

TALLER IS 2023

Actividad 1



Integrantes:

Buil Delfina
Peruchin Ivan
López Agustín

Funcionalidades pendientes

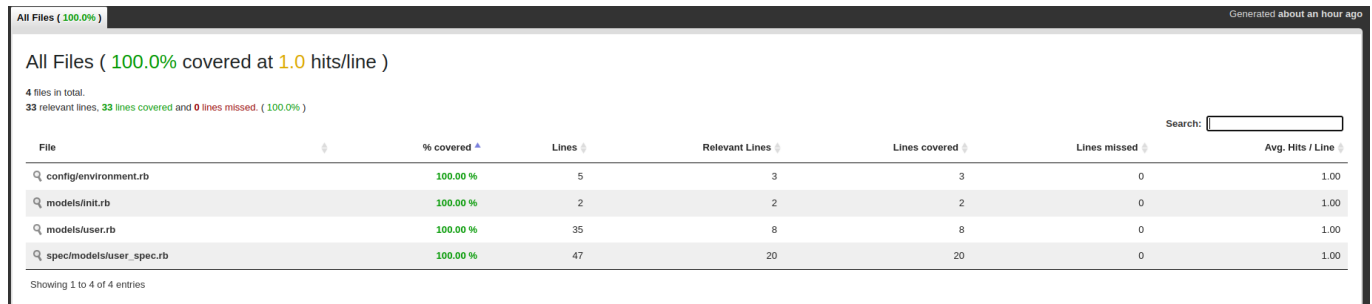
Desde el equipo creemos que no nos quedó ninguna funcionalidad importante pendiente por hacer, de las que habíamos planteado anteriormente, en el taller de la materia AYDS2023. Únicamente quedaron funcionalidades secundarias por realizar, aquellas que habíamos planteado como un agregado a la jugabilidad de la aplicación. Algunas de ellas fueron: Eco Store (tienda de skins y diversos atributos para comprar), nivel máster (último nivel alcanzable), racha del jugador, entre otras.

Tests realizados

Al finalizar el taller de AYDS 2023, nos encontramos sin haber podido realizar casi ningún test sobre nuestro proyecto. Por lo tanto, al correr la suite de test, ningún modelo era tomado con falta de cobertura.

En la siguiente imagen se puede observar este suceso:

SimpleCov



The screenshot shows the SimpleCov web interface. At the top, it says 'All Files (100.0%)' and 'Generated about an hour ago'. Below that, a summary states 'All Files (100.0% covered at 1.0 hits/line)' and '4 files in total, 33 relevant lines, 33 lines covered and 0 lines missed. (100.0%)'. A search bar is on the right. The main table lists the files and their coverage details.

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
config/environment.rb	100.00 %	5	3	3	0	1.00
models/init.rb	100.00 %	2	2	2	0	1.00
models/user.rb	100.00 %	35	8	8	0	1.00
spec/models/user_spec.rb	100.00 %	47	20	20	0	1.00

Showing 1 to 4 of 4 entries

Se puede observar la ejecución de los tests antes de comenzar a agregar nuevos. Muestra 100% de cobertura pero no representa la realidad, ya que nuestro proyecto no había sido testeado aún.

En esta materia, Ingeniería de Software 2023, al comenzar a agregar test unitarios para nuestros modelos empezamos a ganar robustez.

```

Answer
  validations
    requires an option
    requires a user id

Option
  validations
    requires a description
    requires an isCorrect field

Question
  validations
    requires a description
    requires a detail

User
  validations
    requires a username
    requires a password
    requires non-negative points
    requires a valid email format
    requires a minimum length for username and password
    requires a valid email format
    requires a password with at least 6 characters
    when you win a point, only one point is gained

Finished in 0.08547 seconds (files took 1.05 seconds to load)
14 examples, 0 failures

```

Captura de pantalla del resultado obtenido al ejecutar los nuevos tests realizados.

En nuestro caso, decidimos comenzar por el modelo de usuario, ya que entendemos que es uno de los modelos más importantes de nuestro proyecto. Las siguientes imágenes dan a conocer los tests implementados sobre este modelo, los cuales fueron enfocados en los distintos atributos que lo componen:

```

4
5 describe 'User' do
6   describe 'validations' do
7     it 'requires a username' do
8       user = User.new(username: '', password: '123', email: 'test@example.com', birthdate: '1999-12-12')
9       expect(user.valid?).to eq(false)
10      expect(user.errors[:username]).to include("can't be blank")
11    end
12
13    it 'requires a password' do
14      user = User.new(username: 'usertest', password: '', email: 'test@example.com', birthdate: '1999-12-12')
15      expect(user.valid?).to eq(false)
16      expect(user.errors[:password]).to include("can't be blank")
17    end
18
19    it 'requires non-negative points' do
20      user = User.new(username: 'usertest', password: '123', email: 'test@example.com', birthdate: '1999-12-12', points: -5)
21      expect(user.valid?).to eq(false)
22      expect(user.errors[:points]).to include("must be greater than or equal to 0")
23    end
24  end
25 end

```

```

24
25   it 'requires a valid email format' do
26     user = User.new(username: 'usertest', password: '123', email: 'invalid_email', birthdate: '1999-12-12')
27     expect(user.valid?).to eq(false)
28     expect(user.errors[:email]).to include("invalid email format")
29   end
30
31   it 'requires a minimum length for username and password' do
32     user = User.new(username: 'ab', password: '123', email: 'test@example.com', birthdate: '1999-12-12')
33     expect(user.valid?).to eq(false)
34     expect(user.errors[:username]).to include("username must be at least 3 characters")
35     expect(user.errors[:password]).to include("password must be at least 6 characters")
36   end
37
38   it 'requires a valid email format' do
39     user = User.new(username: 'usertest', password: '123', email: 'invalid_email', birthdate: '1999-12-12')
40     expect(user.valid?).to eq(false)
41     expect(user.errors[:email]).to include("invalid email format")
42   end
43
44   it 'requires a password with at least 6 characters' do
45     user = User.new(username: 'usertest', password: 'abcde', email: 'test@example.com', birthdate: '1999-12-12')
46     expect(user.valid?).to eq(false)
47     expect(user.errors[:password]).to include("password must be at least 6 characters")
48   end

```

El modelo User cuenta con muchos atributos que deben ser validados a la hora de crear una nueva instancia, por lo tanto, es interesante controlar mediante tests que dicha validación se esté realizando correctamente.

Realizamos, por cada validación requerida para la creación de un usuario, un test unitario, como se pudo observar en las imágenes anteriores. A continuación se presenta el modelo User con sus distintos atributos que fueron testeados:

```

class User < ActiveRecord::Base
  has_many :questions
  has_many :answers, through: :questions

  validates :username, presence: true, length: { minimum: 3, message: "username must be at least 3 characters" }
  validates :password, presence: true, length: { minimum: 6, message: "password must be at least 6 characters" }
  validates :email, presence: true, format: { with: /\A[\w+\-\.]+\@[a-z\d\-\.\.]+\.[a-z]+\z/i, message: "invalid email format" }
  validates :birthdate, presence: true
  validates :points, presence: true
  validate :points_non_negative

```

Captura de las validaciones que creamos necesarias que debe tener un usuario

Obtenemos entonces el siguiente análisis de la cobertura:

All Files (100.0%) Generated less than a minute ago

All Files (100.0% covered at 1.07 hits/line)

10 files in total.
103 relevant lines, 103 lines covered and 0 lines missed. (100.0%)

Search:

File	% covered [▲]	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
config/environment.rb	100.00 %	5	3	3	0	1.00
models/answer.rb	100.00 %	7	5	5	0	1.00
models/init.rb	100.00 %	5	5	5	0	1.00
models/option.rb	100.00 %	14	8	8	0	1.13
models/question.rb	100.00 %	7	5	5	0	1.00
models/user.rb	100.00 %	30	13	13	0	1.46
spec/models/answer_spec.rb	100.00 %	16	10	10	0	1.00
spec/models/option_spec.rb	100.00 %	18	11	11	0	1.00
spec/models/question_spec.rb	100.00 %	16	10	10	0	1.00
spec/models/user_spec.rb	100.00 %	57	33	33	0	1.00

Showing 1 to 10 of 10 entries

Donde se puede visualizar un gran cambio y avance en comparación al primer análisis de cobertura realizado.

Luego, analizando nuestro código, creímos prudente realizar métodos que son propios del usuario, dentro del modelo, para que así estos no se encuentren en el `server.rb`.

Por ejemplo, el usuario suma puntos por cada pregunta contestada correctamente, por lo que se agregó un método simple para actualizar el atributo “points”, en donde se le añade el nuevo punto obtenido. Esta operación estaba siendo realizada dentro del `server.rb`, donde no resulta la mejor ubicación. Con este nuevo cambio, únicamente las instancias del modelo `User` podrán hacer uso del mismo.

```
13
14     def suma_points
15         self.points += 1
16         save
17     end
18
19     private
20
```

Esto nos permite llamarlo desde la lógica del servidor y hacer un test que controle que únicamente un solo punto sea añadido al puntaje total por cada respuesta correcta, y no una cantidad diferente.

Antes de testear este nuevo método agregado, si observamos el resultado que arroja SimpleCov, se puede determinar que la cobertura disminuyó, dejando esta de estar al 100%. Esto se atribuye a que el suma points no ha sido testeado todavía.

All Files (98.11%)
Generated less than a minute ago

All Files (98.11% covered at 1.05 hits/line)

10 files in total.

106 relevant lines, 104 lines covered and 2 lines missed. (98.11%)

Search:

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
models/user.rb	87.50 %	30	16	14	2	1.25
config/environment.rb	100.00 %	5	3	3	0	1.00
models/answer.rb	100.00 %	7	5	5	0	1.00
models/init.rb	100.00 %	5	5	5	0	1.00
models/option.rb	100.00 %	14	8	8	0	1.13
models/question.rb	100.00 %	7	5	5	0	1.00
spec/models/answer_spec.rb	100.00 %	16	10	10	0	1.00
spec/models/option_spec.rb	100.00 %	18	11	11	0	1.00
spec/models/question_spec.rb	100.00 %	16	10	10	0	1.00
spec/models/user_spec.rb	100.00 %	57	33	33	0	1.00

Showing 1 to 10 of 10 entries

Agregando dicho test...

```
50   it 'when you win a point, only one point is gained' do
51     user = User.new(username: 'usertest', password: 'abcde', email: 'test@example.com', birthdate: '1999-12-12', points: 5)
52     user.suma_points
53     expect(user.points).to eq(6)
54   end
55
56   end
57 end
```

Volvemos a obtener el 100% de cobertura:

All Files (100.0%) Generated less than a minute a

All Files (100.0% covered at 1.07 hits/line)

10 files in total.
110 relevant lines, 110 lines covered and 0 lines missed (100.0%)

File	% covered ^h	Lines ^g	Relevant Lines ^g	Lines covered ^g	Lines missed ^g	Avg. Hits / Line ^g
config/environment.rb	100.00 %	5	3	3	0	1.00
models/answer.rb	100.00 %	7	5	5	0	1.00
models/init.rb	100.00 %	5	5	5	0	1.00
models/option.rb	100.00 %	14	8	8	0	1.13
models/question.rb	100.00 %	7	5	5	0	1.00
models/user.rb	100.00 %	30	16	16	0	1.44
spec/models/answer_spec.rb	100.00 %	16	10	10	0	1.00
spec/models/option_spec.rb	100.00 %	18	11	11	0	1.00
spec/models/question_spec.rb	100.00 %	16	10	10	0	1.00
spec/models/user_spec.rb	100.00 %	57	37	37	0	1.00

Showing 1 to 10 of 10 entries

Luego, continuamos con el modelo de preguntas (Question), en el cual validamos dos atributos importantes: que cada pregunta posea una descripción y un detalle, el cual representa un dato curioso de esa pregunta. Los tests realizados en base a esto, corroborar que si una Question no posee una description o un detail, entonces esta es considerada inválida.

```
4   describe 'Question' do
5     describe 'validations' do
6       it 'requires a description' do
7         question = Question.create(description: '', detail: 'detail test' )
8         expect(question.valid?).to eq(false)
9       end
10
11       it 'requires a detail' do
12         question = Question.create(description: 'description test', detail: '' )
13         expect(question.valid?).to eq(false)
14       end
15     end
16   end
```

Test de preguntas

Para el modelo de opciones (Option), validamos que cada una posea una descripción (description) y que además indique si la misma es correcta o no (isCorrect). Para ello hicimos 2 tests, el primero que determina como inválida a una opción si esta no posee una descripción, y el segundo, aquel que dada una opción sin un valor en el atributo isCorrect, también la asume como inválida.

```
describe 'Option' do
  describe 'validations' do
    it 'requires a description' do
      option = Option.new(description: '', isCorrect: true)
      expect(option.valid?).to eq(false)
    end

    it 'requires an isCorrect field' do
      option = Option.new(description: 'description test', isCorrect: nil)
      expect(option.valid?).to eq(false)
      expect(option.errors[:isCorrect]).to include("isCorrect must be true or false")
    end
  end
end
```

Test de opciones

De igual manera, testeamos las respuestas. Una respuesta será válida si contiene el id de una opción asociada, la cual es la opción que el usuario eligió y el id del usuario que eligió esa respuesta.

Creamos dos tests para probar esto:

Uno que recibe una respuesta con un option_id inválido pero con user_id válido, donde se considera inválida a esa respuesta, y el otro que de igual manera, dado un option_id pero sin un user_id, esta respuesta tampoco será evaluada como válida.

```
4 describe 'Answer' do
5   describe 'validations' do
6     it 'requires an option' do
7       answer = Answer.new(user_id: '1', option_id: ' ')
8       expect(answer.valid?).to eq(false)
9     end
10
11    it 'requires a user id' do
12      answer = Answer.new(user_id: ' ', option_id: '2')
13      expect(answer.valid?).to eq(false)
14    end
15  end
end
```

Tests pendientes

Consideramos que queda pendiente la realización de más testeos entre modelos y cómo estos se relacionan entre sí. Ejemplos de estos se detallan a continuación:

- Un usuario no debería poder responder dos veces la misma pregunta.
- Una pregunta debe tener asociada 4 opciones en total.
- Si un usuario responde una pregunta de manera incorrecta, su puntaje no debería aumentar.
- La cantidad de hojas del árbol de cada usuario se debería corresponder con la cantidad de puntos que este tiene, y que ambos sean actualizados correctamente cuando sea necesario.
- El ranking debería estar ordenado de manera correcta y ser actualizado a medida que los usuarios juegan y sus puntajes cambian.

Conclusión

Pudimos incrementar en gran medida la cantidad de tests, en comparación al cuatrimestre pasado. Observamos mediante la herramienta SimpleCov y Rspec como la cobertura iba modificándose a medida que testeamos modelos y agregaremos nuevos métodos dentro de estos.

No fue necesario modificar el código del `server.rb`, a excepción de las modificaciones realizadas en base al método `suma_points` creado en `User`, ya que anteriormente, a medida que íbamos implementando alguna funcionalidad, testeamos el proyecto jugando y probando escenarios posibles en donde se podría presentar algún conflicto. En caso de que esto pasara, modificamos y solucionamos los errores. Los nuevos tests realizados nos permitieron observar que el sistema se comporta como deseamos, pero no descartamos realizar más testeos para seguir mejorando y encontrando posibles errores.

Creemos que con los nuevos test, hemos incrementado cobertura en todos los modelos, pero más específicamente en el modelo `Usuario`. Esperamos incrementar más los testeos en los modelos de `Preguntas` y `Opciones`, para asegurarnos que la jugabilidad sea la óptima y su comportamiento el correcto.