

PRÁCTICA CALIFICADA 1 – CC3S2

Cesar Jesús Lara Avila

Calagua Mallqui Jairo Andre	20210279F
Chavez Chico Joel Jhotan	20210058J
Salcedo Alvarez Guillermo Ronie	20210164D

Facultad de Ciencias, Universidad Nacional de Ingeniería

Octubre del 2023

Introducción

En el ciclo completo Agile/XP que seguimos, incluimos hablar con el cliente, utilizar BDD para desarrollar escenarios y convertirlos en pruebas ejecutables de integración/aceptación con Cucumber. También empleamos TDD para impulsar la creación del código real y finalmente implementamos el resultado en la nube en cada iteración.

En esta tarea introductoria, nos proporcionan pruebas unitarias de RSpec para que podamos utilizar TDD y desarrollar la lógica de un juego de adivinanzas de palabras. En el ciclo completo de Agile/XP, yo mismo desarrollaría estas pruebas mientras codifico.

Luego, utilizaremos el framework de Sinatra para hacer que el juego Wordguesser esté disponible como SaaS. Adaptar la lógica del juego para SaaS nos permitirá pensar en rutas RESTful y en una arquitectura orientada a servicios. A medida que desarrollamos el juego Wordguessing, utilizaremos Cucumber para describir cómo funcionará el juego desde el punto de vista del jugador y como pruebas de integración "completas" que impulsarán el desarrollo de SaaS. En el ciclo Agile/XP completo, nosotros desarrollaremos escenarios de Cucumber basándonos en consultas con el cliente y crearemos las definiciones de pasos necesarias (código de Cucumber que convierte escenarios en inglés simple en pruebas ejecutables). En esta tarea, nos proporcionan tanto los escenarios como las definiciones de pasos.

Implementaremos nuestro juego en la nube utilizando Heroku, lo que nos brindará experiencia en la automatización de la implementación de SaaS.

En esta tarea, se utiliza Sinatra en lugar de Rails. Dado que nuestra aplicación no tiene una base de datos y tiene muy pocas funciones, Sinatra es una manera fácil de comenzar.

Creación de aplicaciones SaaS

Objetivo: comprender los pasos necesarios para crear, versionar e implementar una aplicación SaaS, incluido el seguimiento de las librerías de las que depende para que sus entornos de producción y desarrollo sean lo más similares posible.

¿Qué haremos?: crear una aplicación sencilla de "Hello world" utilizando el framework Sinatra, versionarla correctamente e implementarla en Heroku.

Creación y versionado de una aplicación SaaS sencilla

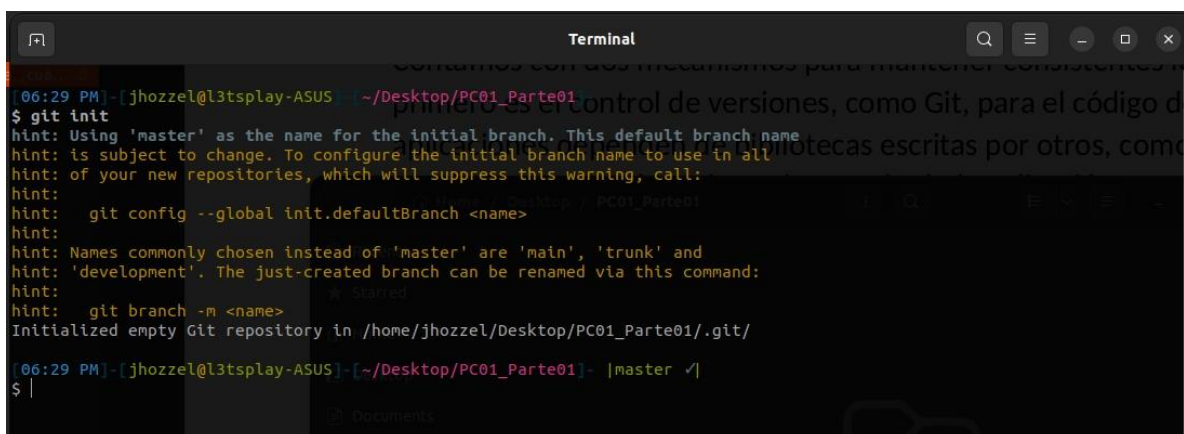
Las aplicaciones SaaS son desarrolladas en la computadora del desarrollador, pero se despliegan en un servidor al que otros pueden acceder. Se busca minimizar las discrepancias entre los entornos de desarrollo y producción para evitar problemas difíciles de diagnosticar, donde el código puede funcionar de una manera en el entorno de desarrollo personal, pero de manera diferente o incluso no funcionar en absoluto cuando se implementa en producción.

Existen dos mecanismos para mantener la consistencia entre los entornos de desarrollo y producción. El primero se basa en el control de versiones, como Git, para gestionar el código de la aplicación. Dado que muchas aplicaciones dependen de bibliotecas externas, como gems en el caso de Ruby, es necesario realizar un seguimiento de las versiones que han sido probadas y asegurarse de que se utilicen tanto en el desarrollo como en producción.

Afortunadamente, en Ruby, existe una herramienta muy útil para gestionar estas dependencias de gemas: Bundler. Bundler busca un archivo llamado Gemfile en el directorio raíz de cada proyecto de aplicación. Este Gemfile contiene una lista de gemas y sus respectivas versiones en las que la aplicación depende. Bundler se encarga de verificar que estas gemas, así como las que dependen de ellas, estén correctamente instaladas en el sistema y sean accesibles para la aplicación.

Comencemos con los siguientes pasos:

- Creamos un nuevo directorio vacío para contener nuestra nueva aplicación y usamos git init en ese directorio para comenzar a versionarlo con Git.



```
Terminal
[06:29 PM]-[jhozzel@l3tsplay-ASUS ~]-[~/Desktop/PC01_Parte01]
$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/jhozzel/Desktop/PC01_Parte01/.git/

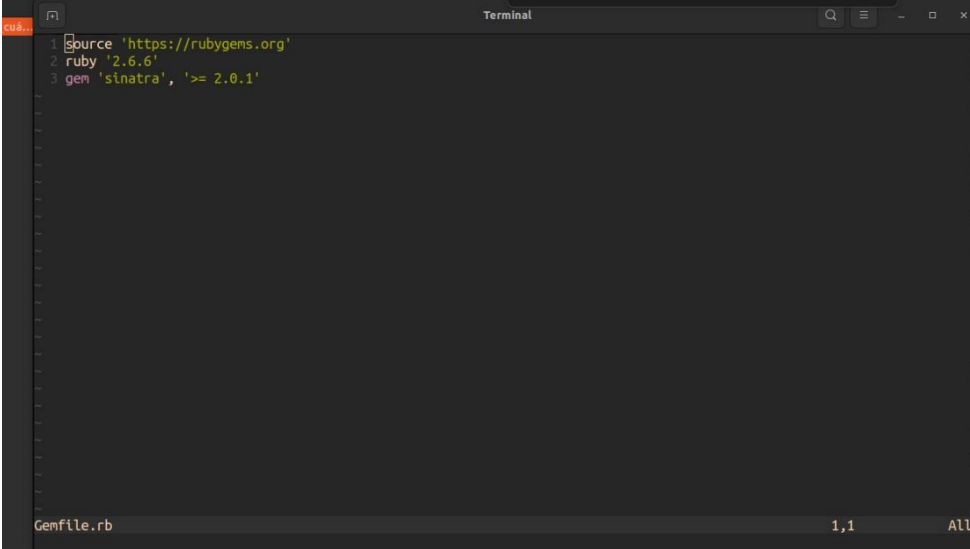
[06:29 PM]-[jhozzel@l3tsplay-ASUS ~]-[~/Desktop/PC01_Parte01]- |master ✓|
$ |
```

- En ese directorio, crea un nuevo archivo llamado Gemfile (las mayúsculas son importantes) con el siguiente contenido. Este archivo será una parte permanente de su aplicación y viajará con su aplicación a donde quiera que vaya:

```
source 'https://rubygems.org'
```

```
ruby '2.6.6'
```

```
gem 'sinatra', '>= 2.0.1'
```

A screenshot of a terminal window titled "Terminal". The window shows a file named "Gemfile.rb" with three lines of code: 1 source 'https://rubygems.org', 2 ruby '2.6.6', and 3 gem 'sinatra', '>= 2.0.1'. The terminal has a dark background with light-colored text. The status bar at the bottom shows "Gemfile.rb", "1,1", and "All".

```
1 source 'https://rubygems.org'
2 ruby '2.6.6'
3 gem 'sinatra', '>= 2.0.1'
```

La primera línea dice que el lugar preferido para descargar las gemas necesarias es <https://rubygems.org>, que es donde la comunidad Ruby registra las gemas "listas para producción".

La segunda línea especifica qué versión del intérprete de lenguaje Ruby se requiere. Si omitiéramos esta línea, Bundler no intentaría verificar qué versión de Ruby está disponible; Existen diferencias sutiles entre las versiones y no todas las gemas funcionan con todas las versiones, por lo que es mejor especificar esto.

La última línea dice que necesitamos la versión 2.0.1 o posterior de Sinatra. En algunos casos no necesitamos especificar qué versión de una gema queremos. En este caso lo especificamos porque nos basamos en algunas características que están ausentes en versiones anteriores de Sinatra.

Correr el Bundler

Ejecuta el comando bundle que examina tu Gemfile para asegurarse de que estén disponibles las gemas correctas (y, cuando se especifique, las versiones correctas), e intenta instalarlas de otra manera. Esto creará un nuevo archivo Gemfile.lock, que deberás colocar bajo control de versiones.

```
Terminal
Gemfile Gemfile.rb

[06:33 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master ??:2 X
$ rm Gemfile
Gemfile
Gemfile.rb

[06:33 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master ??:2 X
$ rm Gemfile.rb
Gemfile
Gemfile

[06:33 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master ??:1 X
$ ls
Gemfile

[06:33 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master ??:1 X
$ nvim Gemfile

[06:34 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master ??:1 X
$ nvim Gemfile

[06:34 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master ??:1 X
$ sudo bundle install
Don't run Bundler as root. Installing your bundle as root will break this application for all non-root users on this machine.
Fetching gem metadata from https://rubygems.org/...
Resolving dependencies...
Bundle complete! 1 Gemfile dependency, 7 gems now installed.
Use `bundle info [gemname]` to see where a bundled gem is installed.

[06:34 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master ??:2 X
$
```

Para colocarlo bajo control de versiones, usa estos comandos:

\$ git agregar.

\$ git commit -m "Configurar el Gemfile"

```
Terminal

add

[06:35 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master S:2 X
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Gemfile
        new file:   Gemfile.lock

[06:35 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master S:2 X
$ git commit -m "Configurar el gemfile"
git: 'commit' is not a git command. See 'git --help'.

The most similar command is
    commit

[06:35 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master S:2 X
$ git commit -m "Configurar el gemfile"
[master (root-commit) 4d91af0] Configurar el gemfile
 2 files changed, 26 insertions(+)
 create mode 100644 Gemfile
 create mode 100644 Gemfile.lock

[06:35 PM] jhozzel@l3tsplay-ASUS: ~/Desktop/PC01_Parte01 |master S:1 X
$
```

El primer comando presenta todos los archivos modificados para su confirmación. El segundo comando confirma los archivos preparados con el comentario entre comillas. Puedes repetir estos comandos para realizar cambios futuros. Recuerda que estas son confirmaciones locales, si deseas estos cambios en GitHub, deberás ejecutar un comando git push, que mostraremos más adelante.

Preguntas

¿Cuál es la diferencia entre el propósito y el contenido de Gemfile y Gemfile.lock? ¿Qué archivo se necesita para reproducir completamente las gemas del entorno de desarrollo en el entorno de producción?

El Gemfile establece las gemas requeridas y, en ocasiones, establece restricciones sobre las versiones que son compatibles. En cambio, el Gemfile.lock registra las versiones *concretas* que se han encontrado, no solo de las gemas que se han especificado de forma explícita, sino también de cualquier otra gema en la que esas dependan. Por lo tanto, el Gemfile.lock es el archivo utilizado en el entorno de producción para replicar de manera precisa las gemas disponibles en el entorno de desarrollo.

Después de ejecutar el bundle, ¿por qué aparecen gemas en Gemfile.lock que no estaban en Gemfile?

Bundler realizó una búsqueda de información para cada gema que solicitamos, en este caso, solo Sinatra, y se percató de que esta gema tenía dependencias adicionales, las cuales, a su vez, dependían de otras gemas. Como resultado, Bundler llevó a cabo una instalación recursiva de todas estas dependencias. Por ejemplo, el servidor de aplicaciones Rack es una de esas gemas, y aunque no la haya solicitado explícitamente, Sinatra depende de ella. Esto ilustra la eficacia de la automatización, ya que en lugar de requerir que nosotros, como desarrollador de la aplicación, comprendamos todas y cada una de las dependencias de las gemas, Bundler automatiza este proceso y nos permite enfocarte únicamente en las dependencias de alto nivel de tu aplicación.

Crea una aplicación SaaS sencilla con Sinatra

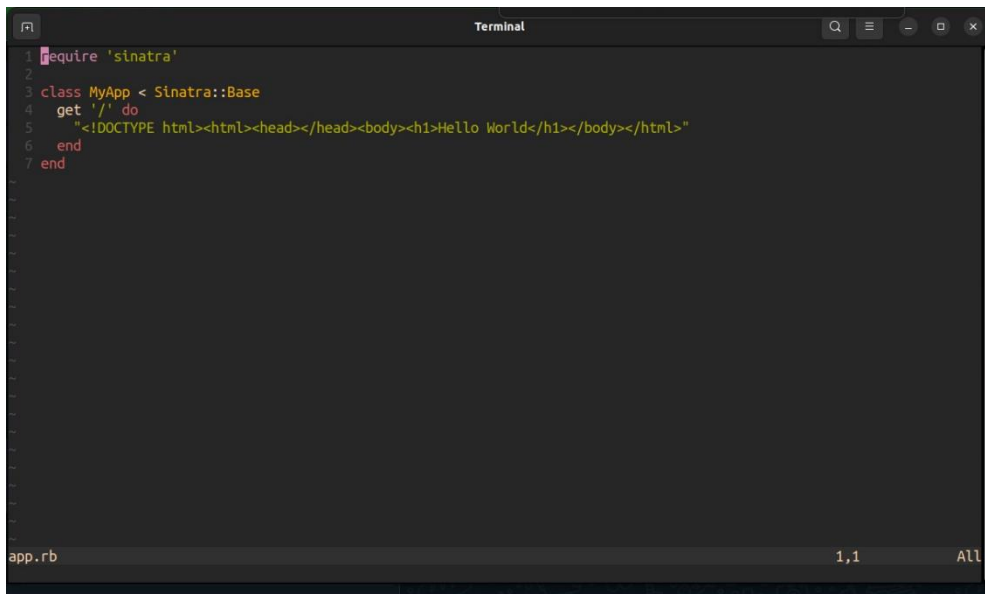
Como se ha explicado las aplicaciones SaaS requieren un servidor web para recibir solicitudes HTTP del mundo exterior y un servidor de aplicaciones que "conecte" la lógica de su aplicación al servidor web. Para el desarrollo, usaremos webrick, un servidor web muy simple basado en Ruby que sería inapropiado para producción pero que está bien para el desarrollo. Tanto en desarrollo como en producción, utilizaremos el servidor de aplicaciones en rack basado en Ruby, que admite aplicaciones Ruby escritas en varios frameworks, incluidos Sinatra y Rails.

Como se ha explicado una aplicación SaaS esencialmente reconoce y responde a las solicitudes HTTP correspondientes a las rutas de la aplicación (recuerda que una ruta consta de un método HTTP como GET o POST más un URI). Sinatra proporciona una abreviatura ligera para coincidir una ruta con el código de aplicación que se ejecutará cuando llegue una solicitud de uso de esa ruta desde el servidor web.

Crea un archivo en tu proyecto llamado `app.rb` que contenga lo siguiente:

```
require 'sinatra'

class MyApp < Sinatra::Base
  get '/' do
    "<!DOCTYPE html><html><head></head><body><h1>Hello World</h1></body></html>"
  end
end
```

A screenshot of a terminal window titled "Terminal". The window shows a text editor with the following code:

```
1 require 'sinatra'
2
3 class MyApp < Sinatra::Base
4   get '/' do
5     "<!DOCTYPE html><html><head></head><body><h1>Hello World</h1></body></html>"
6   end
7 end
```

The code is color-coded: `require` is pink, `class` is blue, `get` is green, and the string content is yellow. The terminal has a dark background and standard window controls at the top. At the bottom, it shows `app.rb` and line `1,1`.

El método `get` lo proporciona la clase `Sinatra::Base`, de la cual hereda la clase `MyApp`. `Sinatra::Base` está disponible porque cargamos la biblioteca `Sinatra` en la línea 1.

Como podemos ver en el ejemplo simple anterior, `Sinatra` nos permite escribir funciones que coinciden con una ruta HTTP entrante, en este caso `GET '/'` (la URL raíz), se devolverá a la presentación un documento HTML muy simple que contiene la cadena `Hello world` como resultado de la solicitud.

run MyApp

The image shows a terminal window titled "Terminal" with a dark background. The terminal content is as follows:

```
1 require './app'
2 run MyApp
```

At the bottom of the terminal, there is a status bar with the text "config.ru" on the left, "1,1" in the center, and "All" on the right.

```
bundle exec rackup --port 3000
```

```

x or q -- stop and exit

18:45:17 [rerun] PCol1_part01 stopping bundle se ejecuta en el entorno de desarrollo. (Los otros entornos que puede especificar son 'test
18:45:45 PM: jhozeal@l3tspaly-ASUS ~ /Desktop/PCol1_part01 [Master U2: 212 A] definir nuevos entornos). Se supone que las especificaciones de gemas
$ exec rerun -- rackup --port 3000 fuera de cualquier bloque de grupo se aplican en todos los entornos.

18:45:23 [rerun] PCol1_part01 launched
18:45:23 [rerun] Rerun (10564) running PCol1_part01 (13350) en la ventana de la terminal para iniciar tu aplicación y verificar que se esté
2023-10-01 18:45:23 INFO WeBrick 1.8.1 en la ventana de la terminal para iniciar tu aplicación y verificar que se esté
2023-10-01 18:45:23 INFO ruby 3.0.2 (2021-07-07) [x86_64-linux-gnu] en la ventana de la terminal para iniciar tu aplicación y verificar que se esté
2023-10-01 18:45:23 INFO WeBrick::HTTPServer#start: pid=13350 port=3000 en la ventana de la terminal para iniciar tu aplicación y verificar que se esté
18:45:25 [rerun] Watching . for **/*.rb,*.coffee,*.css,*.scss,*.sass,*.erb,*.haml,*.ru,*.yml,*.sln,*.d,*.feature,*.c,*.h with Linux adapter
127.0.0.1 - [01/Oct/2023:18:45:32 -0500] "GET / HTTP/1.1" 200 86.0108
127.0.0.1 - [01/Oct/2023:18:45:32 -0500] "GET /favicon.ico HTTP/1.1" 404 593.0009
127.0.0.1 - [01/Oct/2023:18:45:33 -0500] "GET / HTTP/1.1" 200 86.0014
18:45:44 [rerun] Change detected: 1 modified: app.rb de rerun (https://github.com/blasfr/rerun#usage) disponibles. Las
2023-10-01 18:45:44 FATAL: SignalException: SIGTERM los archivos README suelen estar llenos de instrucciones útiles sobre
18:45:44 [rerun] PCol1_part01 stopped
  /usr/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:173:in "select"
  /usr/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:173:in "block in start"
  /usr/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:32:in "start"
  /usr/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:168:in "start"
  /usr/lib/gems/3.0.0/gems/rack-2.2.8/lib/rack/handler/webrick.rb:45:in "run" el bundle exec para asegurarnos de que estamos usando las gemas
  /usr/lib/gems/3.0.0/gems/rack-2.2.8/lib/rack/server.rb:327:in "start" está ahí para asegurarse que el comando que queremos usar a ejecutar
  /usr/lib/gems/3.0.0/gems/rack-2.2.8/lib/rack/server.rb:168:in "start"
  /usr/lib/gems/3.0.0/gems/rack-2.2.8/bin/rackup$1:in "stop (required)" -- $IP. Podríamos lograr el mismo efecto con bundle exec rerun "rackup
  /usr/local/bin/rackup:25:in "load"
  /usr/local/bin/rackup:25:in "main" *0.0.0.0". Son equivalentes. Más importante aún, cualquier cambio detectado ahora hará que
2023-10-01 18:45:44 INFO going to shutdown se ejecuta automáticamente, similar al uso de guard para volver a ejecutar automáticamente
2023-10-01 18:45:44 INFO WeBrick::HTTPServer#start: pid=13362 port=3000 el especifique cuando los archivos cambian.
18:45:45 [rerun] PCol1_part01 restarted
18:45:45 [rerun] Rerun (10564) running PCol1_part01 (13462)
2023-10-01 18:45:45 INFO WeBrick 1.8.1 en la ventana de la terminal para iniciar tu aplicación y verificar que se detecte el cambio actualizando la
2023-10-01 18:45:45 INFO ruby 3.0.2 (2021-07-07) [x86_64-linux-gnu] en la ventana de la terminal para iniciar tu aplicación y verificar que se detecte el cambio actualizando la
2023-10-01 18:45:45 INFO WeBrick::HTTPServer#start: pid=13462 port=3000 la aplicación en ejecución. Además, antes de continuar, debes enviar tus
127.0.0.1 - [01/Oct/2023:18:45:49 -0500] "GET / HTTP/1.1" 200 81.0373
18:46:03 [rerun] Change detected: 1 modified: app.rb
2023-10-01 18:46:03 FATAL: SignalException: SIGTERM
18:46:03 [rerun] PCol1_part01 stopped
  /usr/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:173:in "select"
  /usr/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:173:in "block in start"
  /usr/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:32:in "start"
  /usr/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:168:in "start"
  /usr/lib/gems/3.0.0/gems/rack-2.2.8/lib/rack/handler/webrick.rb:45:in "run"
  /usr/lib/gems/3.0.0/gems/rack-2.2.8/lib/rack/server.rb:327:in "start"
  /usr/lib/gems/3.0.0/gems/rack-2.2.8/bin/rackup$1:in "stop (required)"
  /usr/local/bin/rackup:25:in "load"
  /usr/local/bin/rackup:25:in "main" *0.0.0.0". Son equivalentes. Más importante aún, cualquier cambio detectado ahora hará que

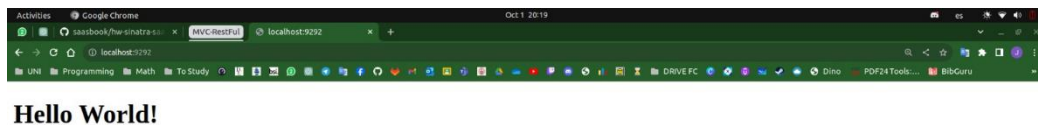
```


Este comando inicia el servidor de aplicaciones Rack y el servidor web WEBrick. Al anteponerle el paquete `exec` se garantiza que se esté ejecutando con las gemas especificadas en `Gemfile.lock`. Rack buscará `config.ru` e intentará iniciar la aplicación basándose en la información que contiene.

Para ver la aplicación web:

Visita `localhost:3000` en tu navegador para ver la aplicación web. Se abrirá en una nueva pestaña en el IDE si hace clic en él, pero debe abrir una nueva pestaña del navegador y pegar esa URL.

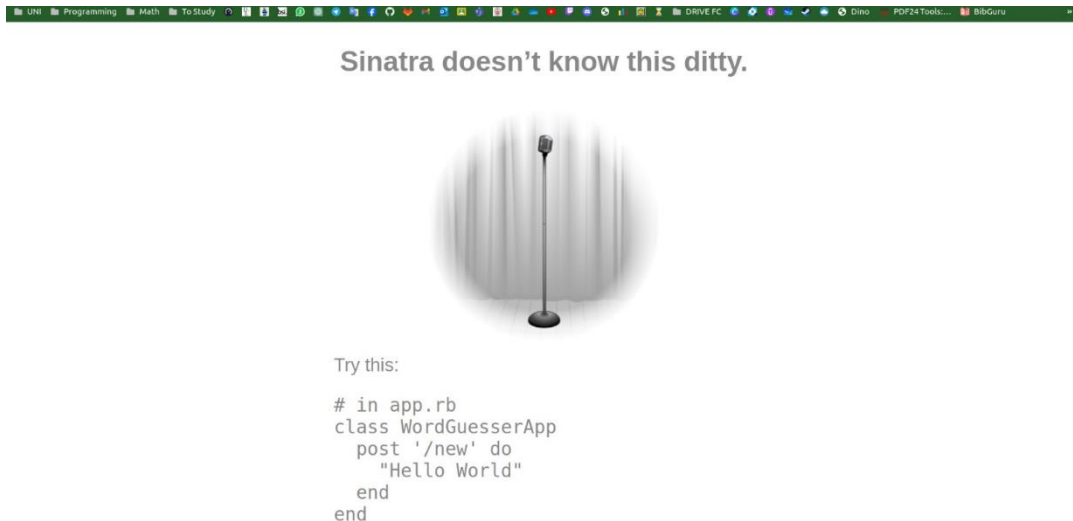
Apunta una nueva pestaña del navegador web a la URL de la aplicación en ejecución y verifica que puedas ver "Hello world".



Pregunta

¿Qué sucede si intentamos visitar una URL no raíz cómo `https://localhost:3000/hello` y por qué? (la raíz de tu URL variará)

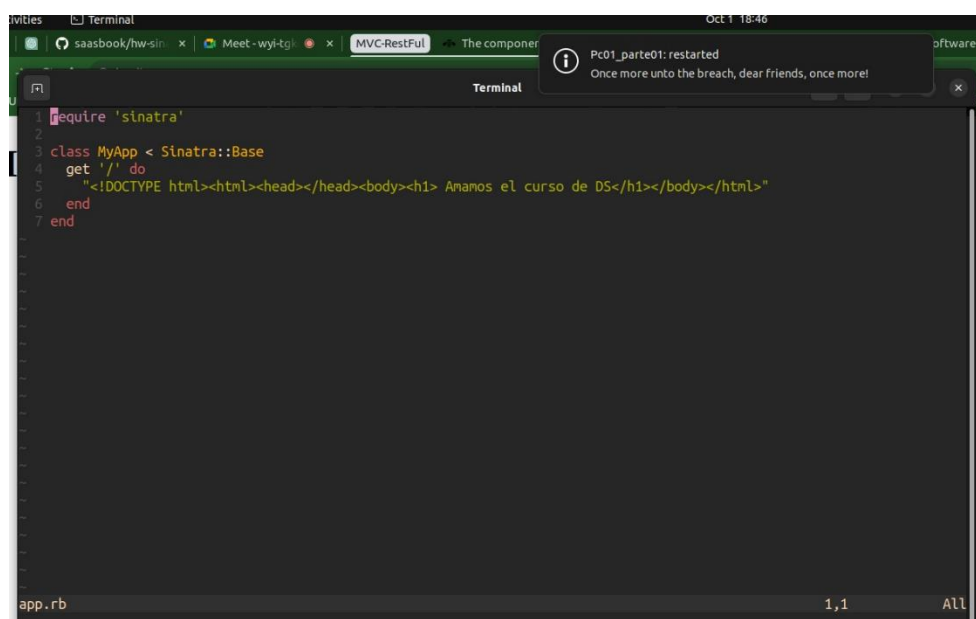
Recibiremos un mensaje de error humorístico proveniente del framework Sinatra, debido a la ausencia de una ruta coincidente con `'get '/hello'` en la aplicación. Puesto que Sinatra es un framework orientado a servicios de software como servicio (SaaS), dicho mensaje de error se encapsula en una página web y se envía al navegador del usuario.



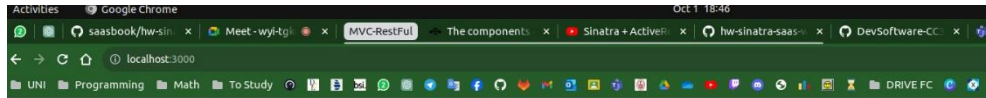
Ahora deberíamos tener los siguientes archivos bajo control de versiones: Gemfile, Gemfile.lock, app.rb, config.ru. Esta es una aplicación SaaS mínima: el archivo de la aplicación en sí, la lista de gemas requeridas explícitamente, la lista de gemas reales instaladas, incluidas las dependencias implícitas en las gemas requeridas y un archivo de configuración que le indica al servidor de aplicaciones cómo iniciar la aplicación.

Modifica la aplicación

Modificamos app.rb para que en lugar de "Hello world" imprima "Amamos el curso de DS". Guardamos los cambios en app.rb y actualizamos la pestaña de nuestro navegador donde se ejecuta la aplicación.



Ahora regresamos a la ventana del shell donde ejecutamos rackup y presionamos Ctrl-C para detener Rack. Luego escribimos `bundle exec rackup --port 3000` para desarrollo local y una vez que se esté ejecutando, regresamos a la pestaña de nuestro navegador con nuestra aplicación y actualizamos la página. Esta vez debería funcionar.

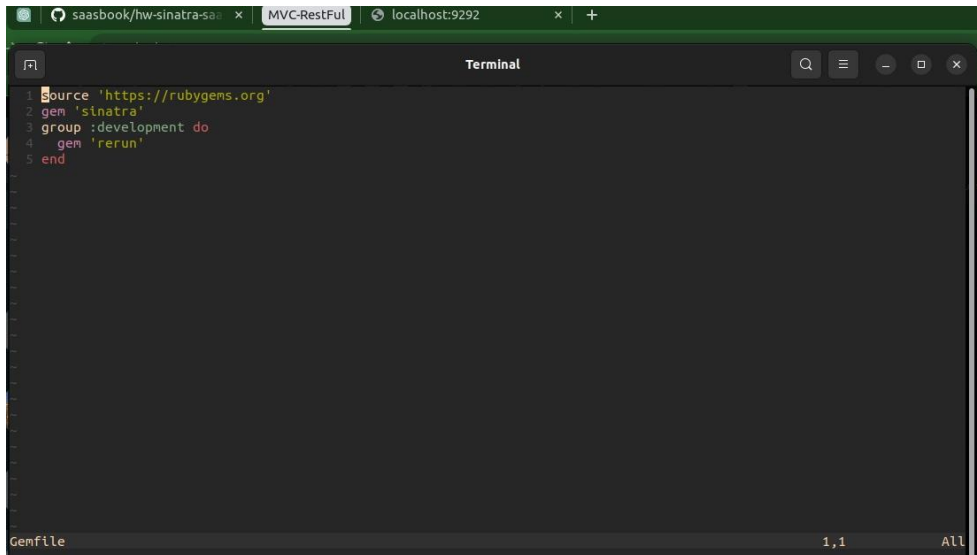


Amamos el curso de DS

Lo que esto nos muestra es que, si modificamos la aplicación mientras se está ejecutando, debemos reiniciar Rack para que "veamos" esos cambios. Dado que reiniciarlo manualmente es tedioso, usaremos la gema de `rerun`, que reinicia Rack automáticamente cuando ve cambios en los archivos en el directorio de la aplicación. (Rails hace esto de forma predeterminada durante el desarrollo, como veremos, pero Sinatra no).

Duda: Si la aplicación depende de esta gema adicional, debemos agregarla al Gemfile y ejecutar el paquete para asegurarnos de que esté realmente presente". ¿Buen pensamiento no? Pero también se nos puede ocurrir que esta gema en particular no sería necesaria en un entorno de producción: sólo la necesitamos como herramienta durante el desarrollo. Afortunadamente, hay una manera de decirle a Bundler que algunas gemas sólo son necesarias en determinados entornos. Agregamos lo siguiente al Gemfile (no importa dónde):

```
group :development do
  gem 'rerun'
end
```

A screenshot of a terminal window with a dark background. The terminal title bar shows 'Terminal' and standard window controls. The content of the terminal is a Gemfile with the following lines: 1 source 'https://rubygems.org', 2 gem 'sinatra', 3 group :development do, 4 gem 'rerun', 5 end. The bottom status bar of the terminal shows 'Gemfile', '1,1', and 'All'.

Ahora ejecutamos bundle install para que descargamos la gema que se vuelve a ejecutar y las dependencias, si aún no están implementadas.

Cualquier especificación de gema dentro de group :development de desarrollo solo se examinará si el bundle se ejecuta en el entorno de desarrollo. (Los otros entornos que puede especificar son :test y :production, y nosotros mismos podemos definir nuevos entornos). Se supone que las especificaciones de gemas fuera de cualquier bloque de grupo se aplican en todos los entornos.

Seguimos lo siguiente en la ventana de la terminal para iniciar nuestra aplicación y verificar que se esté ejecutando:

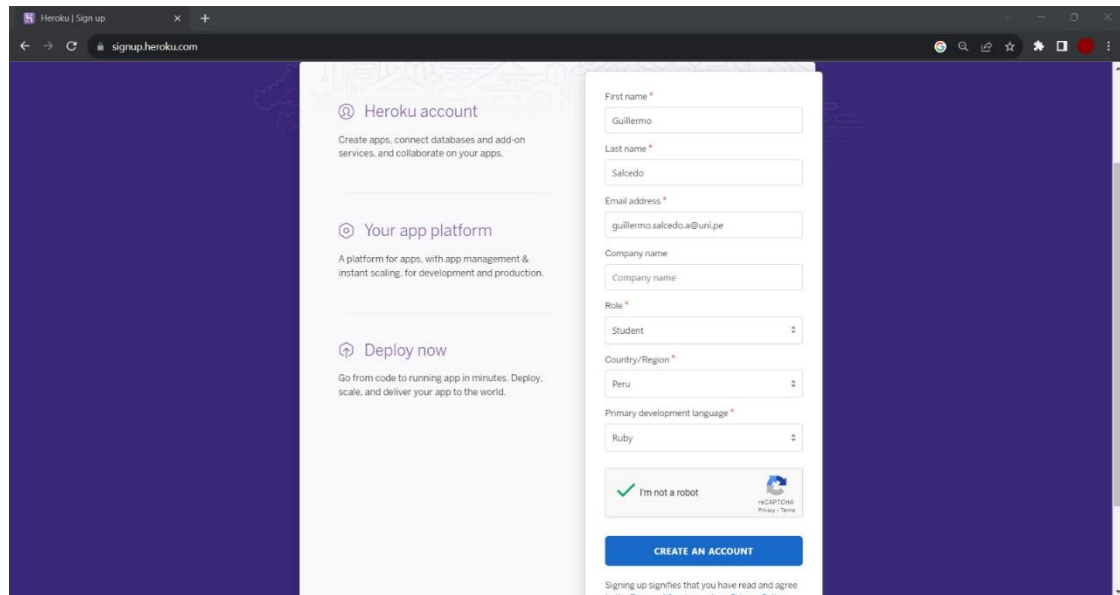
bundle exec rerun -- rackup --port 3000 Hay más detalles sobre el uso de rerun (<https://github.com/alexch/rerun#usage>) disponibles. Las gemas suelen estar en GitHub y sus archivos README suelen estar llenos de instrucciones útiles sobre cómo utilizarlos.

En este caso, volvemos a anteponer el bundle exec para asegurarnos de que estamos usando las gemas en Gemfile.lock y el símbolo -- está ahí para aseverar que el comando que queremos volver a ejecutar para operar es rackup -p \$PORT - \$IP. Podríamos lograr el mismo efecto con bundle exec rerun "rackup -p 3000 -o 0.0.0.0". Son equivalentes. Más importante aún, cualquier cambio detectado ahora hará que el servidor se reinicie automáticamente, similar al uso de guard para volver a ejecutar automáticamente el specfile cuando los archivos cambian.

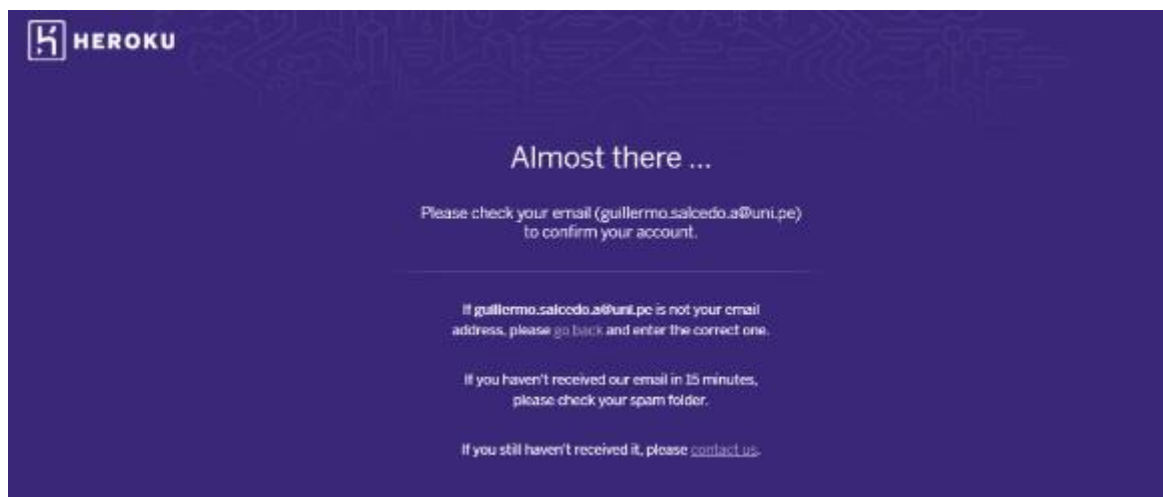
Modificamos app.rb para imprimir un mensaje diferente y verifica que se detecte el cambio actualizando la pestaña de tu navegador con la aplicación en ejecución. Además, antes de continuar, debes enviar tus últimos cambios a git.

Implementar en Heroku

Heroku es una plataforma como servicio (PaaS) en la nube donde podemos implementar las aplicaciones Sinatra (y posteriores Rails). Ya que aún no teníamos una cuenta, nos registramos en <http://www.heroku.com>. Necesitaremos el nombre de usuario y contraseña para el siguiente paso. Instalamos Heroku CLI siguiendo las instrucciones.



The screenshot shows the Heroku sign-up page in a web browser. The page has a dark blue background with a white sidebar on the left containing three sections: 'Heroku account', 'Your app platform', and 'Deploy now'. The main content area is a white form with the following fields: 'First name' (filled with 'Guillermo'), 'Last name' (filled with 'Salcedo'), 'Email address' (filled with 'guillermo.salcedo.a@uni.pe'), 'Company name' (empty), 'Role' (dropdown menu with 'Student' selected), 'Country/Region' (dropdown menu with 'Peru' selected), and 'Primary development language' (dropdown menu with 'Ruby' selected). Below these fields is a reCAPTCHA widget with the text 'I'm not a robot' and a 'CREATE AN ACCOUNT' button. At the bottom, there is a small line of text: 'Signing up signifies that you have read and agree to the Terms of Service and our Privacy Policy'.



Confirm your account on Heroku

External

Inbox x



Heroku <noreply@heroku.com>
to me

20:21 (0 minutes ago) ☆ ↶ ⋮




Thanks for signing up with Heroku! You must follow this link within 30 days of registration to activate your account:

<https://id.heroku.com/account/accept/14503655/005279bdaaf857b8a4a33f207621ef73>

Have fun, and don't hesitate to contact us with your feedback.

The Heroku Team
<https://heroku.com>



Set your password

Create your password and log in to your Heroku account.


Create a new password *

Password confirmation *

Password requirements

- ✓ Must be a minimum of 8 characters.
- ✓ Must contain letters, numbers, and symbols.
- ✓ Passwords must match.

SET PASSWORD AND LOG IN



Log in to your account

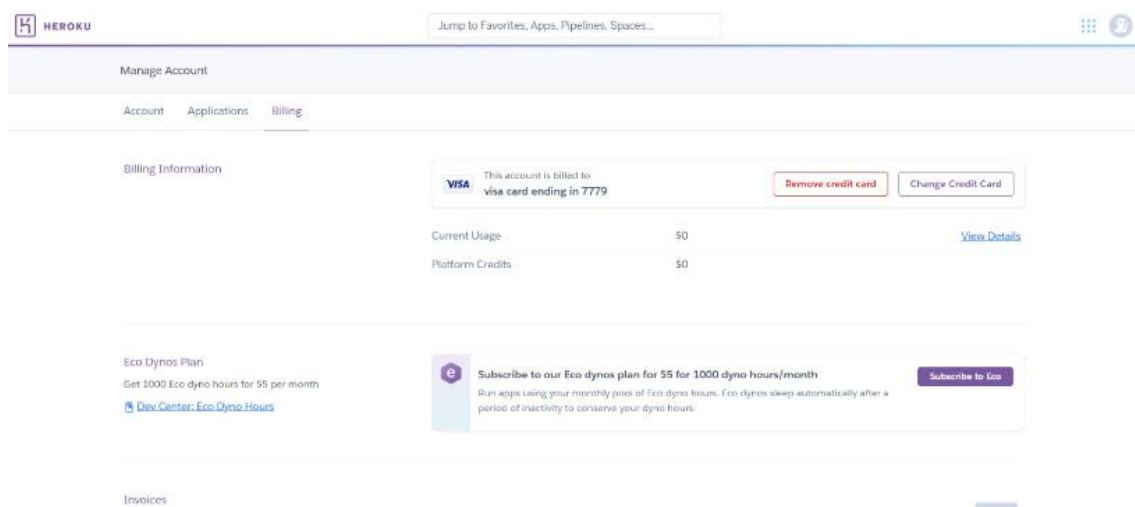
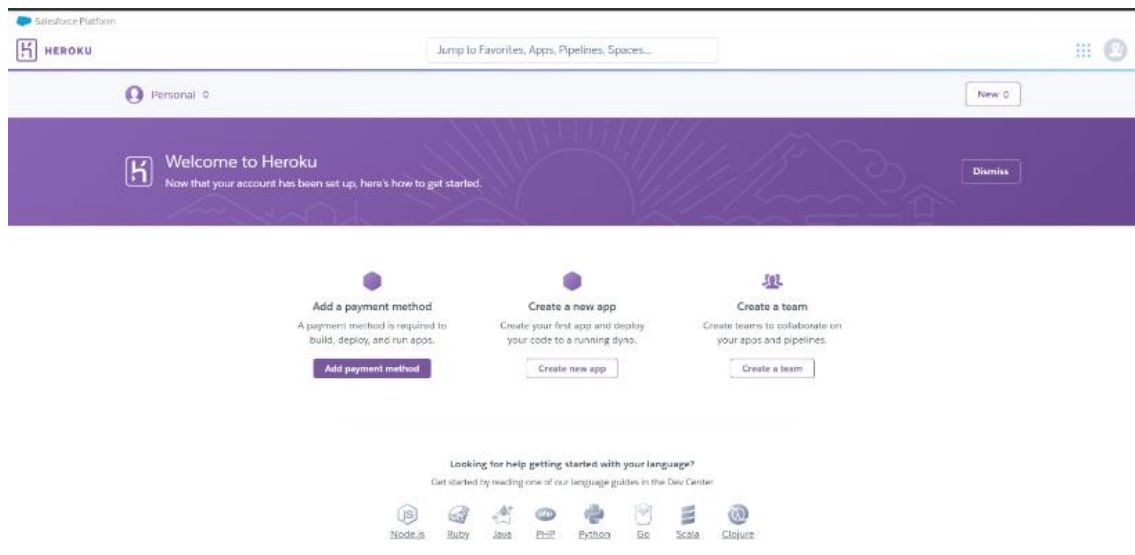
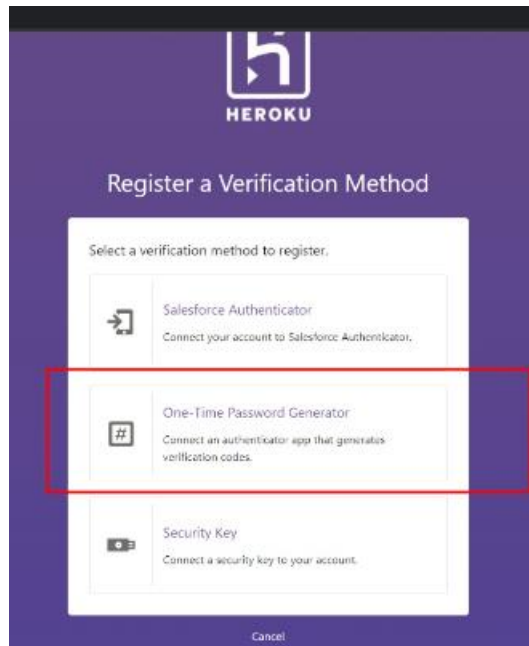
Email address

Password

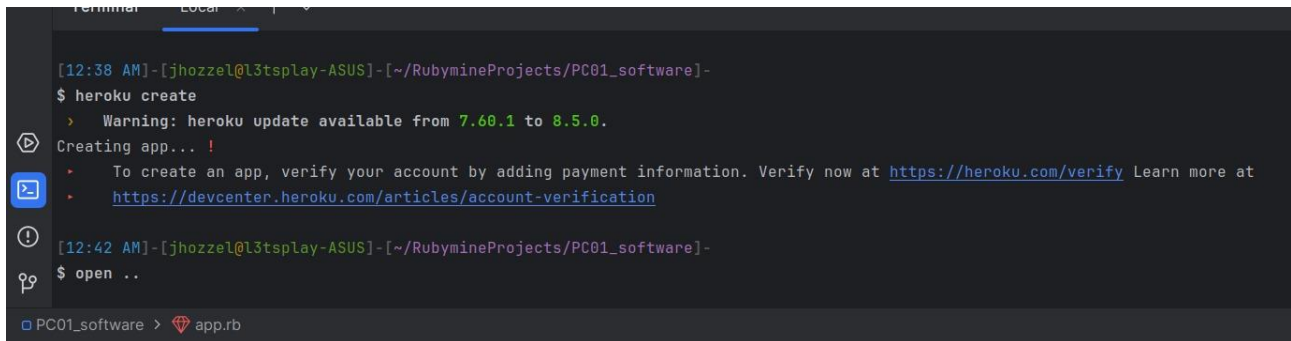
Log In

New to Heroku? [Sign Up](#)

[Log in via SSO](#) [Forgot your password?](#)

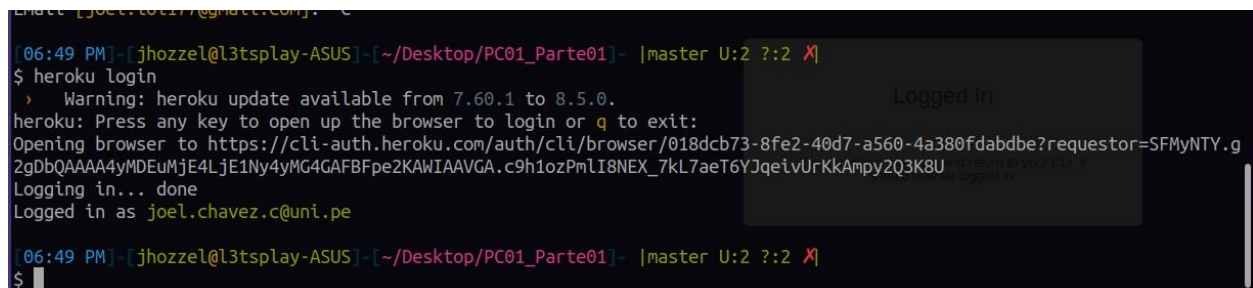


Y si no tenemos una tarjeta asociada a nuestra cuenta, nos aparecerá lo siguiente:



```
[12:38 AM]-[jhozzel@l3tsplay-ASUS]-[~/RubymineProjects/PC01_software]-  
$ heroku create  
  > Warning: heroku update available from 7.60.1 to 8.5.0.  
Creating app... !  
  * To create an app, verify your account by adding payment information. Verify now at https://heroku.com/verify Learn more at  
  * https://devcenter.heroku.com/articles/account-verification  
[12:42 AM]-[jhozzel@l3tsplay-ASUS]-[~/RubymineProjects/PC01_software]-  
$ open ..  
PC01_software > app.rb
```

Iniciamos sesión en nuestra cuenta Heroku escribiendo el comando: `heroku login -i` en la terminal. Esto nos conectará con nuestra cuenta Heroku.



```
[06:49 PM]-[jhozzel@l3tsplay-ASUS]-[~/Desktop/PC01_Parte01]- |master U:2 ??:2 X|  
$ heroku login  
  > Warning: heroku update available from 7.60.1 to 8.5.0.  
heroku: Press any key to open up the browser to login or q to exit:  
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/018dcb73-8fe2-40d7-a560-4a380fdabdbe?requestor=SFMyNTY.g2gDbQAAAA4yMDEuMjE4LjE1Ny4yMG4GAFBFpe2KAWIAAVGA.c9h1ozPmLI8NEX\_7kL7aeT6YJqeivUrKkAmpy2Q3K8U  
Logging in... done  
Logged in as joel.chavez.c@uni.pe  
[06:49 PM]-[jhozzel@l3tsplay-ASUS]-[~/Desktop/PC01_Parte01]- |master U:2 ??:2 X|  
$
```

Mientras estemos en el directorio raíz de nuestro proyecto (no en todo nuestro espacio de trabajo), escribimos `heroku create` para crear un nuevo proyecto en Heroku. Esto le indicará al servicio Heroku que se prepare para algún código entrante y localmente agregará un repositorio git remoto llamado heroku.

A continuación, nos aseguramos de preparar y confirmar todos los cambios localmente como se indicó anteriormente (es decir, git add, git commit, etc.).

```
config.ru

no changes added to commit (use "git add" and/or "git commit -a")
[06:50 PM] - [jhozzel@l3tsplay-ASUS ~] $ git add .
[06:50 PM] - [jhozzel@l3tsplay-ASUS ~] $ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>" to unstage)
        modified:   Gemfile
        modified:   Gemfile.lock
        new file:   app.rb
        new file:   config.ru
[06:50 PM] - [jhozzel@l3tsplay-ASUS ~] $ git commit -m "Update"
[master ffbcf3b] Update
 4 files changed, 22 insertions(+)
 create mode 100644 app.rb
 create mode 100644 config.ru
[06:50 PM] - [jhozzel@l3tsplay-ASUS ~] $
```

Anteriormente vimos que para ejecutar la aplicación localmente se ejecuta rackup para iniciar el servidor de aplicaciones Rack, y Rack busca en config.ru para determinar cómo iniciar tu aplicación Sinatra. ¿Cómo se le dice a un entorno de producción cómo iniciar un servidor de aplicaciones u otros procesos necesarios para recibir solicitudes e iniciar su aplicación?

En el caso de Heroku, esto se hace con un archivo especial llamado Procfile, que especifica uno o más tipos de procesos Heroku que usará tu aplicación y cómo iniciar cada uno. El tipo de proceso Heroku más básico se llama Dyno o "web worker". Dyno puede atender una solicitud de usuario a la vez. Como estamos en el nivel gratuito de Heroku, solo podemos tener un Dyno.

web: bundle exec rackup config.ru -p \$PORT

```
Terminal
x or q -- stop and exit

18:45:17 [rerun] Pc01_parte01 stopping bundle se ejecuta en el entorno de desarrollo. (Los otros entornos que puede especificar son :test
[06:45 PM] - [jhozzel@l3tsplay-ASUS ~] $ exec rerun -- rackup -p 3000
18:45:23 [rerun] Pc01_parte01 launched
18:45:23 [rerun] Rerun (10564) running Pc01_parte01 (13350)
[2023-10-01 18:45:23] INFO WEBrick 1.8.1
[2023-10-01 18:45:23] INFO ruby 3.0.2 (2021-07-07) [x86_64-linux-gnu]
[2023-10-01 18:45:23] INFO WEBrick::HTTPServer#start: pid=13350 port=3000
18:45:25 [rerun] Watching - for **/*: {rb,js,coffee,css,scss,sass,erb,html,haml,ru,yml,sln,md,feature,c,h} with Linux adapter
127.0.0.1 - - [01/Oct/2023:18:45:32 -0500] "GET / HTTP/1.1" 200 80 0.0168
127.0.0.1 - - [01/Oct/2023:18:45:32 -0500] "GET /favicon.ico HTTP/1.1" 404 503 0.0009
127.0.0.1 - - [01/Oct/2023:18:45:33 -0500] "GET / HTTP/1.1" 200 80 0.0014
18:45:44 [rerun] Change detected: 1 modified: app.rb
[2023-10-01 18:45:44] FATAL SignalException: SIGTERM
/var/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:173:in 'select'
/var/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:32:in 'start'
/var/lib/gems/3.0.0/gems/webrick-1.8.1/lib/webrick/server.rb:168:in 'start'
/var/lib/gems/3.0.0/gems/rack-2.2.8/lib/rack/handler/webrick.rb:41:in 'run'
/var/lib/gems/3.0.0/gems/rack-2.2.8/lib/rack/server.rb:327:in 'start'
/var/lib/gems/3.0.0/gems/rack-2.2.8/lib/rack/server.rb:168:in 'start'
/var/lib/gems/3.0.0/gems/rack-2.2.8/bin/rackup:5:in 'start' (required)
/usr/local/bin/rackup:25:in 'load'
/usr/local/bin/rackup:25:in 'main'
[2023-10-01 18:45:44] INFO WEBrick: HTTPServer#start done
[2023-10-01 18:45:44] INFO WEBrick: HTTPServer#start done
18:45:45 [rerun] Pc01_parte01 restarted
18:45:45 [rerun] Rerun (10564) running Pc01_parte01 (13462)
[2023-10-01 18:45:45] INFO WEBrick 1.8.1
[2023-10-01 18:45:45] INFO ruby 3.0.2 (2021-07-07) [x86_64-linux-gnu]
```

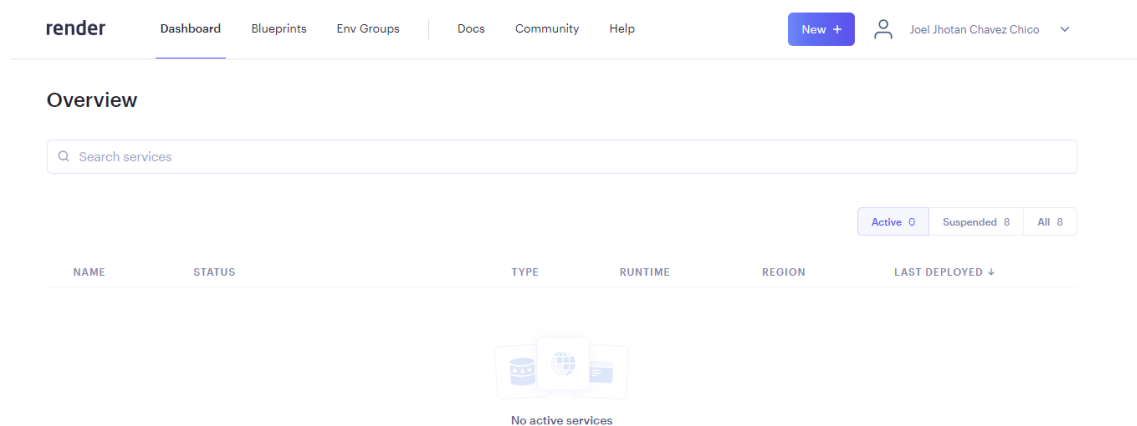
Esto le indica a Heroku que inicie un único trabajador web (Dyno) usando esencialmente la misma línea de comando que se usó para iniciar Rack localmente. Debemos tener en cuenta que, en algunos casos, no es necesario un Procfile ya que Heroku puede deducir de sus archivos cómo iniciar la aplicación. Sin embargo, siempre es mejor ser explícito.

Nuestro repositorio local ahora está listo para implementarse en Heroku, lo único que nos faltaría sería ejecutar el siguiente comando

\$ git push heroku master

(**master** se refiere a qué rama del repositorio remoto de Heroku estamos enviando). Crearemos una instancia en ejecución de nuestra aplicación en alguna URL que termine en herokuapp.com. Ingresamos esa URL en una nueva pestaña del navegador para ver nuestra aplicación ejecutándose en vivo. Debemos tener nuestra aplicación activa.

Sin embargo dado que Heroku ya no cuenta con planes gratuitos desde noviembre de 2022, lo que haremos será utilizar la plataforma de Render



Nosotros ya tenemos creada nuestra cuenta asociada a nuestro GitHub, así que vamos, a la opción de “New” y le damos a “Web Service”,

Create a new Web Service

Connect a Git repository, or use an existing image.

How would you like to deploy your web service?

☒ Build and deploy from a Git repository

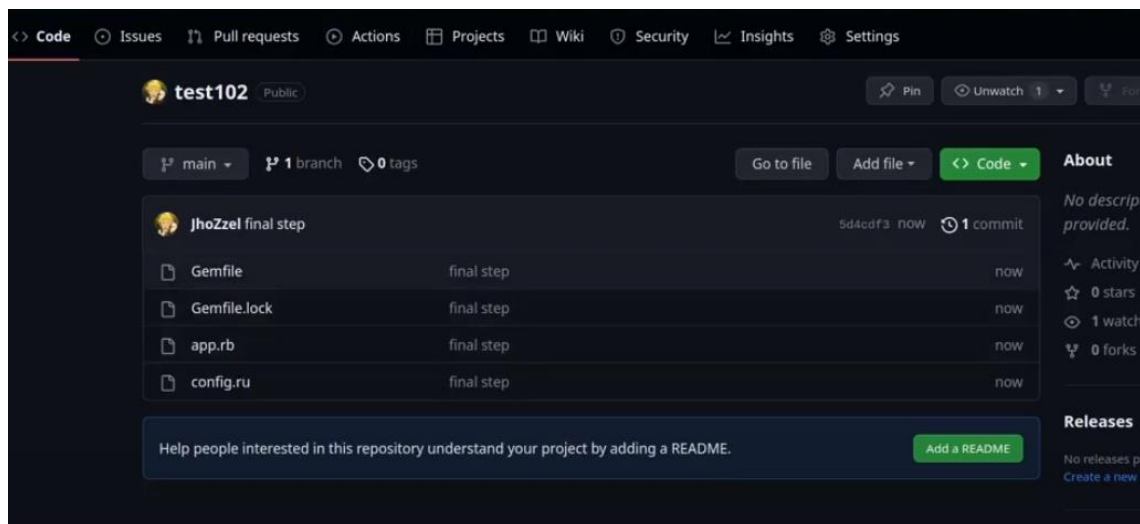
Connect a GitHub or GitLab repository.

☐ Deploy an existing image from a registry **ADVANCED**

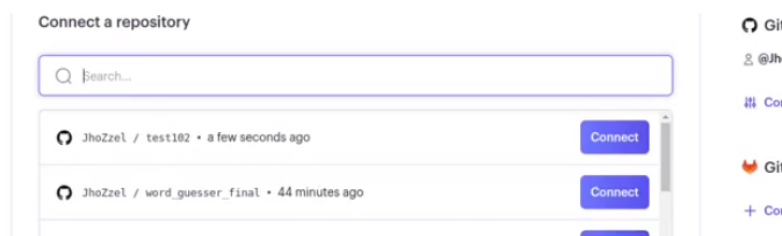
Pull a public image from any registry or a private image from Docker Hub, GitHub, or GitLab.

Next

En nuestro caso hemos subido nuestro repositori bajo el nombre de “test102”,



Nos conectamos al repositorio para poder desplegarlo



Asignamos el nombre de nuestra aplicación SaAs que será desplegada

You are deploying a web service for **JhoZzel/test102**.

You seem to be using **Sinatra**, so we've autofilled some fields accordingly. Make sure the values look right to you!

Name

A unique name for your web service.

PARTE1_test1

Region

The **region** where your web service runs. Services must be in the same region to communicate privately and you currently have services running in **Oregon**.

Oregon (US West)

Branch

The repository branch used for your web service.

main

Root Directory

Optional

Defaults to repository root. When you specify a **root directory** that is different from your repository root, **Render** runs all your commands in the **specified directory** and ignores changes outside the directory.

e.g., src

Runtime

The runtime for your web service.

Ruby

Build Command

This command runs in the root directory of your repository when a new version of your code is pushed, or when you deploy manually. It is typically a script that installs libraries, runs migrations, or compiles resources

\$ bundle install

Y finalmente la desplegamos bajo el plan gratuito

Instance type	RAM	CPU	Price
<input checked="" type="radio"/> Free	512 MB	0.1 CPU	\$0 / month
<input type="radio"/> Starter	512 MB	0.5 CPU	\$7 / month
<input type="radio"/> Standard	2 GB	1 CPU	\$25 / month
<input type="radio"/> Pro	4 GB	2 CPU	\$85 / month
<input type="radio"/> Pro Plus	8 GB	4 CPU	\$175 / month
<input type="radio"/> Pro Max	16 GB	4 CPU	\$225 / month
<input type="radio"/> Pro Ultra	32 GB	8 CPU	\$450 / month

Need a custom instance type? We support up to 512 GB RAM and 64 CPUs.

1

Unlike paid services, free services scale down when inactive. [Learn more about free instance type limits.](#)

Advanced

Create Web Service

Observemos que se esta construyendo la aplicación para su despliegue

WEB SERVICE

PARTE1_test Ruby Free

Connect Manual Deploy

JhoZze1 / test102 main

https://parte1-test.onrender.com

Events

Logs

Disks

Environment

Shell

Previews

Jobs

Metrics

Scaling

Settings

Free instance types will spin down with inactivity. Upgrade to a paid instance type to prevent this behavior. Learn more.

October 6, 2023 at 9:40 PM Building

5d4cdf3 final step

Cancel deploy

Search logs Search

Maximize Scroll to top

```
Oct 6 09:40:02 PM => Cloning from https://github.com/JhoZze1/test102...
Oct 6 09:40:02 PM => Checking out commit 5d4cdf3a026bacd0686fbd5b23df69080e9ffb02 in branch main
Oct 6 09:40:06 PM => Using Node version 14.17.0 (default)
Oct 6 09:40:06 PM => Docs on specifying a Node version: https://render.com/docs/node-version
Oct 6 09:40:06 PM => Using Ruby version 2.6.6 via /opt/render/project/src/Gemfile
Oct 6 09:40:06 PM => Docs on specifying a Ruby version: https://render.com/docs/ruby-version
```

Scroll to bottom

WEB SERVICE

PARTE1_test Ruby Free

Connect Manual Deploy

JhoZze1 / test102 main

https://parte1-test.onrender.com

Events

Logs

Disks

Environment

Shell

Previews

Free instance types will spin down with inactivity. Upgrade to a paid instance type to prevent this behavior. Learn more.

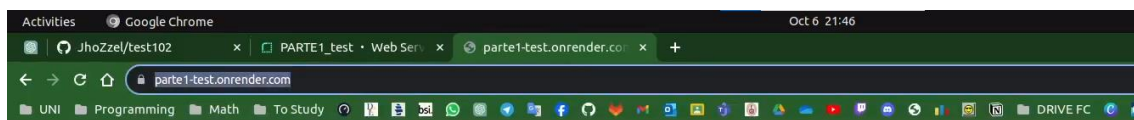
October 6, 2023 at 9:40 PM Live

5d4cdf3 final step

Search logs Search

Maximize Scroll to top

Vista del despliegue completado:

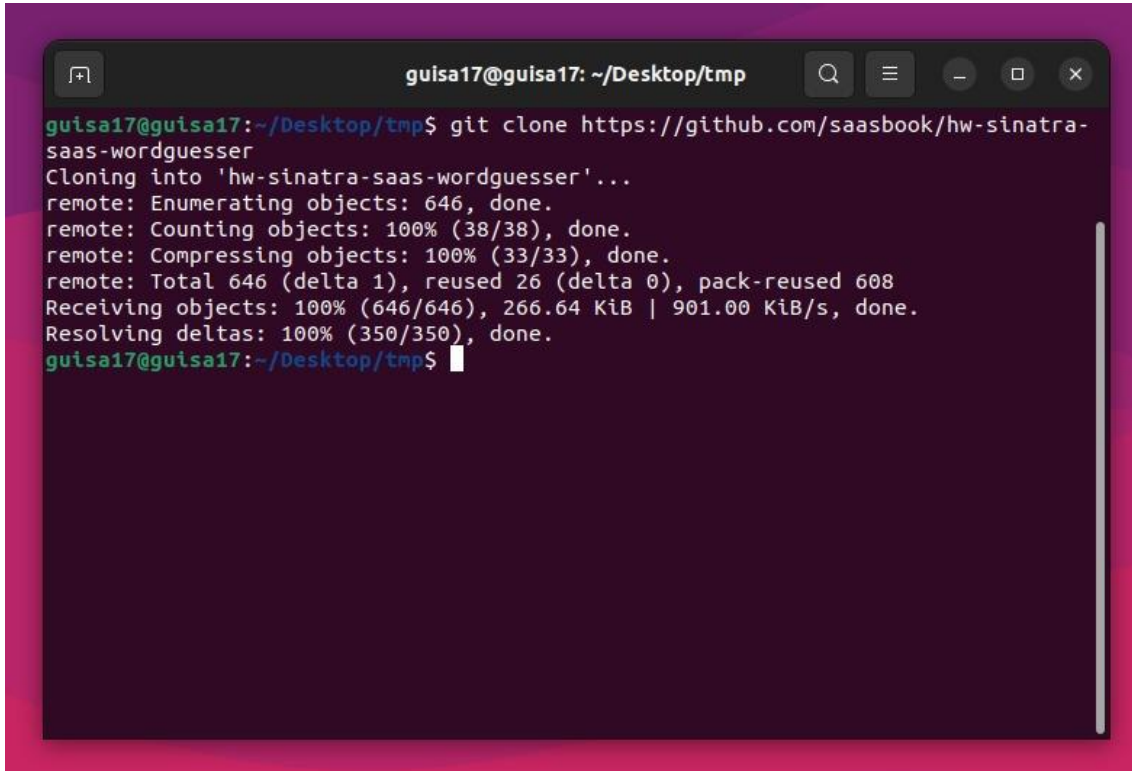


Hello Im Joel

Parte 1: Wordguesser

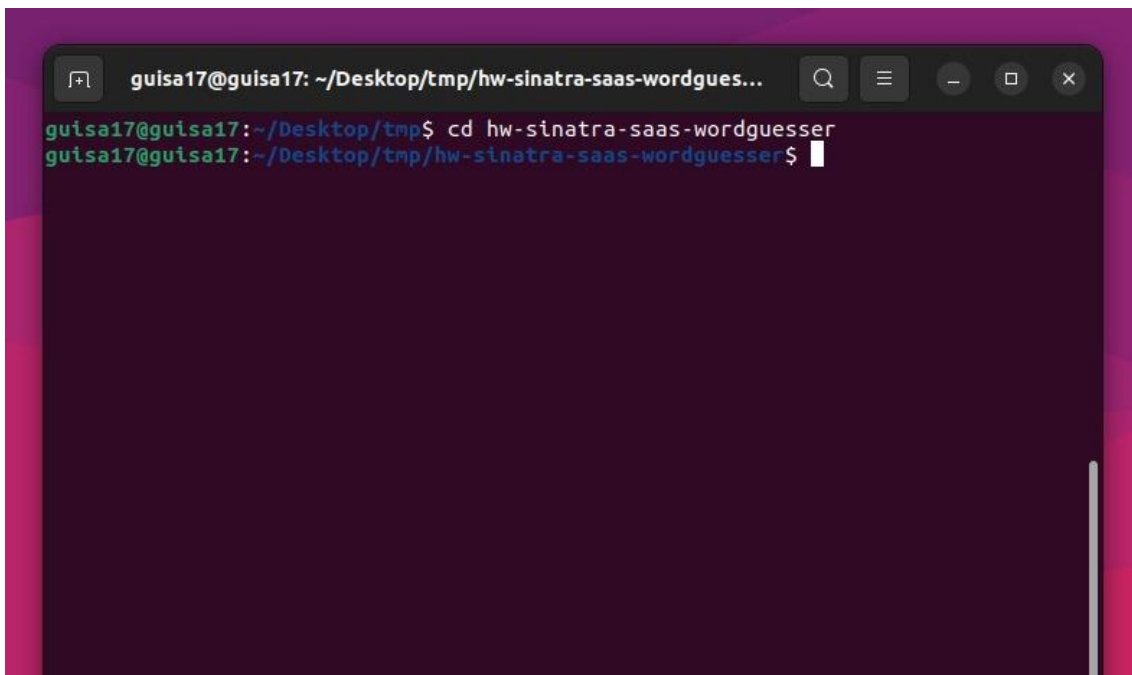
Con toda esta maquinaria en mente, clonamos este repositorio y trabajemos el juego de adivinar palabras (Wordguesser).

git clone <https://github.com/saasbook/hw-sinatra-saas-wordguesser>

A terminal window with a dark background and light green text. The window title is 'guisa17@guisa17: ~/Desktop/tmp'. The command 'git clone https://github.com/saasbook/hw-sinatra-saas-wordguesser' has been executed. The output shows the progress of cloning the repository, including enumerating, counting, and compressing objects, and receiving the final pack file.

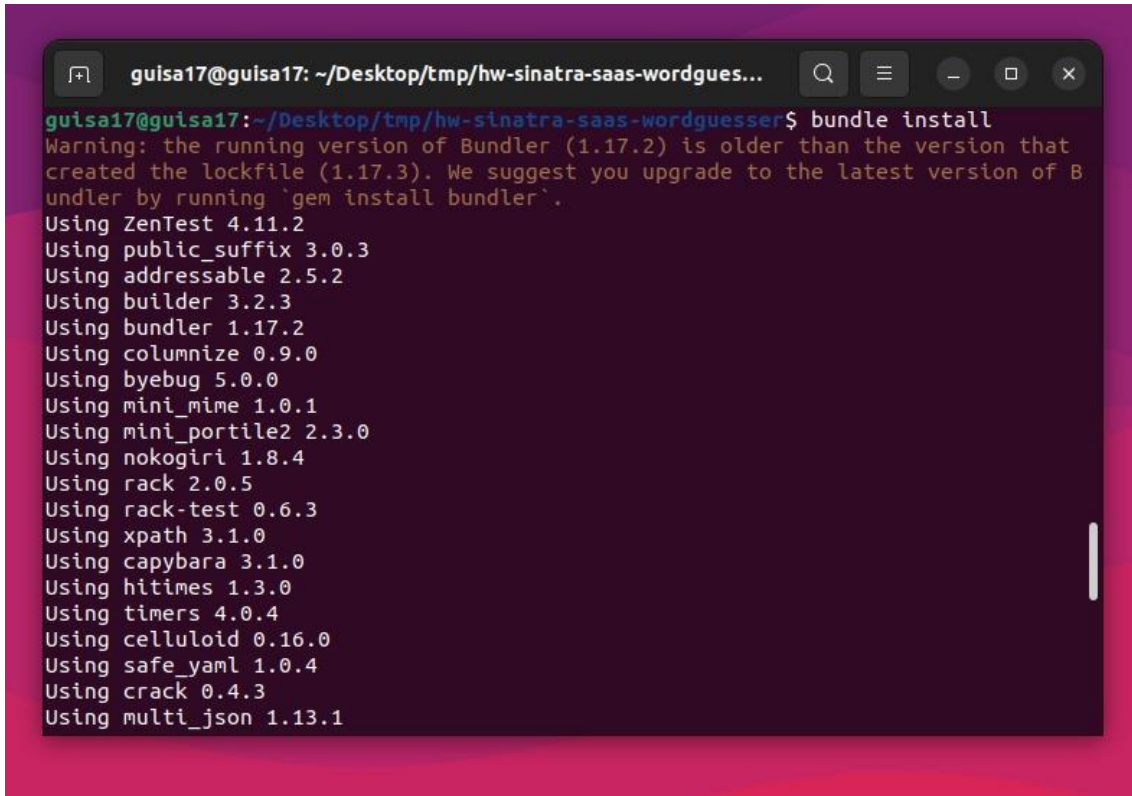
```
guisa17@guisa17:~/Desktop/tmp$ git clone https://github.com/saasbook/hw-sinatra-saas-wordguesser
Cloning into 'hw-sinatra-saas-wordguesser'...
remote: Enumerating objects: 646, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (33/33), done.
remote: Total 646 (delta 1), reused 26 (delta 0), pack-reused 608
Receiving objects: 100% (646/646), 266.64 KiB | 901.00 KiB/s, done.
Resolving deltas: 100% (350/350), done.
guisa17@guisa17:~/Desktop/tmp$
```

cd hw-sinatra-saas-wordguesser

A terminal window with a dark background and light green text. The window title is 'guisa17@guisa17: ~/Desktop/tmp/hw-sinatra-saas-wordgues...'. The command 'cd hw-sinatra-saas-wordguesser' has been executed, changing the current directory to the cloned repository.

```
guisa17@guisa17:~/Desktop/tmp$ cd hw-sinatra-saas-wordguesser
guisa17@guisa17:~/Desktop/tmp/hw-sinatra-saas-wordguesser$
```

bundle install

A terminal window with a dark background and light-colored text. The window title is "guisa17@guisa17: ~/Desktop/tmp/hw-sinatra-saas-wordgues...". The prompt is "guisa17@guisa17:~/Desktop/tmp/hw-sinatra-saas-wordguesser\$". The command "bundle install" has been executed. The output shows a warning about Bundler versions and a list of installed gems. The gems listed are: ZenTest 4.11.2, public_suffix 3.0.3, addressable 2.5.2, builder 3.2.3, bundler 1.17.2, columnize 0.9.0, byebug 5.0.0, mini_mime 1.0.1, mini_portile2 2.3.0, nokogiri 1.8.4, rack 2.0.5, rack-test 0.6.3, xpath 3.1.0, capybara 3.1.0, hitimes 1.3.0, timers 4.0.4, celluloid 0.16.0, safe_yaml 1.0.4, crack 0.4.3, and multi_json 1.13.1.

```
guisa17@guisa17: ~/Desktop/tmp/hw-sinatra-saas-wordgues...
guisa17@guisa17:~/Desktop/tmp/hw-sinatra-saas-wordguesser$ bundle install
Warning: the running version of Bundler (1.17.2) is older than the version that
created the lockfile (1.17.3). We suggest you upgrade to the latest version of B
undler by running `gem install bundler`.
Using ZenTest 4.11.2
Using public_suffix 3.0.3
Using addressable 2.5.2
Using builder 3.2.3
Using bundler 1.17.2
Using columnize 0.9.0
Using byebug 5.0.0
Using mini_mime 1.0.1
Using mini_portile2 2.3.0
Using nokogiri 1.8.4
Using rack 2.0.5
Using rack-test 0.6.3
Using xpath 3.1.0
Using capybara 3.1.0
Using hitimes 1.3.0
Using timers 4.0.4
Using celluloid 0.16.0
Using safe_yaml 1.0.4
Using crack 0.4.3
Using multi_json 1.13.1
```

Desarrollo de Wordguesser usando TDD y Guard

En esta actividad utilizaremos el desarrollo basado en pruebas (TDD) basado en las pruebas que proporcionamos para desarrollar la lógica del juego para Wordguesser, lo que nos obliga a pensar en qué datos son necesarios para capturar el estado del juego. Esto será importante cuando hagamos el juego en forma SaaS en la siguiente parte.

¿Qué haremos?: utilizaremos **autotest**: los casos de prueba proporcionados se volverán a ejecutar cada vez que realicemos un cambio en el código de la aplicación. Una por una, las pruebas pasarán de rojo (reprobado) a verde (aprobado) a medida que creamos el código de la aplicación. Cuando hayamos terminado, tendremos una clase de juego Wordguesser funcional, lista para ser "envuelta" en SaaS usando Sinatra.

El juego de adivinanzas de palabras basado en la Web funcionará de la siguiente manera:

- La computadora elige una palabra al azar.
- El jugador adivina letras para adivinar la palabra.

- Si el jugador adivina la palabra antes de adivinar siete letras incorrectamente, gana, de lo contrario pierden. (Adivinar la misma letra repetidamente simplemente se ignora).
- Una letra que ya ha sido adivinada o que no es un carácter del alfabeto se considera "no válida", es decir, no es una adivinación "válida".

Para que el juego sea divertido, cada vez que iniciemos un nuevo juego, la aplicación recuperará una palabra en inglés aleatoria de un servidor remoto, por lo que cada juego será diferente. Esta característica nos presentará no sólo el uso de un servicio externo (el generador de palabras aleatorias) como un "bloque de construcción" en una arquitectura orientada a servicios, sino también cómo un escenario de Cucumber podemos probar dicha aplicación de manera determinista con pruebas que rompan la dependencia del servicio externo en el momento de la prueba.

- En el directorio raíz de la aplicación, escribimos **bundle exec autotest**.

```
guisa17@guisa17:~/Desktop/CC352-PC1/hw-sinatra-saas-wordguesser$ bundle exec autotest
(Not running features. To run features in autotest, set AUTOFEATURE=true.)
Loading autotest/rspec
'/home/guisa17/.rbenv/versions/2.6.6/bin/ruby' -rrubygems -S "/home/guisa17/.rbenv/versions/2.6.6/lib/ruby/gems/2.6.0/gems/rspec-core-3.3.2/exe/rspec" --tty "/home/guisa17/Desktop/CC352-PC1/hw-sinatra-saas-wordguesser/spec/wordguesser_game_spec.rb"
Run options: exclude {:pending=>true}

WordGuesserGame
  new
    takes a parameter and returns a WordGuesserGame object
  guessing
    correctly
      changes correct guess list
      returns true
    incorrectly
      changes wrong guess list
      returns true
    same letter repeatedly
      does not change correct guess list
      does not change wrong guess list
      returns false
      is case insensitive
    invalid
      throws an error when empty
      throws an error when not a letter
      throws an error when nil
  displayed word with guesses
    should be 'b-n-n-' when guesses are 'bn'
    should be '-----' when guesses are 'def'
    should be 'banana' when guesses are 'ban'
  game status
    should be win when all letters guessed
    should be lose after 7 incorrect guesses
    should continue play if neither win nor lose

Finished in 0.00744 seconds (files took 0.21927 seconds to load)
18 examples, 0 failures
```

Esto activará el framework Autotest, que busca varios archivos para determinar qué tipo de aplicación estamos probando y qué framework de prueba estamos usando. En este caso, descubriremos el archivo llamado rspec, que contiene opciones de RSpec e indica que estamos usando el framework de prueba de RSpec. Por lo tanto, Autotest buscará archivos de prueba en spec/ y los archivos de clase correspondientes en lib/.

Ahora, con Autotest aun ejecutándose, eliminamos `:pending => true` y guardamos el archivo. Deberíamos ver inmediatamente que Autotest se activa y vuelve a ejecutar las pruebas. Ahora deberíamos tener 18 ejemplos, 1 fallido y 17 pendientes.

```
wordguesser_game_spec.rb
1  require 'spec_helper'
2  require 'wordguesser_game'
3
4  describe WordGuesserGame do
5    # helper function: make several guesses
6    def guess_several_letters(game, letters)
7      letters.chars do |letter|
8        game.guess(letter)
9      end
10   end
11
12   describe 'new', :pending => true do
13     it "takes a parameter and returns a WordGuesserGame object" do
14       @game = WordGuesserGame.new('glorp')
15       expect(@game).to be_an_instance_of(WordGuesserGame)
16       expect(@game.word).to eq('glorp')
17       expect(@game.guesses).to eq('')
18       expect(@game.wrong_guesses).to eq('')
19     end
20   end
21 end
```

```
Run options: exclude {:pending=>true}

WordGuesserGame
  new
    takes a parameter and returns a wordguesserGame object (FAILED
- 1)

Failures:

  1) WordGuesserGame new takes a parameter and returns a WordGuesse
rGame object
     Failure/Error: expect(@game.word).to eq('glorp')
     NoMethodError:
       undefined method `word' for #<WordGuesserGame:0x000000000748
b230 @word="glorp">
     # ./spec/wordguesser_game_spec.rb:16:in `block (3 levels) in <
top (required)>'

Finished in 0.04653 seconds (files took 0.6954 seconds to load)
1 example, 1 failure

Failed examples:

rspec ./spec/wordguesser_game_spec.rb:13 # WordGuesserGame new take
s a parameter and returns a wordguesserGame object
```

El bloque describe 'new' significa "el siguiente bloque de pruebas describe el comportamiento de una 'nueva' instancia de WordGuesserGame". La línea WordGuesserGame.new hace que se cree una nueva instancia y las siguientes líneas verifican la presencia y los valores de las variables de instancia.

Esto es útil cuando queremos ejecutar las pruebas uno a uno

Preguntas

Según los casos de prueba, ¿cuántos argumentos espera el constructor de la clase de juegos (identifica la clase) y, por lo tanto, ¿cómo será la primera línea de la definición del método que debes agregar a `wordguesser_game.rb`?

Un argumento (en este ejemplo, "glorp"), y dado que los constructores en Ruby siempre se llaman `initialize`, la primera línea será algo como `def initialize(nueva_palabra)` o algo parecido

Según las pruebas de este bloque describe, ¿qué variables de instancia se espera que tenga `WordGuesserGame`?

Las variables de instancia son:

- `@word`
- `@guesses`
- `@wrong_guesses`

Para aprobar esta prueba fallida, deberemos crear getters y setters para las variables de instancia mencionadas en las pruebas de autoverificación anteriores.

Sugerencia: utilizar `attr_accessor`. Cuando hayamos hecho esto correctamente y hayamos guardado `wordguesser_game.rb`, la prueba automática debería activarse nuevamente y los ejemplos que antes fallaban ahora deberían pasar (verde).

Continuamos de esta manera, eliminando `:pending => true` de uno o dos ejemplos a la vez, avanzando hacia los specs, hasta que hayamos implementado todos los métodos de instancia de la clase de juego: `guess`, que procesa una adivinación y modifica el variables de instancia `wrong_guesses` y `guesses` en consecuencia, `check_win_or_lose`, que devuelve uno de los símbolos `:win`, `:lose` o `:play` dependiendo del estado actual del juego; y `word_with_guesses`, que sustituye las suposiciones correctas realizadas hasta el momento en la palabra.

Depuración

Al ejecutar pruebas, podemos insertar el comando `Ruby byebug` en el código de nuestra aplicación para ingresar al depurador de la línea de comandos e inspeccionar las variables, etc. Escribimos `h` para obtener ayuda en el mensaje de depuración. Escribe `c` para salir del depurador y continuar ejecutando su código.

Echa un vistazo al código del método de clase **`get_random_word`**, que recupera una palabra aleatoria de un servicio web que encontramos que hace precisamente eso. Utilizamos el siguiente comando para verificar que el servicio web funcione así. Lo ejecutamos varias veces para verificar que obtengamos palabras diferentes.

```
$ curl --data " http://randomword.sasbook.info/RandomWord
```

```
guisa17@guisa17:~/Desktop/CC352-PC1/hw-sinatra-saas-wordguesser$ curl --data '' http://randomword.sasbook.info/RandomWord
detailedguisa17@guisa17:~/Desktop/CC352-PC1/hw-sinatra-saas-wordguesser$
```

```

/omword.sasbook.info/RandomWord
erd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
0 47 0 -:--:-- -:--:-- -:--:-- 47possessive

/omword.sasbook.info/RandomWord
erd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
0 25 0 -:--:-- -:--:-- -:--:-- 25flavor

/omword.sasbook.info/RandomWord
erd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
0 31 0 -:--:-- -:--:-- -:--:-- 31married

/omword.sasbook.info/RandomWord
erd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
0 18 0 -:--:-- -:--:-- -:--:-- 18lonely

```

(--data es necesario para forzar a curl a realizar una POST en lugar de un GET. Normalmente, el argumento para --data serían los campos de formulario codificados, pero en este caso no se necesitan campos de formulario). Usar curl es una excelente manera de depurar interacciones con servicios externos. man curl para obtener (muchos) más detalles sobre esta poderosa herramienta de línea de comandos.

Parte 2: RESTful para Wordguesser

Comprender cómo exponer los comportamientos de tu aplicación como acciones RESTful en un entorno SaaS y cómo preservar el estado del juego en solicitudes HTTP (sin estado) a tu aplicación utilizando la abstracción de cookies proporcionada por el servidor de aplicaciones.

Qué haremos: crea una aplicación Sinatra que utilice la lógica de WordGuesserGame desarrollada en la parte anterior, permitiéndote jugar Wordguesser a través de un navegador.

A diferencia de una aplicación empaquetada, SaaS se ejecuta sobre el protocolo HTTP sin estado y cada solicitud HTTP provoca que suceda algo en la aplicación. Y como HTTP no tiene estado, debemos pensar detenidamente en las dos preguntas siguientes:

1. ¿Cuál es el estado total necesario para que la siguiente solicitud HTTP continúe el juego donde lo dejó la solicitud anterior?

Podemos utilizar la abstracción de cookies proporcionada por el servidor de aplicaciones. Las cookies son pequeños fragmentos de datos que se almacenan en el lado del cliente y se envían con cada solicitud HTTP subsiguiente. Podemos utilizar cookies para almacenar información relevante del juego, como la palabra secreta, las letras adivinadas, el número de intentos, etc.

2. ¿Cuáles son las diferentes acciones del juego y cómo deberían corresponderse las solicitudes HTTP con esas acciones?

Podemos asignar las acciones del juego a rutas y métodos HTTP de la siguiente manera:

- Crear un nuevo juego: Para comenzar un nuevo juego, podemos usar una solicitud POST a una ruta como /game o /start que inicie un nuevo juego y devuelva un identificador único para el juego. Podemos establecer este identificador como una cookie en la respuesta para mantener el estado del juego.
- Obtener el estado del juego: Podemos usar una solicitud GET a una ruta como /game/{game_id} para obtener el estado actual del juego. El game_id se puede pasar como un parámetro en la URL o como una cookie para identificar el juego en curso.
- Realizar una jugada: Para permitir que el jugador haga una jugada, podemos usar una solicitud POST a la ruta /game/{game_id}/guess donde game_id identifica el juego en curso. Debemos proporcionar la letra adivinada como parte de los datos de la solicitud. La lógica del juego verificará si la letra es correcta o incorrecta y actualizará el estado del juego en consecuencia.
- Finalizar un juego: Cuando el jugador adivina la palabra o agota los intentos, podemos usar una solicitud DELETE o POST a la ruta /game/{game_id} para finalizar el juego. Esto puede incluir mostrar el resultado (victoria o derrota) y eliminar cualquier estado de juego almacenado en las cookies.
- Ver historial de juegos: Si deseamos permitir a los usuarios ver un historial de juegos anteriores, podemos tener una ruta GET a /history o /games que devuelva una lista de juegos anteriores con sus resultados.

El mecanismo ampliamente utilizado para mantener el estado entre un navegador y un servidor SaaS es una cookie. El servidor puede poner lo que quiera en la cookie (hasta un límite de longitud de 4K bytes); el navegador promete enviar la cookie de regreso al servidor en cada solicitud posterior. Dado que el navegador de cada usuario obtiene su propia cookie, la cookie se puede utilizar de manera efectiva para mantener el estado por usuario.

En la mayoría de las aplicaciones SaaS, la cantidad de información asociada con la sesión de un usuario es demasiado grande para caber en los 4 KB permitidos en una cookie, por lo que, como veremos en Rails, la información de la cookie se usa más a menudo para contener un puntero que indica que vive en una base de datos. Pero para este ejemplo simple, el estado del juego es lo suficientemente pequeño como para que podamos mantenerlo directamente en la cookie de sesión.

Pregunta

Enumera el estado mínimo del juego que se debe mantener durante una partida de Wordguesser.

La palabra secreta; la lista de letras que han sido adivinadas correctamente; la lista de letras que han sido adivinadas incorrectamente. Convenientemente, la clase WordGuesserGame, bien estructurada, encapsula este estado utilizando sus variables de instancia, como recomienda un diseño orientado a objetos apropiado.

El juego como recurso RESTful

Pregunta

Enumera las acciones del jugador que podrían provocar cambios en el estado del juego.

Adivinar una letra: posiblemente modifica las listas de suposiciones correctas o incorrectas; posiblemente resulta en ganar o perder el juego.

Comenzar un nuevo juego: elige una nueva palabra y establece las listas de suposiciones incorrectas y correctas en vacío.

En una arquitectura orientada a servicios, no exponemos el estado interno directamente, en su lugar, exponemos un conjunto de solicitudes HTTP que muestran o realizan alguna operación en un recurso subyacente hipotético. La parte más complicada e importante del diseño RESTful es modelar cuáles son sus recursos y qué operaciones son posibles con ellos.

En este caso, podemos pensar en el juego en sí como el recurso subyacente. Hacerlo da como resultado algunas decisiones de diseño importantes sobre cómo las rutas (URL) se asignarán a las acciones y sobre el código del juego en sí.

Como ya hemos identificado el estado del juego y las acciones de los jugadores que podrían cambiarlo, tiene sentido definir el juego en sí como una clase. Una instancia de esa clase es un juego y representa el recurso que manipula la aplicación SaaS.

Asignación de rutas de recursos a solicitudes HTTP

La lista inicial de operaciones en el recurso podría verse así, donde también le hemos dado un nombre sugerente a cada acción:

1. create: crear un nuevo juego
2. show: Muestra el estado del juego actual

3. guess: Adivina una letra

Pregunta

Para un buen diseño RESTful, ¿cuáles de las operaciones de recursos deberían ser manejadas por HTTP GET y cuáles deberían ser manejadas por HTTP POST?

Las operaciones gestionadas con GET no deben tener efectos secundarios en el recurso, por lo que 'show' puede ser manejado mediante un GET, pero 'create' y 'guess' (que modifican el estado del juego) deben utilizar POST. (De hecho, en una arquitectura orientada a servicios real, también podemos optar por utilizar otros verbos HTTP como PUT y DELETE, pero no los abordaremos en esta tarea).

HTTP es un protocolo de solicitud-respuesta y el navegador web es fundamentalmente una interfaz de usuario de solicitud-respuesta, por lo que cada acción del usuario debe dar como resultado que el navegador muestre algo. Para la acción "mostrar estado del juego actual", está bastante claro que lo que debemos mostrar es la representación HTML del juego actual, como lo hace el método `word_with_guesses` de la clase de juego. (En una versión más elegante, que le recomendamos que pruebes, puedes agregar tus propias animaciones y gráficos personalizados a la página HTML).

Pero cuando el jugador adivina una letra, ya sea que la adivinación sea correcta o no, ¿cuál debería ser la "representación HTML" del resultado de esa acción?

La respuesta a esta pregunta es donde falla el diseño de muchas aplicaciones web.

En términos de juego, probablemente tenga más sentido es que, después de que el jugador envíe una adivinación, se muestre el estado resultante de la adivinación. Pero ya tenemos una acción RESTful para mostrar el estado del juego. Entonces podemos planear usar una redirección HTTP para hacer uso de esa acción.

Esta es una distinción importante, porque una redirección HTTP desencadena una solicitud HTTP completamente nueva. Dado que esa nueva solicitud no "sabe" qué letra se adivinó, toda la responsabilidad de cambiar el estado del juego está asociada con la acción RESTful de adivinar una letra, y toda la responsabilidad de mostrar el estado actual del juego sin cambiar está asociado con la acción de visualización del estado. Esto es bastante diferente de un escenario en el que la acción de adivinar una letra también muestra el estado del juego, porque en ese caso, la acción de visualización del juego tendría acceso a la letra adivinada. Un buen diseño RESTful mantendrá estas responsabilidades separadas, de modo que cada acción RESTful haga exactamente una cosa.

Un argumento similar se aplica a la acción de crear un nuevo juego. La responsabilidad de crear un nuevo objeto de juego recae en esa acción, pero una vez que se crea el nuevo objeto del juego, ya tenemos una acción para mostrar el estado actual del juego.

Entonces podemos comenzar a mapear las acciones RESTful en términos de solicitudes HTTP de la siguiente manera, usando algunos URI simples para las rutas:

- GET /show (ruta y acción) muestra el estado del juego (operación de recursos) muestra las adivinaciones correctas e incorrectas hasta el momento (resultado web)
- POST /guess (ruta y acción) actualiza el estado del juego con una nueva redirección de letra adivinada (operación de recursos) redirecciona a show (resultado web)
- POST /create (ruta y acción) crea un nuevo juego (operación de recursos) redirecciona a show (resultado web)

En una verdadera arquitectura orientada a servicios, casi habríamos terminado. Pero un sitio experimentado a través de un navegador web no es una verdadera arquitectura orientada a servicios.

¿Por qué? Porque un usuario web humano necesita una forma de POST un formulario. Se puede lograr un GET simplemente escribiendo una URL en la barra de direcciones del navegador, pero un POST solo puede ocurrir cuando el usuario envía un formulario HTML (o, como veremos más adelante, cuando el código AJAX en JavaScript activa una acción HTTP).

Entonces, para comenzar un nuevo juego, en realidad debemos proporcionarle al usuario una forma de publicar el formulario que activará la acción POST/create. Puedes pensar en el recurso en cuestión como "la oportunidad de crear un nuevo juego". Entonces podemos agregar otra fila a nuestra tabla de rutas:

- GET /new te da al usuario humano la oportunidad de iniciar un nuevo juego. Muestra un formulario que incluye un botón "iniciar un nuevo juego".

De manera similar, ¿cómo genera el usuario humano el POST para adivinar una nueva letra? Dado que ya tenemos una acción para mostrar el estado actual del juego (show), sería fácil incluir en esa misma página HTML un formulario de "adivina una letra" que, cuando se envía, genera la acción POST /guess.

Veremos este patrón reflejado más adelante en Rails: un recurso típico (como la información sobre un jugador) tendrá operaciones de create y update, pero para permitir que un ser humano proporcione los datos utilizados para crear o actualizar un registro de jugador, Debes proporcionar acciones new y edit respectivamente que permitan al usuario ingresar la información en un formulario HTML.

Preguntas

¿Por qué es apropiado que la nueva acción utilice GET en lugar de POST?

La acción "new" no altera el estado del sistema por sí misma, sino que proporciona un formulario que permite al jugador realizar una acción posterior

Explica por qué la acción GET /new no sería necesaria si tu juego Wordguesser fuera llamado como un servicio en una verdadera arquitectura orientada a servicios.

La acción "new" no es necesaria para el funcionamiento del servicio en sí, sino que se utiliza para facilitar a los usuarios humanos de la web la generación de solicitudes HTTP POST al servicio Wordguesser.

Por último, cuando el juego termina (ganemos o perdamos), no deberíamos aceptar más adivinaciones. Como planeamos que show incluya un formulario para adivinar letras, tal vez deberíamos tener otro tipo de acción show cuando el juego termine, uno que no incluya una forma para que el jugador adivine una letra, pero (tal vez) incluye un botón para iniciar un nuevo juego. Incluso podemos tener páginas separadas para ganar y perder, las cuales le dan al jugador la oportunidad de comenzar un nuevo juego. Dado que la acción show ciertamente puede indicar si el juego ha terminado, puedes redirigir condicionalmente a la acción de ganar o perder cuando se solicita.

Las rutas para cada una de las acciones RESTful del juego, según la descripción de lo que debe hacer la ruta:

- Mostrar el estado del juego, permitir que el jugador ingrese su adivinación. Puede redirigir Win o Lose --> GET /show
- Mostrar formulario que puede generar POST/create --> GET/new
- Empieza un juego nuevo: redirige a Show game después de cambiar el estado --> POST /create
- Adivinación del proceso: redirige a Show game después de cambiar el estado --> POST /guess
- Muestra la página "you win" con el botón para iniciar un nuevo juego --> GET/win
- Muestra la página "you lose" con el botón para iniciar un nuevo juego --> GET/lose

Parte 3: Conexión de WordGuesserGame a Sinatra

Ya conociste a Sinatra. Estas son las novedades del esqueleto de la aplicación Sinatra app.rb que proporcionamos para Wordguesser:

- before do...end es un bloque de código ejecutado antes de cada solicitud de SaaS
- after do...end se ejecuta después de cada solicitud de SaaS
- Las llamadas erb : action hacen que Sinatra busca el archivo views/action .erb y los ejecute a través del procesador Ruby integrado, que busca construcciones <%= like this %>, ejecuta el código Ruby interno y sustituye el resultado. El código se ejecuta en el mismo contexto que la llamada a erb, por lo que el código puede "ver" cualquier variable de instancia configurada en los bloques get o post.

Pregunta

@game en este contexto es una variable de instancia de qué clase?

Es una variable de instancia de la clase WordGuesserApp en el archivo app.rb. Recuerda que estamos tratando con dos clases Ruby aquí: la clase WordGuesserGame encapsula la lógica del juego en sí (es decir, el Modelo en el modelo-vista-controlador), mientras que WordGuesserApp encapsula la lógica que nos permite ofrecer el juego como un servicio SaaS (puedes pensar en ello aproximadamente como la lógica del Controlador más la capacidad de renderizar las vistas mediante erb).

La sesión

Ya identificamos los elementos necesarios para mantener el estado del juego y los encapsulamos en la clase del juego. Dado que HTTP no tiene estado, cuando llega una nueva solicitud HTTP, no existe la noción del "juego actual". Por lo tanto, lo que debemos hacer es guardar el objeto del juego de alguna manera entre solicitudes.

Si el objeto del juego fuera grande, probablemente lo almacenaríamos en una base de datos en el servidor y colocaríamos un identificador para el registro correcto de la base de datos en la cookie. (De hecho, como veremos, esto es exactamente lo que hacen las aplicaciones Rails). Pero como el estado del juego es pequeño, podemos poner todo en la cookie. La librería session de Sinatra nos permite hacer esto: en el contexto de la aplicación Sinatra, cualquier cosa que coloquemos en el hash "mágico" session[] se conserva en todas las solicitudes. De hecho, los objetos colocados allí se serializan en un formato compatible con texto que se conserva para nosotros. Este comportamiento se activa mediante la llamada de Sinatra enable :sessions en app.rb.

Hay otro objeto similar a una sesión que usaremos. En algunos de los casos anteriores, una acción realizará algún cambio de estado y luego redirigirá a otra acción, como cuando la acción Guess (activada por POST /guess) redirige a la acción Show (GET /show) para volver a mostrar el estado del juego después de cada adivinación. Pero ¿qué pasa si la acción Guess quiere mostrar un mensaje al jugador, como por ejemplo para informarle que ha repetido erróneamente una adivinación? El problema es que, dado que cada solicitud no tiene estado, necesitamos

enviar ese mensaje "a través" de la redirección, del mismo modo que necesitamos preservar el estado del juego "a través" de las solicitudes HTTP.

Para hacer esto, usamos la gema sinatra-flash, que puedes ver en el Gemfile. `flash[]` es un hash para recordar mensajes cortos que persisten hasta la siguiente solicitud (normalmente una redirección) y luego se borran.

Pregunta

¿Por qué esto ahorra trabajo en comparación con simplemente almacenar esos mensajes en el hash de sesión []?

Cuando colocamos algo en `session[]` permanece allí hasta que lo eliminamos. El caso común para un mensaje que debe sobrevivir a una redirección es que solo debe mostrarse una vez; `flash[]` incluye la funcionalidad adicional de borrar los mensajes después de la próxima solicitud.

Ejecutando la aplicación Sinatra

Como antes, ejecuta el comando `bundle exec rackup --host 0.0.0.0 --port 3000` para iniciar la aplicación, o `bundle exec rerun -- rackup --host 0.0.0.0 --port 3000` si deseas volver a ejecutar la aplicación cada vez que realizas un cambio de código.

Pregunta

Según el resultado de ejecutar este comando, ¿cuál es la URL completa que debes visitar para visitar la página New Game?

El código Ruby `get '/new'` do... en `app.rb` renderiza la página de Nuevo Juego, por lo que la URL completa tiene el formato `http://localhost:3000/new`.

Debemos visitar esta URL y verifica que aparezca la página Iniciar New Game.

Pregunta

¿Dónde está el código HTML de esta página?

Se encuentra en `views/new.erb`, y se convierte en HTML mediante la directiva `erb :new`

Verifica que cuando haces clic en el botón New Game obtienes un error. Esto se debe a que deliberadamente dejamos incompleto el `<form>` que encierra este botón: no hemos especificado dónde debe publicarse el formulario. Lo haremos a continuación, pero lo haremos mediante pruebas.

Pero primero, instalemos la aplicación en Heroku. En realidad, este es un paso crítico. Necesitamos asegurarnos de que la aplicación se ejecute en heroku antes de comenzar a realizar cambios significativos.

- Primero, ejecuta `bundle install` para asegurarse de que `Gemfile` y `Gemfile.lock` estén sincronizados.
- A continuación, escriba `git add.` para preparar todos los archivos modificados (incluido `Gemfile.lock`)
- Luego escribe `git commit -m "¡Listo para Heroku!"` para confirmar todos los cambios locales.
- A continuación, escribe `heroku login -i` y autentícase.
- Dado que esta es la primera vez que le contamos a Heroku sobre la aplicación `Wordguesser`, debemos escribir `heroku create` para que Heroku se prepare para recibir el código y crear un `git` remoto para hacer referencia al nuevo repositorio remoto. (Ejecuta `git remote -v` si tienes curiosidad).
- Luego, escribe `git push heroku master` para enviar tu código a Heroku.
- Cuando quieras actualizar Heroku más tarde, solo necesitas confirmar tus cambios en `git` localmente y luego enviarlos a Heroku como en el último paso.
- Verifica que `Wordguesser` implementado por Heroku se comporte igual que tu versión de desarrollo antes de continuar. Unas pocas líneas desde la parte inferior de la salida de Heroku en la terminal deberían tener una URL que termine en `herokuapp.com`, pero es más fácil simplemente escribir `heroku info`. Busca eso, cópielo en el portapapeles y pégalo en una pestaña del navegador para ver la aplicación actual. (Como alternativa, ejecuta `heroku open`).
- Verifica la funcionalidad haciendo clic en el botón `new game`.

Parte 4: Cucumber

A continuación, usaras Cucumber para impulsar la creación del código SaaS.

Normalmente, el ciclo sería:

- Utilizar el escenario Cucumber para expresar el comportamiento de un extremo a otro de un escenario
- Cuando comiences a escribir el código para realizar cada paso del escenario, usa `RSpec` para impulsar la creación del código de ese paso.
- Repite hasta que todos los pasos del escenario pasen a color verde.

Preguntas

Lea la sección sobre " Using Capybara with Cucumber" en la página de inicio de Capybara. ¿Qué pasos utiliza Capybara para simular el servidor como lo haría un navegador? ¿Qué pasos utiliza Capybara para inspeccionar la respuesta de la aplicación al estímulo?

Las definiciones de pasos que emplean ciertas acciones como `visit`, `click_button` y `fill_in` están emulando el comportamiento de un navegador al interactuar con una página web y sus formularios, mientras que aquellas que utilizan `have_content` están verificando el contenido generado por la página.

Mirando `features/guess.feature`, ¿cuál es la función de las tres líneas que siguen al encabezado "Feature:"?

Son comentarios que muestran el propósito y los actores de esta historia. Cucumber no los ejecutará.

¿En el mismo archivo, observando el paso del escenario `Given I start a new game with word "garply"` qué líneas en `game_steps.rb` se invocarán cuando Cucumber intente ejecutar este paso y cuál es el papel de la cadena `"garply"` en el paso?

Las líneas 13 a 16 serán ejecutadas porque la palabra `'word'` coincide con el primer grupo de captura de paréntesis en la expresión regular, que en este caso es `'garply'`

Haz que pase tu primer escenario

Primero lograremos que pase el escenario `"I start a new game"`; Luego usaremos las mismas técnicas para hacer pasar los otros escenarios, completando así la aplicación. Así que debemos echar un vistazo a la definición del paso `"I start a new game with word."`.

Ya vimos que podemos cargar la página del nuevo juego, pero aparece un error al hacer clic en el botón para crear un nuevo juego. Ahora reproduciremos este comportamiento con un escenario Cuke.

Pregunta

Cuando el "simulador de navegador" en Capybara emite la solicitud de `visit '/new'`, Capybara realizará un HTTP GET a la URL parcial `/new` en la aplicación. ¿Por qué crees que `visit` siempre realiza un GET, en lugar de dar la opción de realizar un GET o un POST en un paso determinado?

Cucumber/Capybara se adhiere a las acciones que un usuario humano puede realizar en un navegador web, y la forma en que se envía una solicitud POST es a través de la acción `click_button` cuando se envía un formulario HTML

Ejecute el escenario del `"new game"` con:

```
$ cucumber features/start_new_game.feature
```

Si recibimos un error sobre Cucumber como este, simplemente seguiremos los consejos y ejecutaremos el bundle install primero.

```
~/workspace/hw-sinatra-saas-wordguesser (master) $ cucumber
features/start_new_game.feature Could not find proper version of cucumber (2.0.0) in any of the
sources Run `bundle install` to install missing gems.
```

El escenario falla porque la etiqueta <form> en views/new.erb es incorrecta y está incompleta en la información que le dice al navegador en qué URL publicar el formulario. Según la tabla de rutas que desarrollamos en una sección anterior, completa los atributos de la etiqueta <form> adecuadamente. Puedes inspeccionar lo que sucede con varias rutas en app.rb, pero no necesitas editar este archivo todavía. (Pista: si te quedas atascado, echa un vistazo a show.erb (en la parte inferior) para ver un ejemplo similar de una etiqueta de formulario completa).

El código para crear un nuevo juego en la aplicación Sinatra debería hacer lo siguiente:

- Llama al método de clase WordGuesserGame get_random_word
- Crea una nueva instancia de WordGuesserGame usando esa palabra
- Redirige el navegador a la acción show

Veamos cómo se actualizan estos pasos en el archivo app.rb en la ruta post /create do.

Ahora prepararemos y confirmaremos todos los archivos localmente, luego haremos clic en git push heroku master para implementarlo en Heroku nuevamente y verificar manualmente este comportamiento mejorado.

Pregunta

¿Cuál es el significado de usar Given versus When versus Then en el archivo de características? ¿Qué pasa si los cambias? Realiza un experimento sencillo para averiguarlo y luego confirme los resultados utilizando Google.

Las palabras clave son todas alias para el mismo método.Cuál de ellas uses se determina por lo que haga que el escenario sea más legible.

Desarrollar el escenario para adivinar una letra

Para este escenario, en `features/guess.feature`, ya proporcionamos un archivo HTML `show.erb` correcto que envía la adivinación del jugador a la acción `guess`. Ya tenemos un método de instancia `WordGuesserGame#guess` que tiene la funcionalidad necesaria.

Pregunta

En `game_steps.rb`, mira el código del paso "I start a new game..." y, en particular, el comando `stub_request`. Dada la pista de que ese comando lo proporciona una gema (biblioteca) llamada `webmock`, ¿qué sucede con esa línea y por qué es necesaria? (Utiliza Google si es necesario).

`Webmock` permite que nuestras pruebas controlen las solicitudes HTTP generadas por nuestra aplicación hacia otro servicio. En este caso, se está interceptando la solicitud POST y se está simulando la respuesta para que nuestros tests sean predecibles. Esto evita que nuestras pruebas accedan al servidor externo real cada vez que se ejecutan.

Pregunta

En tu código Sinatra para procesar una adivinación, ¿qué expresión usaría para extraer *solo el primer carácter* de lo que el usuario escribió en el campo de adivinación de letras del formulario en `show.erb`?

`params[:guess].to_s[0]` es una expresión que se utiliza para obtener el primer carácter de la entrada proporcionada en el campo de formulario `'guess'`. Si el campo está en blanco (o es `'nil'`), se convierte en una cadena vacía y se obtiene el primer carácter, que también sería una cadena vacía en este caso.

Si el usuario no escribió nada, no habrá ninguna clave coincidente en `params[]`, por lo que al eliminar la referencia al campo del formulario se obtendrá cero. En ese caso, nuestro código debería devolver una cadena vacía en lugar de un error.

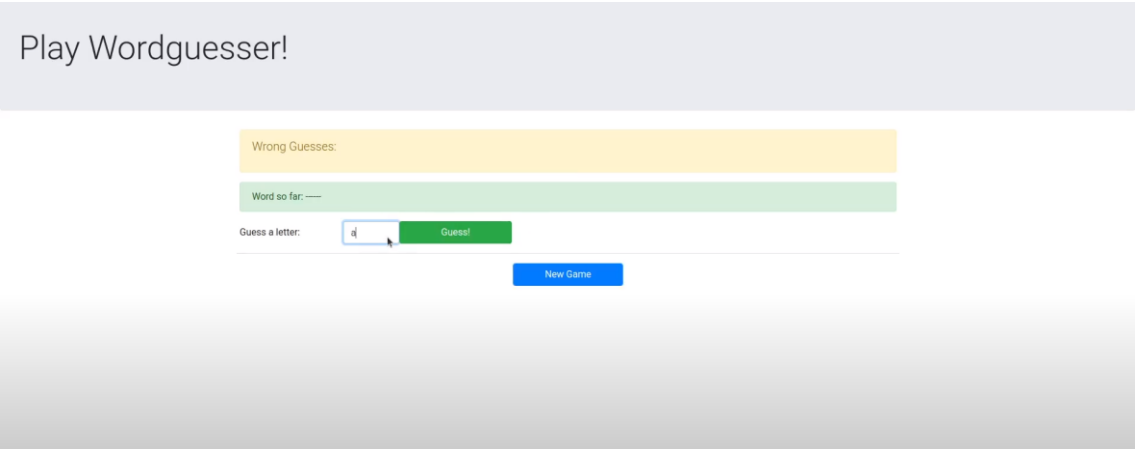
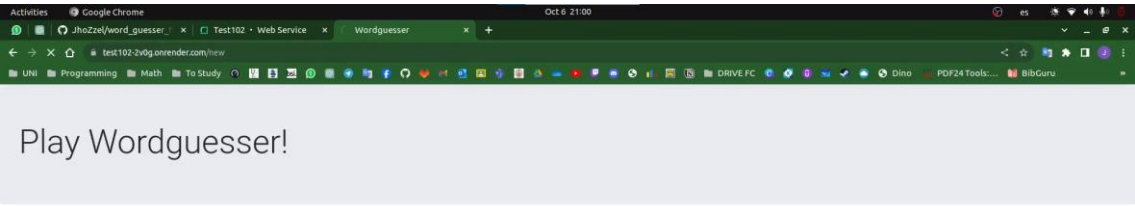
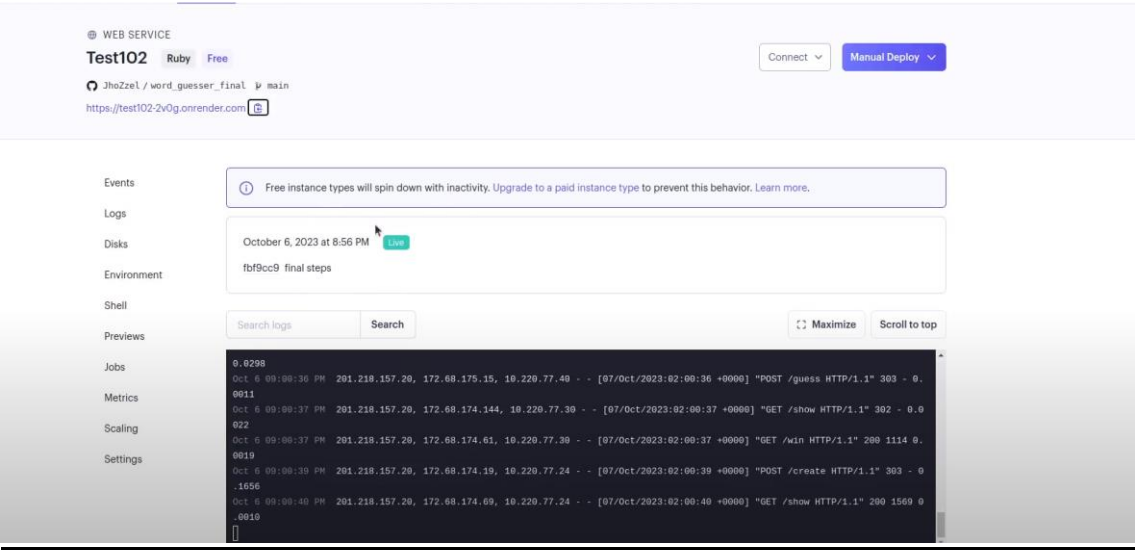
En el código `guess` en el archivo Sinatra `app.rb`, debes:

- Extrae la letra enviada en el formulario. (dado arriba y en el código)
- Utiliza esa letra para adivinar el juego actual. (agrega este código)
- Redirige a la acción `show` para que el jugador pueda ver el resultado de su adivinación.

El comando de Capybara `save_and_open_page` colocado en una definición de paso hará que el paso abra una ventana del navegador web que muestra cómo se ve la página en ese punto del

escenario. La funcionalidad la proporciona en parte una gema llamada launchy que se encuentra en Gemfile.

Despliegue final de la aplicación bajo la plataforma en render



Parte 5: Otros casos

A estas alturas ya deberías estar familiarizado con el ciclo de la actividad:

1. Escoge un nuevo escenario para trabajar
2. Ejecuta el escenario y observa cómo falla
3. Desarrollar código que haga pasar cada paso del escenario.
4. Repita hasta que pasen todos los pasos.

Mientras juegas, ¿qué sucede si agregas /win directamente al final de la URL de tu aplicación? Asegúrate de que el jugador no pueda hacer trampa simplemente visitando GET /win. Considera cómo modificar las acciones de win, lose y show para evitar este comportamiento.

Asegúrate de que se aprueben todos los escenarios de Cucumber. Una forma abreviada de ejecutarlos todos es `cucumber features/` que ejecuta todos los archivos `.feature` en el directorio dado.

Instrucciones de envío

Cuando todos los escenarios hayan pasado, implementa en Heroku y envía la URL de tu juego implementado en un archivo llamado `sinatra-url.txt`.

Puedes crear tu archivo de texto usando el comando `echo` de esta manera:

```
echo 'my-app-12345.herokuapp.com' > sinatra-url.txt
```

Si ejecutas el comando `cat` después de crear el archivo de esta manera:

```
cat sinatra-url.txt
```

entonces la salida debería verse así:

```
my-app-12345.herokuapp.com
```

Por supuesto, cambiarías `'my-app-12345'` para que coincida con tu URL de Heroku.

Por último, visitamos la misma URL que ingresamos en el archivo de texto con nuestro navegador web para asegurarnos de que nuestra aplicación se esté ejecutando correctamente en esa dirección antes de enviar el archivo de texto.

CONCLUSIÓN

El desarrollo dirigido por pruebas (TDD) nos permitirá escribir pruebas mucho más detalladas para nuestro código y determinar su cobertura, es decir, qué tan exhaustivamente tus pruebas evalúan nuestro código. Utilizaremos RSpec para realizar un desarrollo orientado a pruebas, en el que escribimos las pruebas antes de escribir el código, observamos que la prueba falla ("rojo"), escribimos rápidamente la cantidad justa de código para que la prueba pase ("verde"), mejoramos (refactorizamos) el código y pasamos a la siguiente prueba. Utilizaremos la herramienta autotest para ayudarnos a entrar en un ritmo de rojo-verde-refactorizar.

Las métricas de código nos darán información sobre la calidad de nuestro código: ¿es conciso? ¿Está estructurado de manera que minimice el costo de realizar cambios y mejoras? ¿Una clase en particular intenta hacer demasiado (o muy poco)? Utilizaremos CodeClimate (entre otras herramientas) para ayudarnos a entender las respuestas. Podemos verificar tanto métricas cuantitativas, como la cobertura de pruebas y la complejidad de un método único, como métricas cualitativas, como la adherencia a los Principios SOLID de diseño orientado a objetos.

La refactorización significa modificar la estructura de tu código para mejorar su calidad (mantenibilidad, legibilidad, modificabilidad) mientras se conserva su comportamiento. Aprenderemos a identificar antipatrones, señales de advertencia de la calidad deteriorada en tu código, y oportunidades para corregirlos, a veces aplicando patrones de diseño que han surgido como "plantillas" que capturan una solución eficaz para una clase de problemas similares.