Vessel Programming Challenge: Game Server

By: Ben Caine

This is a generic game server used to host arbitrary turn based text games. Games are played through a RESTful interface, with responses and requests sending JSON.

Overview

This game server uses the following technology:

- Ruby 2.0.0
- Sinatra 1.4.5
- DataMapper 1.2.0
- SQLite 3.7.13
- json (gem) 1.8.1

Installation

Ensure you have Ruby 2.0 installed and bundler.

If you don't have bundler, run `gem install bundler`

Then you can run `bundle install` which should install all dependencies.

File Structure

- Gemfile - Used by bundle to install gems
- dmconfig.rb - Used by DataMapper to set up the SQLite database
- game_server.rb - Main file of our service
- game_server.db - SQLite database file (may not exist yet)
- games/ - Directory of all of our games.
- helpers/ - Directory of our helper files
- models/ - Our permanent classes
- resources/ - Our RESTful interface

Running the service

To run the service, simply type `ruby game_server.rb`

This should launch a local server you can access at "localhost:4567"

Check that it works by doing `curl localhost:4567/` and ensuring it returns "Welcome to Ben's Game Server!"

API / Interface

Users

- GET /users/ returns a list of users (in JSON form)
- GET /users/:id/ returns a JSON representation of the user (including wins/losses)
- POST /users/ creates a new User and returns the id of that user.

Games

- GET /games/ returns a list of game types
- PUT /games/:token/ takes JSON, plays the game, returns the current state
- POST /games/:game_type/user/:user_id/ Starts a game of :game_type for user :user_id. Returns token ID to use in /games/:token/

Adding Games

Adding games is very straight forward. You simply need to create a new file (and class) in the games/ directory that includes "Game", with a property :id, and function play(json) which accepts JSON and returns a response hash.

```
require './models/game'
class YourGame
    include Game
    property :id, Serial
    property :game_field, Integer, :default => 0
    # More properties here if needed

    def play(json_data)
        # Code here that implements a turn in the game
        response
    end
end
```

Note: The response must have a :done => true, and a :won => true (or false) when the game is finished (see below)

```
{
    "won":true,
    "done":true,
    "response":"Some Message",
    "some_data": 0
}
```

Then you need to restart the server. On server start, a table will be automatically generated in SQLite and the game will be playable through the game interface.

Playing (Testing)

I provided a simple "game" called `guess_number` that you can use for testing the API's.

To start, we want to first create a user.

```
curl -X POST -d '' localhost:4567/users/

{"user_id":2}
```

We then can test that we can list every user.

```
curl localhost:4567/users/

["{\"id\":1,\"wins\":0,\"losses\":0,\"total\":0,\"created_at\":\"2014-10-
13T20:26:48-04:00\"}"]
```

We can also test that the /users/:id/ endpoint works.

```
curl localhost:4567/users/1/

{"id":1,"wins":0,"losses":0,"total":0,"created_at":"2014-10-13T20:26:48-04:00"}
```

Next, we want to check our game endpoints.

```
curl localhost:4567/games/

["guess_number","mock"]
```

We then want to start a game, which we do with a request to /games/:game_type/user/:user_id/ which returns a token number we will use next.

```
curl -X POST -d '' localhost:4567/games/guess_number/user/1/

{"game_token":1}
```

We can then use that game token number to play the game through the PUT /games/:token/ endpoint.

```
curl -X PUT -H "Content-Type: application/json" -d '{"guess": "5"}'
localhost:4567/games/1/

{"won":true,"done":true,"response":"Congrats, You won!","tries":1,"chances":10}
```

That's it for the API, but you can do some sanity checks.

Check that the users score is updated.

```
curl localhost:4567/users/

["{\"id\":1,\"wins\":1,\"losses\":0,\"total\":1,\"created_at\":\"2014-10-
13T22:52:53-04:00\"}"]
```

Or if you prefer

```
curl localhost:4567/users/1/

{"id":1,"wins":1,"losses":0,"total":1,"created_at":"2014-10-13T22:52:53-04:00"}
```