP> HP remaining!
```