BattlePets

Description

BattlePets is like a REST Tamagotchi, with the added benefit of being able to pit your creatures against each other in games of strength, agility, wit, etc. Through the API, a user may recruit a BattlePet from the wild, and volunteer it for participation in the BattlePets Arena.

Your challenge is to build the API that powers BattlePets. You don't need to be concerned with the frontend application. Also, it doesn't exist.

What I'm Concerned With

- your approach to architecting a multi-service backend system to power an API for consumption by an arbitrary client
- your approach to testing that API
- your approach to facilitating communication between backend services
- your approach to encapsulating logic
- your reasons and justifications for design decisions

Functionality Details

The actual features of this game API should be simple, and you have a lot of discretion over the various formulas and heuristics needed to build out the API. Below is some guidance on the particulars, but feel free to take some creative license.

Recruitment

When a user submits some set of attributes to the BattlePets API, the server manifests a creature and responds with an appropriate payload.

Pet Attributes

These are meant to be taken into account when executing a contest between two BattlePets in the BattlePets Arena. The minimum set can be four numerical values that map to the following properties:

1. Strength
2. Agility
3. Wit
4. Senses

You are welcome to adjust the attributes or make up your own.

Contests

Contests pit one BattlePet against another and evaluate them against a set of rules. There is always one winner and one loser in any contest. None of the contests are fatal, so upon completion, both creatures should earn some experience and be made immediately available for another contest. Experience and past contest results should be recorded.

Contest Evaluation

Contest evaluation should consist of a measure of two BattlePets based on prior experience and attributes. For instance, a contest of strength might consider two BattlePets and declare the creature with the higher strength attribute to be the victor. In the case that both BattlePets have equal strength, the evaluator might declare the more experienced BattlePet to be the winner.

The exact heuristic is up to you, but make sure the design intent is clear. This is a good place to add some unit tests to ensure that your contest evaluation performs as expected.

Experience

Experience should be a number that correlates to the number of contests a creature has completed. You might want to award different amounts of experience for victories and losses.

Contest Types

There should be multiple contest types. This could be as simple as having one contest that considers each attribute, or you could make things interesting and create several multivariate contests.

Implementation Details

Services

Organize the API however you think makes the most sense, but it should have at least two services running in different processes:

- BattlePets Management - a REST API to provision and manage BattlePets
- BattlePets Arena - a REST API to manage Contests

Asynchronous Workers

Contests should be considered resource-intensive operations, so they should be conducted asynchronously in the background with a job queue of your choosing. How to handle reporting asynchronous Contest results is up to you—be it via polling or another method of your choosing.

Programming Languages and Technologies

We use a lot of Ruby at Wunder, but I am more concerned with style, code quality, and thought process than I am with technology. That goes for languages, frameworks, infrastructure, and additional software components.

Ultimately, the choice is much less important than the justification and the quality of the work that comes out of it.

Testing

Here at Wunder, we use automated testing extensively, but we also take a pragmatic approach to testing. When appropriate, we use TDD, but it's not something that we do without consideration for the cost-benefit analysis. We would love to see thorough testing for the critical functions of your BattlePets infrastructure.

Bonus Functionality

- A leaderboard to see what BattlePets are on top
- A training interface to help your BattlePets get in shape before battle
- A history of past battle pets contests and the results
- Feel free to take some creative license if you think of ways to make BattlePets more awesome

Delivery / Execution

Please write a README with sufficient detail to get your BattlePets infrastructure up and running quickly in a development environment. You may want to consider putting together some seed data. Also include instructions for running the test suite.

Please use git for source control - you can push your code to github (or some other hosted git), or deliver it as an tarball / archive of some sort.

Finally, your codebase should include a script that we can run to see your BattlePets infrastructure in action!