

Simple Unit Test for a Simple User Model:

By Brian Hogan

The User Model

The user model here is taken almost directly from the **Agile Web Development with Ruby on Rails** book's **depot** application. Most Railers are familiar with this model so it's a good place to start writing your first test.

Set up the table

First, create a **users** table. You can create a new schema.rb file under your **db** folder:

```
ActiveRecord::Schema.define() do
  create_table "users", :force => true do |t|
    t.column "username", :string, :limit => 100, :default => "", :null => false
    t.column "hashed_password", :string, :default => "", :null => false
  end
end
```

and create your table by using **rake db_schema_import**

Now set up the model

Generate the user model
ruby script/generate model User

Open the newly created `/app/models/user.rb` and paste this code:

```
require "digest/sha1"
class User < ActiveRecord::Base
  attr_accessor :password
  attr_accessible :username, :password
  validates_uniqueness_of :username
  validates_presence_of :username

  #
  def validate_on_create
    errors.add_to_base("Password field must not be left blank!") if self.password == ""
    or self.password.nil?
  end

  # hash the password for storage in the DB
  def before_create
    self.hashed_password = User.hash_password(self.password)
  end

  # hash the password before updating but only if the password field is actually filled
  in.
  def before_update()
    unless self.password.nil?
      self.password = User.hash_password(self.password)
    end
  end

  def after_create
    self.password = nil
  end
end
```

Simple Unit Test for a Simple User Model:

By Brian Hogan

```
# This exists so that you can easily create a "user" by
# simply passing the form params to this object and "try to login"
# on that object. It's just to reduce code.
def try_to_login
  User.login(self.username, self.password)
end

private
def self.hash_password(password)
  Digest::SHA1.hexdigest(password)
end

# Receives a username and password
def self.login(username, password)
  hashed_password = hash_password(password || "")
  find(:first,
    :conditions => ["username = ? and hashed_password = ?",
      username, hashed_password])
end

end
```

The Fixture

A Fixture is your test data. When the test is run, the fixture's data will automatically be loaded into your test database so you can use the data in your tests.

The file `/tests/fixtures/users.yml` should already exist. Simply paste this into that file:

```
first:
  id: 1
  username: hoganbp
  hashed_password: 7110eda4d09e062aa5e4a390b0a572ac0d2c0220
second:
  id: 2
  username: homer
  hashed_password: 8a30fca9ea5b46722cbdad24c7470a442718cac8
third:
  id: 3
  username: homer2
  hashed_password: 8a30fca9ea5b46722cbdad24c7470a442718cac8
```

A Fixture will be automatically loaded when your test starts and any data in that table **will be removed!** That's why it's really important to use a different database for testing!

The Test Database

Define your test database in your `database.yml` file. Once you have that done, you can run the command

```
rake clone_structures_to_test
```

and Rails will duplicate your table structures to your test database.

Simple Unit Test for a Simple User Model:

By Brian Hogan

The Test

The test file is already written for you; you just need to add some stuff

Open `/tests/unit/user_test.rb` and replace the contents of the file with the example below:

```
require File.dirname(__FILE__) + '/../test_helper'

class UserTest < Test::Unit::TestCase
  fixtures :users

  def test_retrieve
    assert users = User.find(:all)
    for user in users
      assert_kind_of User, user
    end
  end

  def test_create
    user = User.new()
    assert_kind_of User, user
    user.username = 'bart'
    user.password = '1234'
    assert user.save
    bart = User.find(user.id)
    assert_equal bart, user
  end

  def test_create_invalid
    user = User.new()
    assert_kind_of User, user
    user.username = 'homer'
    user.password = ''
    assert !user.save
  end

  def test_create_existing
    user = User.new()
    assert_kind_of User, user
    user.username = 'homer'
    user.password = 'simpson'
    assert !user.save
  end

  def test_failed_login
    user = User.new()
    user.username = 'homer'
    user.password = 'simpson'
    homer = user.try_to_login
    assert_nil homer
  end

  def test_successful_login
    user = User.new()
    user.username = 'homer'
    user.password = 'homer'
    homer = user.try_to_login
    assert_equal homer.username, 'homer'
  end

  def test_update
    user = User.find(3)
    user.username = "marge"
    assert user.save
  end

  def test_delete
    assert User.find(1).destroy
  end
end
```

Simple Unit Test for a Simple User Model:

By Brian Hogan

```
end
```

Run the test

Use

```
rake recent
```

to run the most recent tests.

```
Loaded suite c:/ruby/lib/ruby/gems/1.8/gems/rake-0.6.2/lib/rake/rake_test_loader
Started
.....
Finished in 0.312 seconds

8 tests, 15 assertions, 0 failures, 0 errors
```

The individual tests in this test

test_retrieve

This test retrieves all records in the table and verifies that there are records. There'd better be some results because there's data in the fixture!

The test then loops through each item in the result array and verifies that it is indeed a **User**.

test_create

This test ensures that a new user can be created. In this test, we create a new instance of the object and then verify that it is of type **User**. Then we populate it with new data and attempt to save it. A successful save causes the test to pass.

For good measure, we'll retrieve the record from the database and make sure the username equals what we originally entered.

test_create_invalid

This test will ensure that we can't create a user without providing the necessary fields. Our model requires a password (`validates_presence_of :password`) so we'll attempt to create a new user without providing a password. The key here is to assert the inverse... We expect it to fail, thus **assert !user.save** or, "user.save will return false, and we'll assert `!false`, or true when that happens."

test_create_existing

This test ensures that we can't create duplicate users. We already have Homer in our database; let's create him again! Our model requires unique usernames.

Simple Unit Test for a Simple User Model:

By Brian Hogan

Test_update

See if we can retrieve and update a record successfully. This will test two things... it will test to see if we can update just the username without being required to set the password field. Remember, the password field is not stored in the database but it is checked when a record is created.

Test_delete

See if we can delete a record from the database

test_failed_login and test_successful_login

I really hope these are self-explanatory by now.

Stuff you'll want to know!

Tests don't necessarily run in the order you'd expect. Don't do a delete on a record that's being tested in another test! Don't check a value for a record that might have been updated in another test method!

Keep up on your stats too! **Rake stats** can tell you much about your application, including your code-to-test ratio!