Pruebas

1-\*/subir posts de manera correcta/\*

require 'rails\_helper'

RSpec.describe Post, type: :model do

# Prueba: Validación de un post válido

it "is valid with valid attributes" do

post = Post.new(content: "This is a valid post")

expect(post).to be\_valid

end

# Prueba: Validación de un post sin contenido

it "is not valid without content" do

post = Post.new(content: nil)

expect(post).not\_to be\_valid

expect(post.errors[:content]).to include("can't be blank")

end

# Prueba: Validación de longitud mínima del contenido

it "is not valid if content is too short" do

post = Post.new(content: "Hi")

expect(post).not\_to be\_valid

expect(post.errors[:content]).to include("is too short (minimum is 5 characters)")

end

end

2-\*/Verificar que el inicio de sesion con credenciales validas/\*

require 'rails\_helper'

RSpec.describe "User Sessions", type: :request do

let!(:user) { User.create(email: "test@example.com", password: "password123") }

describe "POST /users/sign\_in" do

context "with valid credentials" do

it "logs in the user successfully" do

post "/users/sign\_in", params: {

user: {

email: user.email,

password: "password123"

}

}

expect(response).to have\_http\_status(:ok)

expect(JSON.parse(response.body)["message"]).to eq("Signed in successfully!")

end

end

context "with invalid credentials" do

it "does not log in the user with incorrect password" do

post "/users/sign\_in", params: {

user: {

email: user.email,

password: "wrongpassword"

}

}

expect(response).to have\_http\_status(:unauthorized)

expect(JSON.parse(response.body)["message"]).to eq("Invalid Email or password.")

end

it "does not log in the user with incorrect email" do

post "/users/sign\_in", params: {

user: {

email: "wrong@example.com",

password: "password123"

}

}

expect(response).to have\_http\_status(:unauthorized)

expect(JSON.parse(response.body)["message"]).to eq("Invalid Email or password.")

end

end

end

end

3-\*/Verificar que el correo electronico exista/\*

class Users::SessionsController < Devise::SessionsController

respond\_to :json

private

def respond\_with(resource, \_opts = {})

if current\_user

render json: { message: "Signed in successfully!" }, status: :ok

else

render json: { message: "Invalid Email or password." }, status: :unauthorized

end

end

def respond\_to\_on\_destroy

render json: { message: "Signed out successfully!" }, status: :ok

end

end

4-\*/testear funcionalidad para seguir a personas/\*

require 'rails\_helper'

RSpec.describe "Follow Feature", type: :request do

let!(:user1) { User.create(email: "user1@example.com", password: "password123") }

let!(:user2) { User.create(email: "user2@example.com", password: "password123") }

describe "POST /follow" do

context "when a user follows another user" do

it "allows the user to follow successfully" do

post "/follow", params: { follower\_id: user1.id, followed\_id: user2.id }

expect(response).to have\_http\_status(:ok)

expect(JSON.parse(response.body)["message"]).to eq("Followed successfully!")

expect(user1.following).to include(user2)

end

end

context "when a user tries to follow an already followed user" do

it "does not allow duplicate follows" do

user1.following << user2

post "/follow", params: { follower\_id: user1.id, followed\_id: user2.id }

expect(response).to have\_http\_status(:unprocessable\_entity)

expect(JSON.parse(response.body)["message"]).to eq("You are already following this user.")

end

end

end

end

5-\*/Probar la funcionalidad de Retwittear\*/

require 'rails\_helper'

RSpec.describe "Retweet Feature", type: :request do

let!(:user) { User.create(email: "user@example.com", password: "password123") }

let!(:original\_post) { Post.create(user: user, content: "This is the original post") }

describe "POST /retweet" do

context "when a user retweets a post" do

it "allows the user to retweet successfully" do

post "/retweet", params: { user\_id: user.id, post\_id: original\_post.id }

expect(response).to have\_http\_status(:ok)

expect(JSON.parse(response.body)["message"]).to eq("Retweeted successfully!")

expect(original\_post.retweets.count).to eq(1)

end

end

context "when a user tries to retweet the same post multiple times" do

it "does not allow duplicate retweets" do

Post.create(user: user, original\_post: original\_post) # Simulate an existing retweet

post "/retweet", params: { user\_id: user.id, post\_id: original\_post.id }

expect(response).to have\_http\_status(:unprocessable\_entity)

expect(JSON.parse(response.body)["message"]).to eq("You have already retweeted this post.")

end

end

end

end