Part 3: Connecting Wordguesser to Sinatra

You've already met Sinatra. Here's what's new in the Sinatra app skeleton `app.rb` that we provide for Wordguesser:

- `before do...end` is a block of code executed *before* every SaaS request
- `after do...end` is executed *after* every SaaS request
- The calls `erb :` *action* cause Sinatra to look for the file `views/` *action* `.erb` and run them through the Embedded Ruby processor, which looks for constructions `<%= like this %>`, executes the Ruby code inside, and substitutes the result. The code is executed in the same context as the call to `erb`, so the code can "see" any instance variables set up in the `get` or `post` blocks.

Self Check Question

▶ `@game` in this context is an instance variable of what class? (Careful-- tricky!)

The Session

We've already identified the items necessary to maintain game state, and encapsulated them in the game class. Since HTTP is stateless, when a new HTTP request comes in, there is no notion of the "current game". What we need to do, therefore, is save the game object in some way between requests.

If the game object were large, we'd probably store it in a database on the server, and place an identifier to the correct database record into the cookie. (In fact, as we'll see, this is exactly what Rails apps do.) But since our game state is small, we can just put the whole thing in the cookie. Sinatra's `session` library lets us do this: in the context of the Sinatra app, anything we place into the special "magic" hash `session[]` is preserved across requests. In fact, objects placed there are *serialized* into a text-friendly form that is preserved for us. This behavior is switched on by the Sinatra call `enable :sessions` in `app.rb`.

There is one other session-like object we will use. In some cases above, one action will perform some state change and then redirect to another action, such as when the Guess action (triggered by `POST /guess`) redirects to the Show action (`GET /show`) to redisplay the game state after each guess. But what if the Guess action wants to display a message to the player, such as to inform them that they have erroneously repeated a guess? The problem is that since every request is stateless, we need to get that message "across" the redirect, just as we need to preserve game state "across" HTTP requests.

To do this, we use the `sinatra-flash` gem, which you can see in the Gemfile. `flash[]` is a hash for remembering short messages that persist until the *very next* request (usually a redirect), and are then erased.

Self Check Question

▶ Why does this save work compared to just storing those messages in the `session[]` hash?

Running the Sinatra app

As before, run the shell command `bundle exec rackup --port 3000` to start the app, or `bundle exec rerun -- rackup --port 3000 --host 0.0.0.0` if you want to rerun the app each time you make a code change.

Self Check Question

▶ Based on the output from running this command, what is the full URL you need to visit in order to visit the New Game page?

Visit this URL and verify that the Start New Game page appears.

Self Check Question

▶ Where is the HTML code for this page?

Verify that when you click the New Game button, you get an error. This is because we've deliberately left the `<form>` that encloses this button incomplete: we haven't specified where the form should post to. We'll do that next, but we'll do it in a test-driven way.

But first, let's get our app onto Heroku. This is actually a critical step. We need to ensure that our app will run on heroku **before** we start making significant changes.

- First, run `bundle install` to make sure our Gemfile and Gemfile.lock are in sync.
- Next, type `git add .` to stage all changed files (including Gemfile.lock)
- Then type `git commit -m "Ready for Heroku!"` to commit all local changes.
- Next, type `heroku login` and authenticate.
- Since this is the first time we're telling Heroku about the Wordguesser app, we must type `heroku create` to have Heroku prepare to recieve this code and to have it create a git reference for referencing the new remote repository.
- Then, type `git push heroku master` to push your code to Heroku.
- When you want to update Heroku later, you only need to commit your changes to git locally, then push to Heroku as in the last step.
- Verify that the Heroku-deployed Wordguesser behaves the same as your development version before continuing. A few lines up from the bottom of the Heroku output in the terminal should have a URL ending in herokuapp.com. Find that, copy it to the clipboard, and paste it into a browser tab to see the current app.
- Verify the broken functionality by clicking the new game button.

Next: Part 4 - Cucumber