E:\workspace\payroll\_rails\test\fixtures\attendance.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

staff\_id: C0001

work\_date: 2013-01-12

time\_in: 2013-01-12 23:18:51

time\_out: 2013-01-12 23:18:51

two:

id: 2

staff\_id: C0002

work\_date: 2013-01-12

time\_in: 2013-01-12 23:18:51

time\_out: 2013-01-12 23:18:51

E:\workspace\payroll\_rails\test\fixtures\department.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

name: R & D

two:

id: 2

name: Marketing

E:\workspace\payroll\_rails\test\fixtures\designation.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

title: Programmer

desc: Software development

note: Programming job

two:

id: 2

title: Admin

desc: Admin work

note: Admin job

E:\workspace\payroll\_rails\test\fixtures\employee.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

staff\_id: C0001

first\_name: ben

middle\_name: ong

last\_name: mun

new\_ic: 554498824

old\_ic:

passport\_no: 12889100025

gender: M

marital\_status: S

nationality: Malaysian

dob: 2012-12-17

place\_of\_birth: Serdang

race: Chinese

religion: Christian

is\_bumi: false

user\_id: 2

two:

id: 2

staff\_id: C0002

first\_name: ken

middle\_name: looi

last\_name: kin

new\_ic: 7654399054

old\_ic:

passport\_no: 560083200

gender: F

marital\_status: O

nationality: Malaysian

dob: 2012-12-17

place\_of\_birth: Puchong

race: Chinese

religion: Buddhist

is\_bumi: false

user\_id: 3

E:\workspace\payroll\_rails\test\fixtures\employee\_contact.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

address\_1: 8

address\_2: Jalan Puteri

address\_3: Taman Puteri

city: Kuala Lumpur

state: W.P

postcode: 54000

country: Malaysia

home\_phone: 038766056

mobile\_phone: 0123398765

work\_email: ken@gmail.com

other\_email: ken@yahoo.com

two:

id: 2

address\_1: 9

address\_2: Jalan Maju

address\_3: Taman Bahagia

city: Kuala Lumpur

state: W.P

postcode: 56000

country: Malaysia

home\_phone: 038876599

mobile\_phone: 0128744522

work\_email: ben@gmail.com

other\_email: ben@yahoo.net

E:\workspace\payroll\_rails\test\fixtures\employee\_job.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

designation\_id: 1

department\_id: 1

employment\_status\_id: 1

job\_category\_id: 1

join\_date: 2012-12-17

confirm\_date: 2012-12-17

two:

id: 2

designation\_id: 2

department\_id: 2

employment\_status\_id: 1

job\_category\_id: 2

join\_date: 2012-12-17

confirm\_date: 2012-12-17

E:\workspace\payroll\_rails\test\fixtures\employee\_qualification.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

level: 4

institute: APIT

major: IT

year: 2009

gpa: 3.4

start\_date: 2004-05-06

end\_date: 2006-05-09

two:

id: 2

level: 4

institute: Sedaya

major: Engineering

year: 2008

gpa: 3.7

start\_date: 2003-03-01

end\_date: 2005-05-10

E:\workspace\payroll\_rails\test\fixtures\employee\_salary.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

salary: 4200

allowance: 20

epf: 350

socso: 10

bank\_name: Maybank

bank\_acc\_no: 0956644

bank\_acc\_type: Savings

bank\_address: Serdang

epf\_no: 776499

socso\_no: 9954788

income\_tax\_no: 0096734567

pay\_type: 1

two:

id: 2

salary: 3400

allowance: 50

epf: 300

socso: 30

bank\_name: Public

bank\_acc\_no: 88650066

bank\_acc\_type: Savings

bank\_address: Puchong

epf\_no: 560099

socso\_no: 5930033

income\_tax\_no: 0958467443

pay\_type: 2

E:\workspace\payroll\_rails\test\fixtures\employment\_status.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

name: Probation

two:

id: 2

name: Terminated

E:\workspace\payroll\_rails\test\fixtures\job\_category.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

name: Marketing

two:

id: 2

name: Sales

E:\workspace\payroll\_rails\test\fixtures\overtime\_rate.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

duration: 1.5

year: 2010

pay\_rate: 1.5

two:

duration: 1.5

year: 2008

pay\_rate: 1.5

E:\workspace\payroll\_rails\test\fixtures\pay\_rate.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

staff\_id: C0001

month: 1

year: 2011

hourly\_pay\_rate: 1.5

two:

id: 2

staff\_id: C0002

month: 2

year: 2011

hourly\_pay\_rate: 1.5

E:\workspace\payroll\_rails\test\fixtures\salary\_adjustment.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

one:

id: 1

staff\_id: C0001

inc: 1.5

month: 1

year: 2013

two:

id: 2

staff\_id: C0002

inc: 1.5

month: 1

year: 2013

E:\workspace\payroll\_rails\test\fixtures\user.yml

# Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html

admin:

id: 1

role: 1

username: admin

status: true

password: e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4 # secret

ben:

id: 2

role: 2

username: ben

status: true

password: e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4 # secret

ken:

id: 3

role: 2

username: ken

status: true

password: e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4 # secret

E:\workspace\payroll\_rails\test\test\_helper.rb

ENV["RAILS\_ENV"] = "test"

require File.expand\_path('../../config/environment', \_\_FILE\_\_)

require 'rails/test\_help'

class ActiveSupport::TestCase

# Setup all fixtures in test/fixtures/\*.(yml|csv) for all tests in alphabetical order.

#

# Note: You'll currently still have to declare fixtures explicitly in integration tests

# -- they do not yet inherit this setting

self.use\_transactional\_fixtures = true

set\_fixture\_class :employment\_status => 'EmploymentStatus'

fixtures :all

# Add more helper methods to be used by all tests here...

# Add more helper methods to be used by all tests here...

def login\_as(user)

ouser = user(user)

@request.session[:user\_id] = ouser.id

if ouser.role != User::ADMIN

employee = ouser.employee

@request.session[:employee\_id] = employee.id

@request.session[:staff\_id] = employee.staff\_id

@request.session[:supervisor\_id] = employee.id

end

end

def normal\_user(test\_module)

open\_session do |user|

def user.logs\_in(username, password)

get login\_path

assert\_response :success

assert\_template 'new'

post\_via\_redirect auth\_path, :username => username, :password => password

assert\_response :success

assert\_equal user\_index\_path, path

assert\_template 'index'

assert\_not\_nil session[:user\_id]

assert\_not\_nil session[:employee\_id]

assert\_not\_nil session[:staff\_id]

assert\_not\_nil session[:supervisor\_id]

end

def user.logs\_out

get\_via\_redirect logout\_path

assert\_response :success

assert\_equal login\_path, path

assert\_template 'new'

assert\_nil session[:user\_id]

assert\_nil session[:employee\_id]

assert\_nil session[:staff\_id]

assert\_nil session[:supervisor\_id]

end

user.extend(test\_module)

yield user if block\_given?

end

end

def admin\_user(test\_module)

open\_session do |user|

def user.logs\_in(username, password)

get login\_path

assert\_response :success

assert\_template 'new'

post\_via\_redirect auth\_path, :username => username, :password => password

assert\_response :success

assert\_equal admin\_index\_path, path

assert\_template 'index'

assert\_not\_nil session[:user\_id]

end

def user.logs\_out

get\_via\_redirect logout\_path

assert\_response :success

assert\_equal login\_path, path

assert\_template 'new'

assert\_nil session[:user\_id]

end

user.extend(test\_module)

yield user if block\_given?

end

end

end

E:\workspace\payroll\_rails\test\unit\attendance\_test.rb

require 'test\_helper'

class AttendanceTest < ActiveSupport::TestCase

test 'should create attendance' do

o = Attendance.new

o.id = 3

o.staff\_id = 'S0007'

o.time\_in = Time.new(2009, 4, 3, 8, 34, 0, '+08:00')

o.time\_out = Time.new(2009, 4, 3, 18, 34, 0, '+08:00')

o.work\_date = Date.new(2009, 4, 3)

assert o.save

end

test 'should find attendance' do

id = attendance(:one).id

assert\_nothing\_raised { Attendance.find(id) }

end

test 'should update attendance' do

o = attendance(:two)

assert o.update\_attributes(:staff\_id => 'S0008')

o.work\_date = Date.new(2010, 6, 8)

assert o.save

o = Attendance.find(o.id)

assert\_equal '2010-06-08', o.work\_date.strftime('%Y-%m-%d')

end

test 'should destroy attendance' do

o = attendance(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { Attendance.find(o.id) }

end

test 'should not create a attendance without required fields' do

o = Attendance.new

assert !o.valid?

assert o.errors[:work\_date].any?

assert o.errors[:time\_in].any?

assert o.errors[:time\_out].any?

assert\_equal ['Work date is required'], o.errors[:work\_date]

assert\_equal ['Time in is required'], o.errors[:time\_in]

assert\_equal ['Time out is required'], o.errors[:time\_out]

end

end

E:\workspace\payroll\_rails\test\unit\department\_test.rb

require 'test\_helper'

class DepartmentTest < ActiveSupport::TestCase

test 'should create department' do

o = Department.new

o.name = 'Account'

assert o.save

end

test 'should find department' do

id = department(:one).id

assert\_nothing\_raised { Department.find(id) }

end

test 'should update department' do

o = department(:two)

assert o.update\_attributes(:name => 'QC')

o.name = 'Support'

assert o.save

o = Department.find(o.id)

assert\_equal 'Support', o.name

end

test 'should destroy department' do

o = department(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { Department.find(o.id) }

end

test 'should not create a department without required fields' do

o = Department.new

assert !o.valid?

assert o.errors[:name].any?

assert\_equal ['Name is required'], o.errors[:name]

end

test 'should not create a department with duplicate name' do

o = Department.new

o.name = 'Marketing'

assert !o.valid?

assert o.errors[:name].any?

assert\_equal ['Department Marketing already exist'], o.errors[:name]

end

end

E:\workspace\payroll\_rails\test\unit\designation\_test.rb

require 'test\_helper'

class DesignationTest < ActiveSupport::TestCase

test 'should create designation' do

o = Designation.new

o.title = 'Manager'

assert o.save

end

test 'should find designation' do

id = designation(:one).id

assert\_nothing\_raised { Designation.find(id) }

end

test 'should update designation' do

o = designation(:two)

assert o.update\_attributes(:title => 'Admin')

o.title = 'Account'

assert o.save

o = Designation.find(o.id)

assert\_equal 'Account', o.title

end

test 'should destroy designation' do

o = designation(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { Designation.find(o.id) }

end

test 'should not create a designation without required fields' do

o = Designation.new

assert !o.valid?

assert o.errors[:title].any?

assert\_equal ['Job Title is required'], o.errors[:title]

end

test 'should not create a designation with duplicate title' do

o = Designation.new

o.title = 'Admin'

assert !o.valid?

assert o.errors[:title].any?

assert\_equal ['Job Title Admin already exist'], o.errors[:title]

end

end

E:\workspace\payroll\_rails\test\unit\employee\_contact\_test.rb

require 'test\_helper'

class EmployeeContactTest < ActiveSupport::TestCase

test 'should create employee\_contact' do

o = EmployeeContact.new

o.id = 3

o.address\_1 = 'Jalan Maju'

o.address\_2 = 'Blok 6'

o.address\_3 = 'Taman Gembira'

o.city = 'KL'

o.state = 'WP'

o.postcode = '56000'

o.country = 'Malaysia'

o.home\_phone = '666'

o.mobile\_phone = '777'

o.work\_email = 'test@gmail.com'

o.other\_email = 'home@gmail.com'

assert o.save

end

test 'should find employee\_contact' do

id = employee\_contact(:one).id

assert\_nothing\_raised { EmployeeContact.find(id) }

end

test 'should update employee\_contact' do

o = employee\_contact(:two)

assert o.update\_attributes(:address\_1 => 'Jalan Rasa', :address\_2 => 'Taman Kerinchi',

:address\_3 => 'Taman Pinang')

o.state = 'Selangor'

assert o.save

o = EmployeeContact.find(o.id)

assert\_equal 'Selangor', o.state

assert\_equal 'Taman Pinang', o.address\_3

end

test 'should destroy employee\_contact' do

o = employee\_contact(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { EmployeeContact.find(o.id) }

end

test 'should not create a employee\_contact without required fields' do

o = EmployeeContact.new

assert !o.valid?

assert o.errors[:address\_1].any?

assert o.errors[:city].any?

assert o.errors[:state].any?

assert o.errors[:postcode].any?

assert o.errors[:country].any?

assert o.errors[:work\_email].any?

assert\_equal ['Address 1 is required'], o.errors[:address\_1]

assert\_equal ['City is required'], o.errors[:city]

assert\_equal ['State is required'], o.errors[:state]

assert\_equal ['Postal Code is required'], o.errors[:postcode]

assert\_equal ['Country is required'], o.errors[:country]

assert\_equal ['Work Email is required'], o.errors[:work\_email]

end

end

E:\workspace\payroll\_rails\test\unit\employee\_job\_test.rb

require 'test\_helper'

class EmployeeJobTest < ActiveSupport::TestCase

test 'should create employee\_job' do

o = EmployeeJob.new

o.id = 8

o.confirm\_date = Date.new(2010, 7, 1)

o.department\_id = 1

o.designation\_id = 2

o.employment\_status\_id = 3

o.job\_category\_id = 4

o.join\_date = Date.new(2010, 6, 1)

assert o.save

end

test 'should find employee\_job' do

id = employee\_job(:one).id

assert\_nothing\_raised { EmployeeJob.find(id) }

end

test 'should update employee\_job' do

o = employee\_job(:two)

assert o.update\_attributes(:confirm\_date => Date.new(2009, 5, 4), :department\_id => 5,

:designation\_id => 4, :employment\_status\_id => 6,

:job\_category\_id => 4, :join\_date => Date.new(2009, 2, 6))

o.join\_date = Date.new(2009, 2, 3)

assert o.save

o = EmployeeJob.find(o.id)

assert\_equal '2009-02-03', o.join\_date.strftime('%Y-%m-%d')

end

test 'should destroy employee\_job' do

o = employee\_job(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { EmployeeJob.find(o.id) }

end

test 'should not create a employee\_job without required fields' do

o = EmployeeJob.new

assert !o.valid?

assert o.errors[:designation\_id].any?

assert o.errors[:department\_id].any?

assert o.errors[:employment\_status\_id].any?

assert o.errors[:job\_category\_id].any?

assert o.errors[:join\_date].any?

assert\_equal ['Designation is required'], o.errors[:designation\_id]

assert\_equal ['Department is required'], o.errors[:department\_id]

assert\_equal ['Employment Status is required'], o.errors[:employment\_status\_id]

assert\_equal ['Job Category is required'], o.errors[:job\_category\_id]

assert\_equal ['Join Date is required'], o.errors[:join\_date]

end

end

E:\workspace\payroll\_rails\test\unit\employee\_qualification\_test.rb

require 'test\_helper'

class EmployeeQualificationTest < ActiveSupport::TestCase

test 'should create employee\_qualification' do

o = EmployeeQualification.new

o.id = 3

o.level = 4

o.institute = 'Informatics'

o.major = 'Computer Science'

o.year = 2005

o.gpa = 3.0

o.start\_date = Date.new(2005, 8, 6)

o.end\_date = Date.new(2007, 3, 7)

assert o.save

end

test 'should find employee\_qualification' do

id = employee\_qualification(:one).id

assert\_nothing\_raised { EmployeeQualification.find(id) }

end

test 'should update employee\_qualification' do

o = employee\_qualification(:two)

assert o.update\_attributes(:level => 5)

o.institute = 'UPM'

assert o.save

o = EmployeeQualification.find(o.id)

assert\_equal 'UPM', o.institute

end

test 'should destroy employee\_qualification' do

o = employee\_qualification(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { EmployeeQualification.find(o.id) }

end

test 'should not create a department without required fields' do

o = EmployeeQualification.new

assert !o.valid?

assert o.errors[:level].any?

assert o.errors[:institute].any?

assert o.errors[:year].any?

assert o.errors[:start\_date].any?

assert o.errors[:end\_date].any?

assert\_equal ['Qualification Level is required'], o.errors[:level]

assert\_equal ['Institute name is required'], o.errors[:institute]

assert\_equal ['Year obtained is required'], o.errors[:year]

assert\_equal ['Start Date is required'], o.errors[:start\_date]

assert\_equal ['End Date is required'], o.errors[:end\_date]

end

end

E:\workspace\payroll\_rails\test\unit\employee\_salary\_test.rb

require 'test\_helper'

class EmployeeSalaryTest < ActiveSupport::TestCase

test 'should create employee\_salary' do

o = EmployeeSalary.new

o.id = 3

o.salary = 5000

o.allowance = 40

o.epf = 67

o.socso = 88

o.income\_tax = 89

o.bank\_name = 'cimb'

o.bank\_acc\_no = '67886554'

o.bank\_acc\_type = 'savings'

o.bank\_address = 'klcc'

o.epf\_no = '4478966'

o.socso\_no = '5590876'

o.income\_tax\_no = '667546'

o.pay\_type = 1

assert o.save

end

test 'should find employee\_salary' do

id = employee\_salary(:one).id

assert\_nothing\_raised { EmployeeSalary.find(id) }

end

test 'should update employee\_salary' do

o = employee\_salary(:two)

assert o.update\_attributes(:salary => 5500, :allowance => 56)

o.bank\_name = 'rhb'

assert o.save

o = EmployeeSalary.find(o.id)

assert\_equal 'rhb', o.bank\_name

end

test 'should destroy employee\_salary' do

o = employee\_salary(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { EmployeeSalary.find(o.id) }

end

test 'should not create a employee\_salary without required fields' do

o = EmployeeSalary.new

o.salary = -90

o.epf = -88

assert !o.valid?

assert o.errors[:salary].any?

assert o.errors[:bank\_name].any?

assert o.errors[:bank\_acc\_no].any?

assert o.errors[:bank\_acc\_type].any?

assert o.errors[:bank\_address].any?

assert o.errors[:epf\_no].any?

assert o.errors[:pay\_type].any?

assert o.errors[:epf].any?

assert\_equal ['Salary is invalid'], o.errors[:salary]

assert\_equal ['Bank Name is required'], o.errors[:bank\_name]

assert\_equal ['Bank Account No. is required'], o.errors[:bank\_acc\_no]

assert\_equal ['Bank Account Type is required'], o.errors[:bank\_acc\_type]

assert\_equal ['Bank Address is required'], o.errors[:bank\_address]

assert\_equal ['EPF No. is required'], o.errors[:epf\_no]

assert\_equal ['Pay Type is required'], o.errors[:pay\_type]

assert\_equal ['EPF Deduction is invalid'], o.errors[:epf]

end

end

E:\workspace\payroll\_rails\test\unit\employee\_test.rb

require 'test\_helper'

class EmployeeTest < ActiveSupport::TestCase

test 'should create employee' do

o = Employee.new

o.id = '0007'

o.staff\_id = 'S0007'

o.first\_name = 'Ken'

o.middle\_name = 'Chua'

o.last\_name = 'Leong'

o.new\_ic = '887760489'

o.gender = 'M'

o.marital\_status = 'S'

o.nationality = 'Malaysian'

o.dob = Date.new(1987, 9, 6)

o.place\_of\_birth = 'PJ'

o.race = 'Chinese'

o.religion = 'Buddhist'

o.is\_bumi = false

o.user\_id = 2

assert o.save

end

test 'should find employee' do

id = employee(:one).id

assert\_nothing\_raised { Employee.find(id) }

end

test 'should update employee' do

o = employee(:two)

assert o.update\_attributes(:new\_ic => '776593367', :last\_name => 'Paul')

o.first\_name = 'Jerry'

assert o.save

o = Employee.find(o.id)

assert\_equal 'Jerry', o.first\_name

end

test 'should destroy employee' do

o = employee(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { Employee.find(o.id) }

end

test 'should not create a employee without required fields' do

o = Employee.new

assert !o.valid?

assert o.errors[:staff\_id].any?

assert o.errors[:first\_name].any?

assert o.errors[:last\_name].any?

assert o.errors[:new\_ic].any?

assert o.errors[:gender].any?

assert o.errors[:marital\_status].any?

assert o.errors[:nationality].any?

assert o.errors[:dob].any?

assert o.errors[:place\_of\_birth].any?

assert o.errors[:race].any?

assert\_equal ['Employee ID is required'], o.errors[:staff\_id]

assert\_equal ['First Name is required'], o.errors[:first\_name]

assert\_equal ['Last Name is required'], o.errors[:last\_name]

assert\_equal ['New IC No. is required'], o.errors[:new\_ic]

assert\_equal ['Gender is required'], o.errors[:gender]

assert\_equal ['Marital Status is required'], o.errors[:marital\_status]

assert\_equal ['Nationality is required'], o.errors[:nationality]

assert\_equal ['Date of Birth is required'], o.errors[:dob]

assert\_equal ['Place of Birth is required'], o.errors[:place\_of\_birth]

assert\_equal ['Race is required'], o.errors[:race]

end

test 'should not create a employee with duplicate staff\_id' do

o = Employee.new

o.staff\_id = 'C0002'

assert !o.valid?

assert o.errors[:staff\_id].any?

assert\_equal ['Employee ID C0002 already exist'], o.errors[:staff\_id]

end

end

E:\workspace\payroll\_rails\test\unit\employment\_status\_test.rb

require 'test\_helper'

class EmploymentStatusTest < ActiveSupport::TestCase

test 'should create employment\_status' do

o = EmploymentStatus.new

o.name = 'Confirmed'

assert o.save

end

test 'should find employment\_status' do

id = employment\_status(:one).id

assert\_nothing\_raised { EmploymentStatus.find(id) }

end

test 'should update employment\_status' do

o = employment\_status(:two)

assert o.update\_attributes(:name => 'Temporary')

o.name = 'Contract'

assert o.save

o = EmploymentStatus.find(o.id)

assert\_equal 'Contract', o.name

end

test 'should destroy employment\_status' do

o = employment\_status(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { EmploymentStatus.find(o.id) }

end

test 'should not create a employment\_status without required fields' do

o = EmploymentStatus.new

assert !o.valid?

assert o.errors[:name].any?

assert\_equal ['Name is required'], o.errors[:name]

end

test 'should not create a employment\_status with duplicate name' do

o = EmploymentStatus.new

o.name = 'Probation'

assert !o.valid?

assert o.errors[:name].any?

assert\_equal ['Employment Status Probation already exist'], o.errors[:name]

end

end

E:\workspace\payroll\_rails\test\unit\job\_category\_test.rb

require 'test\_helper'

class JobCategoryTest < ActiveSupport::TestCase

test 'should create job\_category' do

o = JobCategory.new

o.name = 'Technical'

assert o.save

end

test 'should find job\_category' do

id = job\_category(:one).id

assert\_nothing\_raised { JobCategory.find(id) }

end

test 'should update job\_category' do

o = job\_category(:two)

assert o.update\_attributes(:name => 'QC')

o.name = 'Support'

assert o.save

o = JobCategory.find(o.id)

assert\_equal 'Support', o.name

end

test 'should destroy job\_category' do

o = job\_category(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { JobCategory.find(o.id) }

end

test 'should not create a job\_category without required fields' do

o = JobCategory.new

assert !o.valid?

assert o.errors[:name].any?

assert\_equal ['Name is required'], o.errors[:name]

end

test 'should not create a job\_category with duplicate name' do

o = JobCategory.new

o.name = 'Marketing'

assert !o.valid?

assert o.errors[:name].any?

assert\_equal ['Category Marketing already exist'], o.errors[:name]

end

end

E:\workspace\payroll\_rails\test\unit\overtime\_rate\_test.rb

require 'test\_helper'

class OvertimeRateTest < ActiveSupport::TestCase

test 'should create overtime\_rate' do

o = OvertimeRate.new

o.duration = 2

o.year = 2009

o.pay\_rate = 89

assert o.save

end

test 'should find overtime\_rate' do

id = overtime\_rate(:one).id

assert\_nothing\_raised { OvertimeRate.find(id) }

end

test 'should update overtime\_rate' do

o = overtime\_rate(:two)

assert o.update\_attributes(:duration => 3, :year => 2007, :pay\_rate => 86)

o.duration = 1

assert o.save

o = OvertimeRate.find(o.id)

assert\_equal 1, o.duration

end

test 'should destroy overtime\_rate' do

o = overtime\_rate(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { OvertimeRate.find(o.id) }

end

test 'should not create a overtime\_rate without required fields' do

o = OvertimeRate.new

o.duration = -8

o.pay\_rate = 0

assert !o.valid?

assert o.errors[:duration].any?

assert o.errors[:year].any?

assert o.errors[:pay\_rate].any?

assert\_equal ['Duration is invalid'], o.errors[:duration]

assert\_equal ['Year is required', 'Year is invalid'], o.errors[:year]

assert\_equal ['Pay Rate is invalid'], o.errors[:pay\_rate]

end

test 'should not create a overtime\_rate with duplicate year' do

o = OvertimeRate.new

o.year = 2008

assert !o.valid?

assert o.errors[:year].any?

assert\_equal ['Overtime rate for year 2008 already exist'], o.errors[:year]

end

end

E:\workspace\payroll\_rails\test\unit\pay\_rate\_test.rb

require 'test\_helper'

class PayRateTest < ActiveSupport::TestCase

test 'should create pay\_rate' do

o = PayRate.new

o.id = 3

o.staff\_id = 'S0009'

o.month = 8

o.year = 2009

o.hourly\_pay\_rate = 67

assert o.save

end

test 'should find pay\_rate' do

id = pay\_rate(:one).id

assert\_nothing\_raised { PayRate.find(id) }

end

test 'should update pay\_rate' do

o = pay\_rate(:two)

assert o.update\_attributes(:month => 7, :year => 2006, :hourly\_pay\_rate => 44)

o.month = 6

assert o.save

o = PayRate.find(o.id)

assert\_equal 6, o.month

end

test 'should destroy pay\_rate' do

o = pay\_rate(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { PayRate.find(o.id) }

end

test 'should not create a pay\_rate without required fields' do

o = PayRate.new

assert !o.valid?

assert o.errors[:staff\_id].any?

assert o.errors[:month].any?

assert o.errors[:year].any?

assert\_equal ['Staff ID is required'], o.errors[:staff\_id]

assert\_equal ['Month is required', 'Month is invalid', 'Month is invalid'], o.errors[:month]

assert\_equal ['Year is required', 'Year is invalid'], o.errors[:year]

o.month = 0

o.year = 0

o.hourly\_pay\_rate = 0

assert !o.valid?

assert\_equal ['Month is invalid'], o.errors[:month]

assert\_equal ['Year is invalid'], o.errors[:year]

assert\_equal ['Hourly pay rate is invalid'], o.errors[:hourly\_pay\_rate]

end

end

E:\workspace\payroll\_rails\test\unit\route\_test.rb

require 'test\_helper'

class RouteTest < ActionController::TestCase

test 'should route to auth' do

assert\_routing '/login', { :controller => 'application', :action => 'new' }

assert\_routing '/auth', { :controller => 'application', :action => 'create' }

assert\_routing '/logout', { :controller => 'application', :action => 'destroy' }

end

test 'should route to admin' do

assert\_routing '/', { :controller => 'admin/admin', :action => 'index' }

assert\_routing '/admin/user', { :controller => 'admin/user', :action => 'index' }

assert\_routing '/admin/user/list', { :controller => 'admin/user', :action => 'list' }

assert\_routing '/admin/user/new', { :controller => 'admin/user', :action => 'new' }

assert\_routing({ :method => :post, :path => '/admin/user/create' },

{ :controller => 'admin/user', :action => 'create' })

assert\_routing '/admin/user/edit/1', { :controller => 'admin/user', :action => 'edit', :id => '1' }

assert\_routing({ :method => :post, :path => '/admin/user/update' },

{ :controller => 'admin/user', :action => 'update' })

assert\_routing({ :method => :post, :path => '/admin/user/delete'},

{ :controller => 'admin/user', :action => 'destroy' })

assert\_routing '/admin/employee', { :controller => 'admin/employee', :action => 'index' }

assert\_routing '/admin/employee/list', { :controller => 'admin/employee', :action => 'list' }

assert\_routing '/admin/employee/new', { :controller => 'admin/employee', :action => 'new' }

assert\_routing({ :method => :post, :path => '/admin/employee/create' },

{ :controller => 'admin/employee', :action => 'create' })

assert\_routing '/admin/employee/edit/1', { :controller => 'admin/employee', :action => 'edit', :id => '1' }

assert\_routing({ :method => :post, :path => '/admin/employee/update' },

{ :controller => 'admin/employee', :action => 'update' })

assert\_routing({ :method => :post, :path => '/admin/employee/delete'},

{ :controller => 'admin/employee', :action => 'destroy' })

assert\_routing '/admin/designation', { :controller => 'admin/designation', :action => 'index' }

assert\_routing '/admin/designation/list', { :controller => 'admin/designation', :action => 'list' }

assert\_routing '/admin/designation/new', { :controller => 'admin/designation', :action => 'new' }

assert\_routing({ :method => :post, :path => '/admin/designation/create' },

{ :controller => 'admin/designation', :action => 'create' })

assert\_routing '/admin/designation/edit/1', { :controller => 'admin/designation', :action => 'edit', :id => '1' }

assert\_routing({ :method => :post, :path => '/admin/designation/update' },

{ :controller => 'admin/designation', :action => 'update' })

assert\_routing({ :method => :post, :path => '/admin/designation/delete'},

{ :controller => 'admin/designation', :action => 'destroy' })

assert\_routing '/admin/empstatus', { :controller => 'admin/employment\_status', :action => 'index' }

assert\_routing '/admin/empstatus/list', { :controller => 'admin/employment\_status', :action => 'list' }

assert\_routing '/admin/empstatus/new', { :controller => 'admin/employment\_status', :action => 'new' }

assert\_routing({ :method => :post, :path => '/admin/empstatus/create' },

{ :controller => 'admin/employment\_status', :action => 'create' })

assert\_routing '/admin/empstatus/edit/1', { :controller => 'admin/employment\_status', :action => 'edit', :id => '1' }

assert\_routing({ :method => :post, :path => '/admin/empstatus/update' },

{ :controller => 'admin/employment\_status', :action => 'update' })

assert\_routing({ :method => :post, :path => '/admin/empstatus/delete'},

{ :controller => 'admin/employment\_status', :action => 'destroy' })

assert\_routing '/admin/jobcat', { :controller => 'admin/job\_category', :action => 'index' }

assert\_routing '/admin/jobcat/list', { :controller => 'admin/job\_category', :action => 'list' }

assert\_routing '/admin/jobcat/new', { :controller => 'admin/job\_category', :action => 'new' }

assert\_routing({ :method => :post, :path => '/admin/jobcat/create' },

{ :controller => 'admin/job\_category', :action => 'create' })

assert\_routing '/admin/jobcat/edit/1', { :controller => 'admin/job\_category', :action => 'edit', :id => '1' }

assert\_routing({ :method => :post, :path => '/admin/jobcat/update' },

{ :controller => 'admin/job\_category', :action => 'update' })

assert\_routing({ :method => :post, :path => '/admin/jobcat/delete'},

{ :controller => 'admin/job\_category', :action => 'destroy' })

assert\_routing '/admin/dept', { :controller => 'admin/department', :action => 'index' }

assert\_routing '/admin/dept/list', { :controller => 'admin/department', :action => 'list' }

assert\_routing '/admin/dept/new', { :controller => 'admin/department', :action => 'new' }

assert\_routing({ :method => :post, :path => '/admin/dept/create' },

{ :controller => 'admin/department', :action => 'create' })

assert\_routing '/admin/dept/edit/1', { :controller => 'admin/department', :action => 'edit', :id => '1' }

assert\_routing({ :method => :post, :path => '/admin/dept/update' },

{ :controller => 'admin/department', :action => 'update' })

assert\_routing({ :method => :post, :path => '/admin/dept/delete'},

{ :controller => 'admin/department', :action => 'destroy' })

assert\_routing '/admin/payrate', { :controller => 'admin/pay\_rate', :action => 'index' }

assert\_routing '/admin/payrate/list', { :controller => 'admin/pay\_rate', :action => 'list' }

assert\_routing '/admin/payrate/new', { :controller => 'admin/pay\_rate', :action => 'new' }

assert\_routing({ :method => :post, :path => '/admin/payrate/create' },

{ :controller => 'admin/pay\_rate', :action => 'create' })

assert\_routing '/admin/payrate/edit/1', { :controller => 'admin/pay\_rate', :action => 'edit', :id => '1' }

assert\_routing({ :method => :post, :path => '/admin/payrate/update' },

{ :controller => 'admin/pay\_rate', :action => 'update' })

assert\_routing({ :method => :post, :path => '/admin/payrate/delete'},

{ :controller => 'admin/pay\_rate', :action => 'destroy' })

assert\_routing '/admin/hourly/chart', { :controller => 'admin/hourly\_payroll\_chart', :action => 'index' }

assert\_routing '/admin/hourly/chart/data', { :controller => 'admin/hourly\_payroll\_chart', :action => 'data' }

assert\_routing 'admin/payslip', { :controller => 'admin/payslip', :action => 'index' }

assert\_routing 'admin/payslip/list', { :controller => 'admin/payslip', :action => 'list' }

assert\_routing 'admin/payslip/slip/1/2/2009', { :controller => 'admin/payslip', :action => 'payslip',

:id => '1', :month => '2', :year => '2009' }

assert\_routing 'admin/att', { :controller => 'admin/attendance', :action => 'index' }

assert\_routing 'admin/att/list', { :controller => 'admin/attendance', :action => 'list' }

assert\_routing '/admin/overtime/rate', { :controller => 'admin/overtime\_rate', :action => 'index' }

assert\_routing '/admin/overtime/rate/list', { :controller => 'admin/overtime\_rate', :action => 'list' }

assert\_routing '/admin/overtime/rate/new', { :controller => 'admin/overtime\_rate', :action => 'new' }

assert\_routing({ :method => :post, :path => '/admin/overtime/rate/create' },

{ :controller => 'admin/overtime\_rate', :action => 'create' })

assert\_routing '/admin/overtime/rate/edit/1', { :controller => 'admin/overtime\_rate', :action => 'edit', :id => '1' }

assert\_routing({ :method => :post, :path => '/admin/overtime/rate/update' },

{ :controller => 'admin/overtime\_rate', :action => 'update' })

assert\_routing({ :method => :post, :path => '/admin/overtime/rate/delete'},

{ :controller => 'admin/overtime\_rate', :action => 'destroy' })

assert\_routing '/admin/overtime/chart', { :controller => 'admin/overtime\_chart', :action => 'index' }

assert\_routing '/admin/overtime/chart/data', { :controller => 'admin/overtime\_chart', :action => 'data' }

assert\_routing '/admin/workhours/chart', { :controller => 'admin/total\_work\_hours\_chart', :action => 'index' }

assert\_routing '/admin/workhours/chart/data', { :controller => 'admin/total\_work\_hours\_chart', :action => 'data' }

assert\_routing '/admin/salaryadj', { :controller => 'admin/salary\_adjustment', :action => 'index' }

assert\_routing '/admin/salaryadj/list', { :controller => 'admin/salary\_adjustment', :action => 'list' }

assert\_routing '/admin/salaryadj/new', { :controller => 'admin/salary\_adjustment', :action => 'new' }

assert\_routing({ :method => :post, :path => '/admin/salaryadj/create' },

{ :controller => 'admin/salary\_adjustment', :action => 'create' })

assert\_routing '/admin/salaryadj/edit/1', { :controller => 'admin/salary\_adjustment', :action => 'edit', :id => '1' }

assert\_routing({ :method => :post, :path => '/admin/salaryadj/update' },

{ :controller => 'admin/salary\_adjustment', :action => 'update' })

assert\_routing({ :method => :post, :path => '/admin/salaryadj/delete'},

{ :controller => 'admin/salary\_adjustment', :action => 'destroy' })

end

test 'should route to user' do

assert\_routing '/user/index', { :controller => 'user/user', :action => 'index' }

assert\_routing '/user/info', { :controller => 'user/info', :action => 'index' }

assert\_routing({ :method => :post, :path => '/user/info/update' },

{ :controller => 'user/info', :action => 'update' })

assert\_routing '/user/contact', { :controller => 'user/contact', :action => 'index' }

assert\_routing({ :method => :post, :path => '/user/contact/update' },

{ :controller => 'user/contact', :action => 'update' })

assert\_routing '/user/job', { :controller => 'user/job', :action => 'index' }

assert\_routing '/user/salary', { :controller => 'user/salary', :action => 'index' }

assert\_routing '/user/qualification', { :controller => 'user/qualification', :action => 'index' }

assert\_routing({ :method => :post, :path => '/user/qualification/update' },

{ :controller => 'user/qualification', :action => 'update' })

assert\_routing '/user/overtime/chart', { :controller => 'user/overtime\_chart', :action => 'index' }

assert\_routing '/user/overtime/chart/data', { :controller => 'user/overtime\_chart', :action => 'data' }

assert\_routing '/user/workhours/chart', { :controller => 'user/total\_work\_hours\_chart', :action => 'index' }

assert\_routing '/user/workhours/chart/data', { :controller => 'user/total\_work\_hours\_chart', :action => 'data' }

assert\_routing '/user/hourly/chart', { :controller => 'user/hourly\_payroll\_chart', :action => 'index' }

assert\_routing '/user/hourly/chart/data', { :controller => 'user/hourly\_payroll\_chart', :action => 'data' }

assert\_routing '/user/payslip', { :controller => 'user/payslip', :action => 'index' }

assert\_routing '/user/payslip/slip/1/2010', { :controller => 'user/payslip', :action => 'payslip',

:month => '1', :year => '2010' }

end

end

E:\workspace\payroll\_rails\test\unit\salary\_adjustment\_test.rb

require 'test\_helper'

class SalaryAdjustmentTest < ActiveSupport::TestCase

test 'should create salary\_adjustment' do

o = SalaryAdjustment.new

o.id = 5

o.staff\_id = 'S0010'

o.inc = 60

o.month = 9

o.year = 2009

assert o.save

end

test 'should find salary\_adjustment' do

id = salary\_adjustment(:one).id

assert\_nothing\_raised { SalaryAdjustment.find(id) }

end

test 'should update salary\_adjustment' do

o = salary\_adjustment(:two)

assert o.update\_attributes(:inc => 70)

o.month = 8

assert o.save

o = SalaryAdjustment.find(o.id)

assert\_equal 8, o.month

end

test 'should destroy salary\_adjustment' do

o = salary\_adjustment(:one)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { SalaryAdjustment.find(o.id) }

end

test 'should not create a salary\_adjustment without required fields' do

o = SalaryAdjustment.new

assert !o.valid?

assert o.errors[:staff\_id].any?

assert o.errors[:inc].any?

assert o.errors[:month].any?

assert o.errors[:year].any?

assert\_equal ['Staff ID is required'], o.errors[:staff\_id]

assert\_equal ['Month is required', 'Month is invalid', 'Month is invalid'], o.errors[:month]

assert\_equal ['Year is required', 'Year is invalid'], o.errors[:year]

o.inc = -9

o.month = 13

o.year = 0

assert !o.valid?

assert\_equal ['Increment is invalid'], o.errors[:inc]

assert\_equal ['Month is invalid'], o.errors[:month]

assert\_equal ['Year is invalid'], o.errors[:year]

end

end

E:\workspace\payroll\_rails\test\unit\user\_test.rb

require 'test\_helper'

class UserTest < ActiveSupport::TestCase

test 'should create user' do

o = User.new

o.role = 1

o.username = 'beny'

o.status = true

o.pwd = 'benben'

assert o.save

end

test 'should find user' do

id = user(:ben).id

assert\_nothing\_raised { User.find(id) }

end

test 'should update user' do

o = user(:ken)

assert o.update\_attributes(:role => 2, :status => false)

o.password = 'bbb'

assert o.save

o = User.find(o.id)

assert\_equal 'bbb', o.password

end

test 'should destroy user' do

o = user(:ben)

o.destroy

assert\_raise(ActiveRecord::RecordNotFound) { User.find(o.id) }

end

test 'should not create a user without required fields' do

o = User.new

assert !o.valid?

assert o.errors[:username].any?

assert o.errors[:pwd].any?

assert\_equal ['Minimum is 3 characters'], o.errors[:username]

assert\_equal ['Minimum is 4 characters', 'Password is required'], o.errors[:pwd]

end

test 'should not create a user with duplicate username' do

o = User.new

o.username = 'admin'

assert !o.valid?

assert o.errors[:username].any?

assert\_equal ['Username admin already exist'], o.errors[:username]

end

test 'should not create/update user with wrong password confirmation' do

o = User.new(:username => 'Ken', :pwd => 'hhhh', :pwd\_confirmation => 'aaaa')

assert !o.valid?

assert o.errors[:pwd].any?

assert\_equal ["Password doesn't match confirmation"], o.errors[:pwd]

o = user(:ken)

assert !o.update\_attributes(:pwd => 'jjjj', :pwd\_confirmation => 'mmmm')

assert o.errors[:pwd].any?

assert\_equal ["Password doesn't match confirmation"], o.errors[:pwd]

end

end

E:\workspace\payroll\_rails\test\functional\admin\admin\_controller\_test.rb

require 'test\_helper'

class Admin::AdminControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

end

end

E:\workspace\payroll\_rails\test\functional\admin\attendance\_controller\_test.rb

require 'test\_helper'

class Admin::AttendanceControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

end

test 'should get list' do

login\_as :admin

get :list, { :work\_date => '12-01-2013', :employee => 'ben' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

end

E:\workspace\payroll\_rails\test\functional\admin\department\_controller\_test.rb

require 'test\_helper'

class Admin::DepartmentControllerTest < ActionController::TestCase

setup do

@dept = department(:one)

end

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

end

test 'should get list' do

login\_as :admin

get :list, { :keyword => 'r' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

test 'should get new' do

login\_as :admin

get :new

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:dept)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

end

test 'should create department' do

login\_as :admin

assert\_difference('Department.count') do

post :create, { :name => 'Sales' }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should get edit' do

login\_as :admin

get :edit, { :id => @dept.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:dept)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

end

test 'should update department' do

login\_as :admin

post :update, { :id => @dept.id, :name => 'Admin' }

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should destroy department' do

login\_as :admin

assert\_difference('Department.count', -1) do

post :destroy, { :id => [@dept.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 1 of 1', data['itemscount']

end

end

E:\workspace\payroll\_rails\test\functional\admin\designation\_controller\_test.rb

require 'test\_helper'

class Admin::DesignationControllerTest < ActionController::TestCase

setup do

@des = designation(:one)

end

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

end

test 'should get list' do

login\_as :admin

get :list, { :keyword => 'rammer' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

test 'should get new' do

login\_as :admin

get :new

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:designation)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

end

test 'should create designation' do

login\_as :admin

assert\_difference('Designation.count') do

post :create, { :title => 'Account', :desc => 'Accounting',

:note => 'account related tasks' }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should get edit' do

login\_as :admin

get :edit, { :id => @des.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:designation)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

end

test 'should update designation' do

login\_as :admin

post :update, { :id => @des.id, :title => 'Audit', :desc => 'Audit job',

:note => 'Audit task' }

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should destroy designation' do

login\_as :admin

assert\_difference('Designation.count', -1) do

post :destroy, { :id => [@des.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 1 of 1', data['itemscount']

end

end

E:\workspace\payroll\_rails\test\functional\admin\employee\_controller\_test.rb

require 'test\_helper'

class Admin::EmployeeControllerTest < ActionController::TestCase

setup do

@employee = employee(:one)

@employee\_contact = employee\_contact(:one)

@employee\_job = employee\_job(:one)

@employee\_salary = employee\_salary(:one)

@employee\_qualification = employee\_qualification(:one)

end

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

assert\_not\_nil assigns(:employmentstatus)

assert\_not\_nil assigns(:designation)

assert\_not\_nil assigns(:dept)

end

test 'should get list' do

login\_as :admin

get :list, { :staff\_id => 'C0001' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

test 'should get new' do

login\_as :admin

get :new

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_contact)

assert\_not\_nil assigns(:employee\_job)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:employee\_qualification)

assert\_not\_nil assigns(:form\_id)

assert\_not\_nil assigns(:users)

assert\_not\_nil assigns(:designations)

assert\_not\_nil assigns(:employment\_statuses)

assert\_not\_nil assigns(:job\_categories)

assert\_not\_nil assigns(:departments)

assert\_equal 'add-form', assigns(:form\_id)

end

test 'should create employee' do

login\_as :admin

assert\_difference('Employee.count') do

post :create, {

:employee => {

:staff\_id => 'C0003', :first\_name => 'wong', :middle\_name => 'yan',

:last\_name => 'kin', :new\_ic => '098455673', :old\_ic => '88744532',

:passport\_no => @employee.passport\_no, :gender => @employee.gender, :marital\_status => @employee.marital\_status,

:nationality => @employee.nationality, :dob => '15-08-1977', :place\_of\_birth => @employee.place\_of\_birth,

:race => @employee.race, :religion => @employee.religion, :is\_bumi => @employee.is\_bumi, :user\_id => @employee.user\_id

},

:employee\_contact => {

:id => @employee\_contact.id, :address\_1 => @employee\_contact.address\_1, :address\_2 => @employee\_contact.address\_2,

:address\_3 => @employee\_contact.address\_3, :city => @employee\_contact.city, :state => @employee\_contact.state,

:postcode => @employee\_contact.postcode, :country => @employee\_contact.country, :home\_phone => @employee\_contact.home\_phone,

:mobile\_phone => @employee\_contact.mobile\_phone, :work\_email => @employee\_contact.work\_email,

:other\_email => @employee\_contact.other\_email

},

:employee\_job => {

:id => @employee\_job.id, :designation\_id => @employee\_job.designation\_id, :department\_id => @employee\_job.department\_id,

:employment\_status\_id => @employee\_job.employment\_status\_id, :job\_category\_id => @employee\_job.job\_category\_id,

:join\_date => '09-03-2011', :confirm\_date => '09-06-2011'

},

:employee\_salary => {

:id => @employee\_salary.id, :salary => @employee\_salary.salary, :allowance => @employee\_salary.allowance,

:epf => @employee\_salary.epf, :socso => @employee\_salary.socso, :income\_tax => @employee\_salary.income\_tax,

:bank\_name => @employee\_salary.bank\_name, :bank\_acc\_no => @employee\_salary.bank\_acc\_no,

:bank\_acc\_type => @employee\_salary.bank\_acc\_type, :bank\_address => @employee\_salary.bank\_address,

:epf\_no => @employee\_salary.epf\_no, :socso\_no => @employee\_salary.socso\_no, :income\_tax\_no => @employee\_salary.income\_tax\_no,

:pay\_type => @employee\_salary.pay\_type

},

:employee\_qualification => {

:id => @employee\_qualification.id, :level => @employee\_qualification.level, :institute => @employee\_qualification.institute,

:major => @employee\_qualification.major, :year => 2010, :gpa => @employee\_qualification.gpa,

:start\_date => '03-04-2006', :end\_date => '02-04-2010'

}

}

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should get edit' do

login\_as :admin

get :edit, { :id => @employee.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_contact)

assert\_not\_nil assigns(:employee\_job)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:employee\_qualification)

assert\_not\_nil assigns(:form\_id)

assert\_not\_nil assigns(:users)

assert\_not\_nil assigns(:designations)

assert\_not\_nil assigns(:employment\_statuses)

assert\_not\_nil assigns(:job\_categories)

assert\_not\_nil assigns(:designations)

assert\_equal 'edit-form', assigns(:form\_id)

end

test 'should update employee' do

login\_as :admin

post :update,

{

:id => @employee.id,

:employee => {

:staff\_id => 'C0005', :first\_name => @employee.first\_name, :middle\_name => @employee.middle\_name,

:last\_name => @employee.last\_name, :new\_ic => @employee.new\_ic, :old\_ic => @employee.old\_ic,

:passport\_no => @employee.passport\_no, :gender => @employee.gender, :marital\_status => @employee.marital\_status,

:nationality => @employee.nationality, :dob => '15-08-1977', :place\_of\_birth => @employee.place\_of\_birth,

:race => @employee.race, :religion => @employee.religion, :is\_bumi => @employee.is\_bumi, :user\_id => @employee.user\_id

},

:employee\_contact => {

:id => @employee\_contact.id, :address\_1 => @employee\_contact.address\_1, :address\_2 => @employee\_contact.address\_2,

:address\_3 => @employee\_contact.address\_3, :city => @employee\_contact.city, :state => @employee\_contact.state,

:postcode => @employee\_contact.postcode, :country => @employee\_contact.country, :home\_phone => @employee\_contact.home\_phone,

:mobile\_phone => @employee\_contact.mobile\_phone, :work\_email => @employee\_contact.work\_email,

:other\_email => @employee\_contact.other\_email

},

:employee\_job => {

:id => @employee\_job.id, :designation\_id => @employee\_job.designation\_id, :department\_id => @employee\_job.department\_id,

:employment\_status\_id => @employee\_job.employment\_status\_id, :job\_category\_id => @employee\_job.job\_category\_id,

:join\_date => '09-03-2011', :confirm\_date => '09-06-2011'

},

:employee\_salary => {

:id => @employee\_salary.id, :salary => @employee\_salary.salary, :allowance => @employee\_salary.allowance,

:epf => @employee\_salary.epf, :socso => @employee\_salary.socso, :income\_tax => @employee\_salary.income\_tax,

:bank\_name => @employee\_salary.bank\_name, :bank\_acc\_no => @employee\_salary.bank\_acc\_no,

:bank\_acc\_type => @employee\_salary.bank\_acc\_type, :bank\_address => @employee\_salary.bank\_address,

:epf\_no => @employee\_salary.epf\_no, :socso\_no => @employee\_salary.socso\_no, :income\_tax\_no => @employee\_salary.income\_tax\_no,

:pay\_type => @employee\_salary.pay\_type

},

:employee\_qualification => {

:id => @employee\_qualification.id, :level => @employee\_qualification.level, :institute => @employee\_qualification.institute,

:major => @employee\_qualification.major, :year => 2010, :gpa => @employee\_qualification.gpa,

:start\_date => '03-04-2006', :end\_date => '02-04-2010'

}

}

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should destroy employee' do

login\_as :admin

assert\_difference('Employee.count', -1) do

post :destroy, { :id => [@employee.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 1 of 1', data['itemscount']

end

end

E:\workspace\payroll\_rails\test\functional\admin\employment\_status\_controller\_test.rb

require 'test\_helper'

class Admin::EmploymentStatusControllerTest < ActionController::TestCase

setup do

@employment\_status = employment\_status(:one)

end

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

end

test 'should get list' do

login\_as :admin

get :list, { :keyword => 'bat' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

test 'should get new' do

login\_as :admin

get :new

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:empstatus)

end

test 'should create employment status' do

login\_as :admin

assert\_difference('EmploymentStatus.count') do

post :create, { :name => 'Contract' }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should get edit' do

login\_as :admin

get :edit, { :id => @employment\_status.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:empstatus)

end

test 'should update employment status' do

login\_as :admin

post :update, { :id => @employment\_status.id, :name => 'Part time' }

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should destroy employment status' do

login\_as :admin

assert\_difference('EmploymentStatus.count', -1) do

post :destroy, { :id => [@employment\_status.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 1 of 1', data['itemscount']

end

end

E:\workspace\payroll\_rails\test\functional\admin\hourly\_payroll\_chart\_controller\_test.rb

require 'test\_helper'

class Admin::HourlyPayrollChartControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

end

test 'should get data' do

login\_as :admin

get :data

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

E:\workspace\payroll\_rails\test\functional\admin\job\_category\_controller\_test.rb

require 'test\_helper'

class Admin::JobCategoryControllerTest < ActionController::TestCase

setup do

@jobcat = job\_category(:one)

end

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

end

test 'should get list' do

login\_as :admin

get :list, { :keyword => 'les' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

test 'should get new' do

login\_as :admin

get :new

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:jobcat)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

end

test 'should create job category' do

login\_as :admin

assert\_difference('JobCategory.count') do

post :create, { :name => 'Management' }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should get edit' do

login\_as :admin

get :edit, { :id => @jobcat.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:jobcat)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

end

test 'should update job category' do

login\_as :admin

post :update, { :id => @jobcat.id, :name => 'Accounting' }

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should destroy job category' do

login\_as :admin

assert\_difference('JobCategory.count', -1) do

post :destroy, { :id => [@jobcat.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 1 of 1', data['itemscount']

end

end

E:\workspace\payroll\_rails\test\functional\admin\overtime\_chart\_controller\_test.rb

require 'test\_helper'

class Admin::OvertimeChartControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

end

test 'should get data' do

login\_as :admin

get :data

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

E:\workspace\payroll\_rails\test\functional\admin\overtime\_rate\_controller\_test.rb

require 'test\_helper'

class Admin::OvertimeRateControllerTest < ActionController::TestCase

setup do

@rate = overtime\_rate(:one)

end

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

end

test 'should get list' do

login\_as :admin

get :list, { :year => 2008 }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

test 'should get new' do

login\_as :admin

get :new

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:rate)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

end

test 'should create overtime rate' do

login\_as :admin

assert\_difference('OvertimeRate.count') do

post :create, { :duration => @rate.duration, :year => 2011, :pay\_rate => @rate.pay\_rate }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should get edit' do

login\_as :admin

get :edit, { :id => @rate.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:rate)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

end

test 'should update overtime rate' do

login\_as :admin

post :update, { :id => @rate.id, :duration => @rate.duration, :year => 2005,

:pay\_rate => @rate.pay\_rate }

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should destroy overtime rate' do

login\_as :admin

assert\_difference('OvertimeRate.count', -1) do

post :destroy, { :id => [@rate.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 1 of 1', data['itemscount']

end

end

E:\workspace\payroll\_rails\test\functional\admin\pay\_rate\_controller\_test.rb

require 'test\_helper'

class Admin::PayRateControllerTest < ActionController::TestCase

setup do

@rate = pay\_rate(:one)

end

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

end

test 'should get list' do

login\_as :admin

get :list, { :staff\_id => 'C0001' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

test 'should get new' do

login\_as :admin

get :new

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:payrate)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

end

test 'should create pay rate' do

login\_as :admin

assert\_difference('PayRate.count') do

post :create, { :staff\_id => 'C0007',

:month => @rate.month, :year => @rate.year,

:pay\_rate => @rate.hourly\_pay\_rate }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should get edit' do

login\_as :admin

get :edit, { :id => @rate.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:payrate)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

end

test 'should update pay rate' do

login\_as :admin

post :update, { :id => @rate.id, :staff\_id => 'C0006', :month => 6,

:year => 2012, :pay\_rate => 3.4 }

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should destroy pay rate' do

login\_as :admin

assert\_difference('PayRate.count', -1) do

post :destroy, { :id => [@rate.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 1 of 1', data['itemscount']

end

end

E:\workspace\payroll\_rails\test\functional\admin\payslip\_controller\_test.rb

require 'test\_helper'

class Admin::PayslipControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

assert\_not\_nil assigns(:employmentstatus)

assert\_not\_nil assigns(:designation)

assert\_not\_nil assigns(:dept)

end

test 'should get list' do

login\_as :admin

get :list, { :employee => 'ben' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

test 'should get monthly payslip' do

login\_as :admin

get :payslip, { :id => 1 }

assert\_response :success

assert\_template 'payslip\_monthly'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:period)

assert\_not\_nil assigns(:total\_overtime)

assert\_not\_nil assigns(:total\_overtime\_earnings)

assert\_not\_nil assigns(:adjustment)

assert\_not\_nil assigns(:total\_earnings)

assert\_not\_nil assigns(:total\_deduct)

assert\_not\_nil assigns(:nett\_salary)

assert\_not\_nil assigns(:basic\_pay)

end

test 'should get hourly payslip' do

login\_as :admin

get :payslip, { :id => 2 }

assert\_response :success

assert\_template 'payslip\_hourly'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:total\_earnings)

assert\_not\_nil assigns(:total\_hours)

assert\_not\_nil assigns(:hourly\_pay\_rate)

assert\_not\_nil assigns(:total\_deduct)

assert\_not\_nil assigns(:nett\_salary)

end

end

E:\workspace\payroll\_rails\test\functional\admin\salary\_adjustment\_controller\_test.rb

require 'test\_helper'

class Admin::SalaryAdjustmentControllerTest < ActionController::TestCase

setup do

@adj = salary\_adjustment(:one)

end

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

end

test 'should get list' do

login\_as :admin

get :list, { :staff\_id => 'C0001' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

test 'should get new' do

login\_as :admin

get :new

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:adj)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

end

test 'should create salary adjustment' do

login\_as :admin

assert\_difference('SalaryAdjustment.count') do

post :create, { :staff\_id => 'C0004', :inc => 300, :month => 8, :year => 2013 }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should get edit' do

login\_as :admin

get :edit, { :id => @adj.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:adj)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

end

test 'should update salary adjustment' do

login\_as :admin

post :update, { :id => @adj.id, :staff\_id => 'C0003', :inc => 250,

:month => 10, :year => 2013 }

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should destroy salary adjustment' do

login\_as :admin

assert\_difference('SalaryAdjustment.count', -1) do

post :destroy, { :id => [@adj.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 1 of 1', data['itemscount']

end

end

E:\workspace\payroll\_rails\test\functional\admin\total\_work\_hours\_chart\_controller\_test.rb

require 'test\_helper'

class Admin::TotalWorkHoursChartControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

end

test 'should get data' do

login\_as :admin

get :data

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

E:\workspace\payroll\_rails\test\functional\admin\user\_controller\_test.rb

require 'test\_helper'

class Admin::UserControllerTest < ActionController::TestCase

setup do

@user = user(:ben)

end

test 'should get index' do

login\_as :admin

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

end

test 'should get list' do

login\_as :admin

get :list, { :username => 'en' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

test 'should get new' do

login\_as :admin

get :new

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:user)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

end

test 'should create user' do

login\_as :admin

assert\_difference('User.count') do

post :create, { :role => 2, :username => 'mary', :status => '1',

:pwd => 'abc111', :pwdconfirm => 'abc111' }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should get edit' do

login\_as :admin

get :edit, { :id => @user.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:user)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

end

test 'should update user' do

login\_as :admin

post :update, { :id => @user.id, :role => 2, :username => 'mary',

:status => '1', :pwd => 'kim111', :pwdconfirm => 'kim111' }

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

test 'should destroy user' do

login\_as :admin

assert\_difference('User.count', -1) do

post :destroy, { :id => [@user.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 2 of 2', data['itemscount']

end

end

E:\workspace\payroll\_rails\test\functional\user\contact\_controller\_test.rb

require 'test\_helper'

class User::ContactControllerTest < ActionController::TestCase

setup do

@employee\_contact = employee\_contact(:two)

end

test 'should get index' do

login\_as :ben

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:employee\_contact)

end

test 'should update' do

login\_as :ben

post :update, {

:employee\_contact => {

:address\_1 => @employee\_contact.address\_1, :address\_2 => @employee\_contact.address\_2,

:address\_3 => @employee\_contact.address\_3, :city => @employee\_contact.city, :state => @employee\_contact.state,

:postcode => @employee\_contact.postcode, :country => @employee\_contact.country, :home\_phone => @employee\_contact.home\_phone,

:mobile\_phone => @employee\_contact.mobile\_phone, :work\_email => @employee\_contact.work\_email,

:other\_email => @employee\_contact.other\_email

},

}

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

end

E:\workspace\payroll\_rails\test\functional\user\hourly\_payroll\_chart\_controller\_test.rb

require 'test\_helper'

class User::HourlyPayrollChartControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :ben

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

end

test 'should get data' do

login\_as :ben

get :data

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

E:\workspace\payroll\_rails\test\functional\user\info\_controller\_test.rb

require 'test\_helper'

class User::InfoControllerTest < ActionController::TestCase

setup do

@employee = employee(:one)

end

test 'should get index' do

login\_as :ben

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:user)

end

test 'should update' do

login\_as :ben

post :update, {

:employee => {

:first\_name => 'wong', :middle\_name => 'yan', :last\_name => 'kin', :new\_ic => '098455673', :old\_ic => '88744532',

:passport\_no => @employee.passport\_no, :gender => @employee.gender, :marital\_status => @employee.marital\_status,

:nationality => @employee.nationality, :dob => '15-08-1977', :place\_of\_birth => @employee.place\_of\_birth,

:race => @employee.race, :religion => @employee.religion, :is\_bumi => @employee.is\_bumi

}

}

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

end

E:\workspace\payroll\_rails\test\functional\user\job\_controller\_test.rb

require 'test\_helper'

class User::JobControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :ben

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:employee\_job)

end

end

E:\workspace\payroll\_rails\test\functional\user\overtime\_chart\_controller\_test.rb

require 'test\_helper'

class User::OvertimeChartControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :ben

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

end

test 'should get data' do

login\_as :ben

get :data

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

E:\workspace\payroll\_rails\test\functional\user\payslip\_controller\_test.rb

require 'test\_helper'

class User::PayslipControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :ben

get :index

assert\_response :success

assert\_template 'index'

end

test 'should get monthly payslip' do

login\_as :ben

get :payslip, { :id => 2 }

assert\_response :success

assert\_template 'admin/payslip/payslip\_monthly'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:period)

assert\_not\_nil assigns(:total\_overtime)

assert\_not\_nil assigns(:total\_overtime\_earnings)

assert\_not\_nil assigns(:adjustment)

assert\_not\_nil assigns(:total\_earnings)

assert\_not\_nil assigns(:total\_deduct)

assert\_not\_nil assigns(:nett\_salary)

assert\_not\_nil assigns(:basic\_pay)

end

test 'should get hourly payslip' do

login\_as :ken

get :payslip, { :id => 3 }

assert\_response :success

assert\_template 'admin/payslip/payslip\_hourly'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:total\_earnings)

assert\_not\_nil assigns(:total\_hours)

assert\_not\_nil assigns(:hourly\_pay\_rate)

assert\_not\_nil assigns(:total\_deduct)

assert\_not\_nil assigns(:nett\_salary)

end

end

E:\workspace\payroll\_rails\test\functional\user\qualification\_controller\_test.rb

require 'test\_helper'

class User::QualificationControllerTest < ActionController::TestCase

setup do

@employee\_qualification = employee\_qualification(:one)

end

test 'should get index' do

login\_as :ben

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:employee\_qualification)

end

test 'should update' do

login\_as :ben

post :update, {

:employee\_qualification => {

:level => @employee\_qualification.level, :institute => @employee\_qualification.institute,

:major => @employee\_qualification.major, :year => 2010, :gpa => @employee\_qualification.gpa,

:start\_date => '03-04-2006', :end\_date => '02-04-2010'

},

}

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

end

E:\workspace\payroll\_rails\test\functional\user\salary\_controller\_test.rb

require 'test\_helper'

class User::SalaryControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :ben

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:adjustment)

assert\_not\_nil assigns(:basic\_pay)

end

end

E:\workspace\payroll\_rails\test\functional\user\total\_work\_hours\_chart\_controller\_test.rb

require 'test\_helper'

class User::TotalWorkHoursChartControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :ken

get :index

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

end

test 'should get data' do

login\_as :ken

get :data

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

E:\workspace\payroll\_rails\test\functional\user\user\_controller\_test.rb

require 'test\_helper'

class User::UserControllerTest < ActionController::TestCase

test 'should get index' do

login\_as :ben

get :index

assert\_response :success

end

end

E:\workspace\payroll\_rails\test\integration\admin\_attendance\_flows\_test.rb

require 'test\_helper'

class AdminAttendanceFlowsTest < ActionDispatch::IntegrationTest

setup do

@admin = user(:admin)

end

test 'browse attendance' do

a = admin\_user(AdminAttendanceFlows)

a.logs\_in @admin.username, 'secret'

a.browse

a.logs\_out

end

private

module AdminAttendanceFlows

def browse

get admin\_att\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

get admin\_att\_list\_path, { :work\_date => '12-01-2013', :employee => 'ben' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_department\_flows\_test.rb

require 'test\_helper'

class AdminDepartmentFlowsTest < ActionDispatch::IntegrationTest

setup do

@admin = user(:admin)

@dept = department(:one)

end

test 'login, search, create, update, and delete department' do

a = admin\_user(AdminDepartmentFlows)

a.logs\_in @admin.username, 'secret'

a.search

a.create :name => 'Sales'

a.update @dept, { :id => @dept.id, :name => 'Admin' }

a.destroy @dept

a.logs\_out

end

private

module AdminDepartmentFlows

def search

get admin\_dept\_path

assert\_template 'index'

assert\_not\_nil assigns(:data)

get admin\_dept\_list\_path, { :keyword => 'r' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

def create(obj\_hash)

get admin\_dept\_new\_path

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:dept)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

assert\_difference('Department.count') do

post admin\_dept\_create\_path, obj\_hash

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def update(o, obj\_hash)

get admin\_dept\_edit\_path, { :id => o.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:dept)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

post admin\_dept\_update\_path, obj\_hash

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def destroy(o)

assert\_difference('Department.count', -1) do

post admin\_dept\_delete\_path, { :id => [o.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 2 of 2', data['itemscount']

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_designation\_flows\_test.rb

require 'test\_helper'

class AdminDesignationFlowsTest < ActionDispatch::IntegrationTest

setup do

@admin = user(:admin)

@des = designation(:one)

end

test 'login, search, create, update, and delete designation' do

a = admin\_user(AdminDesignationFlows)

a.logs\_in @admin.username, 'secret'

a.search

a.create :title => 'Account', :desc => 'Accounting',

:note => 'account related tasks'

a.update @des, { :id => @des.id, :title => 'Audit', :desc => 'Audit job',

:note => 'Audit task' }

a.destroy @des

a.logs\_out

end

private

module AdminDesignationFlows

def search

get admin\_designation\_path

assert\_template 'index'

assert\_not\_nil assigns(:data)

get admin\_designation\_list\_path, { :keyword => 'rammer' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

def create(obj\_hash)

get admin\_designation\_new\_path

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:designation)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

assert\_difference('Designation.count') do

post admin\_designation\_create\_path, obj\_hash

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def update(o, obj\_hash)

get admin\_designation\_edit\_path, { :id => o.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:designation)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

post admin\_designation\_update\_path, obj\_hash

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def destroy(o)

assert\_difference('Designation.count', -1) do

post admin\_designation\_delete\_path, { :id => [o.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 2 of 2', data['itemscount']

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_employee\_flows\_test.rb

require 'test\_helper'

class AdminEmployeeFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:admin)

@employee = employee(:one)

@employee\_contact = employee\_contact(:one)

@employee\_job = employee\_job(:one)

@employee\_salary = employee\_salary(:one)

@employee\_qualification = employee\_qualification(:one)

end

test '' do

a = admin\_user(AdminEmployeeFlows)

a.logs\_in @user.username, 'secret'

d = {

:employee => {

:staff\_id => 'C0003', :first\_name => 'wong', :middle\_name => 'yan',

:last\_name => 'kin', :new\_ic => '098455673', :old\_ic => '88744532',

:passport\_no => @employee.passport\_no, :gender => @employee.gender,

:marital\_status => @employee.marital\_status, :nationality => @employee.nationality,

:dob => '15-08-1977', :place\_of\_birth => @employee.place\_of\_birth,

:race => @employee.race, :religion => @employee.religion,

:is\_bumi => @employee.is\_bumi, :user\_id => @employee.user\_id

},

:employee\_contact => {

:id => @employee\_contact.id, :address\_1 => @employee\_contact.address\_1,

:address\_2 => @employee\_contact.address\_2,

:address\_3 => @employee\_contact.address\_3, :city => @employee\_contact.city,

:state => @employee\_contact.state, :postcode => @employee\_contact.postcode,

:country => @employee\_contact.country, :home\_phone => @employee\_contact.home\_phone,

:mobile\_phone => @employee\_contact.mobile\_phone,

:work\_email => @employee\_contact.work\_email,

:other\_email => @employee\_contact.other\_email

},

:employee\_job => {

:id => @employee\_job.id, :designation\_id => @employee\_job.designation\_id,

:department\_id => @employee\_job.department\_id,

:employment\_status\_id => @employee\_job.employment\_status\_id,

:job\_category\_id => @employee\_job.job\_category\_id, :join\_date => '09-03-2011',

:confirm\_date => '09-06-2011'

},

:employee\_salary => {

:id => @employee\_salary.id, :salary => @employee\_salary.salary,

:allowance => @employee\_salary.allowance, :epf => @employee\_salary.epf,

:socso => @employee\_salary.socso, :income\_tax => @employee\_salary.income\_tax,

:bank\_name => @employee\_salary.bank\_name,

:bank\_acc\_no => @employee\_salary.bank\_acc\_no,

:bank\_acc\_type => @employee\_salary.bank\_acc\_type,

:bank\_address => @employee\_salary.bank\_address,

:epf\_no => @employee\_salary.epf\_no, :socso\_no => @employee\_salary.socso\_no,

:income\_tax\_no => @employee\_salary.income\_tax\_no,

:pay\_type => @employee\_salary.pay\_type

},

:employee\_qualification => {

:id => @employee\_qualification.id, :level => @employee\_qualification.level,

:institute => @employee\_qualification.institute,

:major => @employee\_qualification.major, :year => 2010,

:gpa => @employee\_qualification.gpa, :start\_date => '03-04-2006',

:end\_date => '02-04-2010'

}

}

a.create d

a.update @employee,

{

:id => @employee.id,

:employee => {

:staff\_id => 'C0005', :first\_name => @employee.first\_name, :middle\_name => @employee.middle\_name,

:last\_name => @employee.last\_name, :new\_ic => @employee.new\_ic, :old\_ic => @employee.old\_ic,

:passport\_no => @employee.passport\_no, :gender => @employee.gender, :marital\_status => @employee.marital\_status,

:nationality => @employee.nationality, :dob => '15-08-1977', :place\_of\_birth => @employee.place\_of\_birth,

:race => @employee.race, :religion => @employee.religion, :is\_bumi => @employee.is\_bumi, :user\_id => @employee.user\_id

},

:employee\_contact => {

:id => @employee\_contact.id, :address\_1 => @employee\_contact.address\_1, :address\_2 => @employee\_contact.address\_2,

:address\_3 => @employee\_contact.address\_3, :city => @employee\_contact.city, :state => @employee\_contact.state,

:postcode => @employee\_contact.postcode, :country => @employee\_contact.country, :home\_phone => @employee\_contact.home\_phone,

:mobile\_phone => @employee\_contact.mobile\_phone, :work\_email => @employee\_contact.work\_email,

:other\_email => @employee\_contact.other\_email

},

:employee\_job => {

:id => @employee\_job.id, :designation\_id => @employee\_job.designation\_id, :department\_id => @employee\_job.department\_id,

:employment\_status\_id => @employee\_job.employment\_status\_id, :job\_category\_id => @employee\_job.job\_category\_id,

:join\_date => '09-03-2011', :confirm\_date => '09-06-2011'

},

:employee\_salary => {

:id => @employee\_salary.id, :salary => @employee\_salary.salary, :allowance => @employee\_salary.allowance,

:epf => @employee\_salary.epf, :socso => @employee\_salary.socso, :income\_tax => @employee\_salary.income\_tax,

:bank\_name => @employee\_salary.bank\_name, :bank\_acc\_no => @employee\_salary.bank\_acc\_no,

:bank\_acc\_type => @employee\_salary.bank\_acc\_type, :bank\_address => @employee\_salary.bank\_address,

:epf\_no => @employee\_salary.epf\_no, :socso\_no => @employee\_salary.socso\_no, :income\_tax\_no => @employee\_salary.income\_tax\_no,

:pay\_type => @employee\_salary.pay\_type

},

:employee\_qualification => {

:id => @employee\_qualification.id, :level => @employee\_qualification.level, :institute => @employee\_qualification.institute,

:major => @employee\_qualification.major, :year => 2010, :gpa => @employee\_qualification.gpa,

:start\_date => '03-04-2006', :end\_date => '02-04-2010'

}

}

a.destroy @employee

a.logs\_out

end

private

module AdminEmployeeFlows

def search

get admin\_employee\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

assert\_not\_nil assigns(:employmentstatus)

assert\_not\_nil assigns(:designation)

assert\_not\_nil assigns(:dept)

get admin\_employee\_list\_path, { :staff\_id => 'C0001' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

def create(obj\_hash)

get admin\_employee\_new\_path

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_contact)

assert\_not\_nil assigns(:employee\_job)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:employee\_qualification)

assert\_not\_nil assigns(:form\_id)

assert\_not\_nil assigns(:users)

assert\_not\_nil assigns(:designations)

assert\_not\_nil assigns(:employment\_statuses)

assert\_not\_nil assigns(:job\_categories)

assert\_not\_nil assigns(:departments)

assert\_equal 'add-form', assigns(:form\_id)

assert\_difference('Employee.count') do

post admin\_employee\_create\_path, obj\_hash

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def update(o, obj\_hash)

get admin\_employee\_edit\_path, { :id => o.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_contact)

assert\_not\_nil assigns(:employee\_job)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:employee\_qualification)

assert\_not\_nil assigns(:form\_id)

assert\_not\_nil assigns(:users)

assert\_not\_nil assigns(:designations)

assert\_not\_nil assigns(:employment\_statuses)

assert\_not\_nil assigns(:job\_categories)

assert\_not\_nil assigns(:designations)

assert\_equal 'edit-form', assigns(:form\_id)

post admin\_employee\_update\_path, obj\_hash

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def destroy(o)

assert\_difference('Employee.count', -1) do

post admin\_employee\_delete\_path, { :id => [o.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 2 of 2', data['itemscount']

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_employment\_status\_flows\_test.rb

require 'test\_helper'

class AdminEmploymentStatusFlowsTest < ActionDispatch::IntegrationTest

setup do

@admin = user(:admin)

@employment\_status = employment\_status(:one)

end

test 'login, search, create, update, and delete employment status' do

a = admin\_user(AdminEmploymentStatusFlows)

a.logs\_in @admin.username, 'secret'

a.search

a.create :name => 'Contract'

a.update @employment\_status, { :id => @employment\_status.id, :name => 'Part time' }

a.destroy @employment\_status

a.logs\_out

end

private

module AdminEmploymentStatusFlows

def search

get admin\_empstatus\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

get admin\_empstatus\_list\_path, { :keyword => 'bat' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

def create(obj\_hash)

get admin\_empstatus\_new\_path

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:empstatus)

assert\_difference('EmploymentStatus.count') do

post admin\_empstatus\_create\_path, obj\_hash

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def update(o, obj\_hash)

get admin\_empstatus\_edit\_path, { :id => o.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:empstatus)

post admin\_empstatus\_update\_path, obj\_hash

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def destroy(o)

assert\_difference('EmploymentStatus.count', -1) do

post admin\_empstatus\_delete\_path, { :id => [o.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 2 of 2', data['itemscount']

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_hourly\_payroll\_chart\_flows\_test.rb

require 'test\_helper'

class AdminHourlyPayrollChartFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:admin)

end

test 'browse chart' do

a = admin\_user(AdminHourlyPayrollChartFlows)

a.logs\_in @user.username, 'secret'

a.browse

a.logs\_out

end

private

module AdminHourlyPayrollChartFlows

def browse

get admin\_hourly\_chart\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

get admin\_hourly\_chart\_data\_path

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_job\_category\_flows\_test.rb

require 'test\_helper'

class AdminJobCategoryFlowsTest < ActionDispatch::IntegrationTest

setup do

@admin = user(:admin)

@jobcat = job\_category(:one)

end

test 'login, search, create, update, and delete job category' do

a = admin\_user(AdminJobCategoryFlows)

a.logs\_in @admin.username, 'secret'

a.search

a.create :name => 'Management'

a.update @jobcat, { :id => @jobcat.id, :name => 'Accounting' }

a.destroy @jobcat

a.logs\_out

end

private

module AdminJobCategoryFlows

def search

get admin\_jobcat\_path

assert\_template 'index'

assert\_not\_nil assigns(:data)

get admin\_jobcat\_list\_path, { :keyword => 'les' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

def create(obj\_hash)

get admin\_jobcat\_new\_path

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:jobcat)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

assert\_difference('JobCategory.count') do

post admin\_jobcat\_create\_path, obj\_hash

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def update(o, obj\_hash)

get admin\_jobcat\_edit\_path, { :id => o.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:jobcat)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

post admin\_jobcat\_update\_path, obj\_hash

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def destroy(o)

assert\_difference('JobCategory.count', -1) do

post admin\_jobcat\_delete\_path, { :id => [o.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 2 of 2', data['itemscount']

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_overtime\_chart\_flows\_test.rb

require 'test\_helper'

class AdminOvertimeChartFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:admin)

end

test 'browse chart' do

a = admin\_user(AdminOvertimeChartFlows)

a.logs\_in @user.username, 'secret'

a.browse

a.logs\_out

end

private

module AdminOvertimeChartFlows

def browse

get admin\_overtime\_chart\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

get admin\_overtime\_chart\_data\_path

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_overtime\_rate\_flows\_test.rb

require 'test\_helper'

class AdminOvertimeRateFlowsTest < ActionDispatch::IntegrationTest

setup do

@admin = user(:admin)

@rate = overtime\_rate(:one)

end

test 'login, search, create, update, and delete overtime rate' do

a = admin\_user(AdminOvertimeRateFlows)

a.logs\_in @admin.username, 'secret'

a.search

a.create :duration => @rate.duration, :year => 2011, :pay\_rate => @rate.pay\_rate

a.update @rate, { :id => @rate.id, :duration => @rate.duration, :year => 2005,

:pay\_rate => @rate.pay\_rate }

a.destroy @rate

a.logs\_out

end

private

module AdminOvertimeRateFlows

def search

get admin\_overtime\_rate\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

get admin\_overtime\_rate\_list\_path, { :year => 2008 }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

def create(obj\_hash)

get admin\_overtime\_rate\_new\_path

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:rate)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

assert\_difference('OvertimeRate.count') do

post admin\_overtime\_rate\_create\_path, obj\_hash

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def update(o, obj\_hash)

get admin\_overtime\_rate\_edit\_path, { :id => o.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:rate)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

post admin\_overtime\_rate\_update\_path, obj\_hash

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def destroy(o)

assert\_difference('OvertimeRate.count', -1) do

post admin\_overtime\_rate\_delete\_path, { :id => [o.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 2 of 2', data['itemscount']

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_pay\_rate\_flows\_test.rb

require 'test\_helper'

class AdminPayRateFlowsTest < ActionDispatch::IntegrationTest

setup do

@admin = user(:admin)

@rate = pay\_rate(:one)

end

test 'login, search, create, update, and delete pay rate' do

a = admin\_user(AdminPayRateFlows)

a.logs\_in @admin.username, 'secret'

a.search

a.create :staff\_id => 'C0007',

:month => @rate.month, :year => @rate.year,

:pay\_rate => @rate.hourly\_pay\_rate

a.update @rate, { :id => @rate.id, :staff\_id => 'C0006', :month => 6,

:year => 2012, :pay\_rate => 3.4 }

a.destroy @rate

a.logs\_out

end

private

module AdminPayRateFlows

def search

get admin\_payrate\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

get admin\_payrate\_list\_path, { :staff\_id => 'C0001' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

def create(obj\_hash)

get admin\_payrate\_new\_path

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:payrate)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

assert\_difference('PayRate.count') do

post admin\_payrate\_create\_path, obj\_hash

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def update(o, obj\_hash)

get admin\_payrate\_edit\_path, { :id => o.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:payrate)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

post admin\_payrate\_update\_path, obj\_hash

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def destroy(o)

assert\_difference('PayRate.count', -1) do

post admin\_payrate\_delete\_path, { :id => [o.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 2 of 2', data['itemscount']

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_payslip\_flows\_test.rb

require 'test\_helper'

class AdminPayslipFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:admin)

end

test 'browse and view payslip' do

a = admin\_user(AdminPayslipFlows)

a.logs\_in @user.username, 'secret'

a.browse

a.logs\_out

end

private

module AdminPayslipFlows

def browse

get admin\_payslip\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

assert\_not\_nil assigns(:employmentstatus)

assert\_not\_nil assigns(:designation)

assert\_not\_nil assigns(:dept)

get admin\_payslip\_list\_path, { :employee => 'ben' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

get admin\_payslip\_slip\_path, { :id => 1 }

assert\_response :success

assert\_template 'payslip\_monthly'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:period)

assert\_not\_nil assigns(:total\_overtime)

assert\_not\_nil assigns(:total\_overtime\_earnings)

assert\_not\_nil assigns(:adjustment)

assert\_not\_nil assigns(:total\_earnings)

assert\_not\_nil assigns(:total\_deduct)

assert\_not\_nil assigns(:nett\_salary)

assert\_not\_nil assigns(:basic\_pay)

get admin\_payslip\_slip\_path, { :id => 2 }

assert\_response :success

assert\_template 'payslip\_hourly'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:total\_earnings)

assert\_not\_nil assigns(:total\_hours)

assert\_not\_nil assigns(:hourly\_pay\_rate)

assert\_not\_nil assigns(:total\_deduct)

assert\_not\_nil assigns(:nett\_salary)

end

end

end

require 'test\_helper'

E:\workspace\payroll\_rails\test\integration\admin\_salary\_adjustment\_flows\_test.rb

class AdminSalaryAdjustmentFlowsTest < ActionDispatch::IntegrationTest

setup do

@admin = user(:admin)

@adj = salary\_adjustment(:one)

end

test 'login, search, create, update, and delete salary adjustment' do

a = admin\_user(AdminSalaryAdjustmentFlows)

a.logs\_in @admin.username, 'secret'

a.search

a.create :staff\_id => 'C0003', :inc => 300, :month => 8, :year => 2013

a.update @adj, { :id => @adj.id, :staff\_id => 'C0003', :inc => 250,

:month => 10, :year => 2013 }

a.destroy @adj

a.logs\_out

end

private

module AdminSalaryAdjustmentFlows

def search

get admin\_salaryadj\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:data)

get admin\_salaryadj\_list\_path, { :staff\_id => 'C0001' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

def create(obj\_hash)

get admin\_salaryadj\_new\_path

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:adj)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

assert\_difference('SalaryAdjustment.count') do

post admin\_salaryadj\_create\_path, obj\_hash

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def update(o, obj\_hash)

get admin\_salaryadj\_edit\_path, { :id => o.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:adj)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

post admin\_salaryadj\_update\_path, obj\_hash

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def destroy(o)

assert\_difference('SalaryAdjustment.count', -1) do

post admin\_salaryadj\_delete\_path, { :id => [o.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 2 of 2', data['itemscount']

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_total\_work\_hours\_chart\_flows\_test.rb

require 'test\_helper'

class AdminTotalWorkHoursChartFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:admin)

end

test 'browse chart' do

a = admin\_user(AdminTotalWorkHoursChartFlows)

a.logs\_in @user.username, 'secret'

a.browse

a.logs\_out

end

private

module AdminTotalWorkHoursChartFlows

def browse

get admin\_workhours\_chart\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

get admin\_workhours\_chart\_data\_path

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

end

E:\workspace\payroll\_rails\test\integration\admin\_user\_flows\_test.rb

require 'test\_helper'

class AdminUserFlowsTest < ActionDispatch::IntegrationTest

setup do

@admin = user(:admin)

@user = user(:ken)

end

test 'login, search, create, update, and delete user' do

a = admin\_user(AdminUserFlows)

a.logs\_in @admin.username, 'secret'

a.search

a.create :role => 2, :username => 'jane', :status => '1',

:pwd => 'abc111', :pwdconfirm => 'abc111'

a.update @user, { :id => @user.id, :role => 2, :username => 'pauline', :status => '1',

:pwd => 'kim111', :pwdconfirm => 'kim111' }

a.destroy @user

a.logs\_out

end

private

module AdminUserFlows

def search

get admin\_user\_path

assert\_template 'index'

assert\_not\_nil assigns(:data)

get admin\_user\_list\_path, { :username => 'en' }

assert\_response :success

assert\_template 'list'

assert\_not\_nil assigns(:data)

end

def create(obj\_hash)

get admin\_user\_new\_path

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:user)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'add-form', assigns(:form\_id)

assert\_difference('User.count') do

post admin\_user\_create\_path, obj\_hash

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def update(o, obj\_hash)

get admin\_user\_edit\_path, { :id => o.id }

assert\_response :success

assert\_template 'form'

assert\_not\_nil assigns(:user)

assert\_not\_nil assigns(:form\_id)

assert\_equal 'edit-form', assigns(:form\_id)

post admin\_user\_update\_path, obj\_hash

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

def destroy(o)

assert\_difference('User.count', -1) do

post admin\_user\_delete\_path, { :id => [o.id] }

end

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

assert\_equal '1 to 3 of 3', data['itemscount']

end

end

end

E:\workspace\payroll\_rails\test\integration\user\_contact\_flows\_test.rb

require 'test\_helper'

class UserContactFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:ben)

@employee\_contact = employee\_contact(:two)

end

test 'view and update contact' do

a = normal\_user(UserContactFlows)

a.logs\_in @user.username, 'secret'

a.update @employee\_contact

a.logs\_out

end

private

module UserContactFlows

def update(o)

get user\_contact\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:employee\_contact)

post user\_contact\_update\_path, {

:employee\_contact => {

:address\_1 => o.address\_1, :address\_2 => o.address\_2,

:address\_3 => o.address\_3, :city => o.city, :state => o.state,

:postcode => o.postcode, :country => o.country, :home\_phone => o.home\_phone,

:mobile\_phone => o.mobile\_phone, :work\_email => o.work\_email,

:other\_email => o.other\_email

},

}

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

end

end

E:\workspace\payroll\_rails\test\integration\user\_hourly\_payroll\_chart\_flows\_test.rb

require 'test\_helper'

class UserHourlyPayrollChartFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:ken)

end

test 'browse chart' do

a = normal\_user(UserHourlyPayrollChartFlows)

a.logs\_in @user.username, 'secret'

a.browse

a.logs\_out

end

private

module UserHourlyPayrollChartFlows

def browse

get user\_hourly\_chart\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

get user\_hourly\_chart\_data\_path

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

end

E:\workspace\payroll\_rails\test\integration\user\_info\_flows\_test.rb

require 'test\_helper'

class UserInfoFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:ben)

@employee = employee(:one)

end

test 'view and update info' do

a = normal\_user(UserInfoFlows)

a.logs\_in @user.username, 'secret'

a.update @employee

a.logs\_out

end

private

module UserInfoFlows

def update(o)

get user\_info\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:user)

post user\_info\_update\_path, {

:employee => {

:first\_name => 'wong', :middle\_name => 'yan', :last\_name => 'kin', :new\_ic => '098455673', :old\_ic => '88744532',

:passport\_no => o.passport\_no, :gender => o.gender, :marital\_status => o.marital\_status,

:nationality => o.nationality, :dob => '15-08-1977', :place\_of\_birth => o.place\_of\_birth,

:race => o.race, :religion => o.religion, :is\_bumi => o.is\_bumi

}

}

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

end

end

E:\workspace\payroll\_rails\test\integration\user\_job\_flows\_test.rb

require 'test\_helper'

class UserJobFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:ben)

end

test 'view job' do

a = normal\_user(UserJobFlows)

a.logs\_in @user.username, 'secret'

a.view

a.logs\_out

end

private

module UserJobFlows

def view

get user\_job\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:employee\_job)

end

end

end

E:\workspace\payroll\_rails\test\integration\user\_overtime\_chart\_flows\_test.rb

require 'test\_helper'

class UserOvertimeChartFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:ben)

end

test 'browse chart' do

a = normal\_user(UserOvertimeChartFlows)

a.logs\_in @user.username, 'secret'

a.browse

a.logs\_out

end

module UserOvertimeChartFlows

def browse

get user\_overtime\_chart\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

get user\_overtime\_chart\_data\_path

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

end

E:\workspace\payroll\_rails\test\integration\user\_payslip\_flows\_test.rb

require 'test\_helper'

class UserPayslipFlowsTest < ActionDispatch::IntegrationTest

setup do

@ben = user(:ben)

@ken = user(:ken)

end

test 'view payslip' do

a = normal\_user(UserPayslipFlows)

b = normal\_user(UserPayslipFlows)

a.logs\_in @ben.username, 'secret'

b.logs\_in @ken.username, 'secret'

a.view\_monthly

b.view\_hourly

a.logs\_out

b.logs\_out

end

module UserPayslipFlows

def view\_monthly

get user\_payslip\_path

assert\_response :success

assert\_template 'index'

get user\_payslip\_slip\_path, { :id => 2 }

assert\_response :success

assert\_template 'admin/payslip/payslip\_monthly'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:period)

assert\_not\_nil assigns(:total\_overtime)

assert\_not\_nil assigns(:total\_overtime\_earnings)

assert\_not\_nil assigns(:adjustment)

assert\_not\_nil assigns(:total\_earnings)

assert\_not\_nil assigns(:total\_deduct)

assert\_not\_nil assigns(:nett\_salary)

assert\_not\_nil assigns(:basic\_pay)

end

def view\_hourly

get user\_payslip\_path

assert\_response :success

assert\_template 'index'

get user\_payslip\_slip\_path, { :id => 3 }

assert\_response :success

assert\_template 'admin/payslip/payslip\_hourly'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:total\_earnings)

assert\_not\_nil assigns(:total\_hours)

assert\_not\_nil assigns(:hourly\_pay\_rate)

assert\_not\_nil assigns(:total\_deduct)

assert\_not\_nil assigns(:nett\_salary)

end

end

end

E:\workspace\payroll\_rails\test\integration\user\_qualification\_flows\_test.rb

require 'test\_helper'

class UserQualificationFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:ben)

@employee\_qualification = employee\_qualification(:one)

end

test 'view and update qualification' do

a = normal\_user(UserQualificationFlows)

a.logs\_in @user.username, 'secret'

a.update @employee\_qualification

a.logs\_out

end

private

module UserQualificationFlows

def update(o)

get user\_qualification\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:employee\_qualification)

post user\_qualification\_update\_path, {

:employee\_qualification => {

:level => o.level, :institute => o.institute,

:major => o.major, :year => 2010, :gpa => o.gpa,

:start\_date => '03-04-2006', :end\_date => '02-04-2010'

},

}

assert\_response :success

data = JSON.parse(@response.body)

assert\_equal 1, data['success']

end

end

end

E:\workspace\payroll\_rails\test\integration\user\_salary\_flows\_test.rb

require 'test\_helper'

class UserSalaryFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:ben)

end

test 'view salary' do

a = normal\_user(UserSalaryFlows)

a.logs\_in @user.username, 'secret'

a.view

a.logs\_out

end

private

module UserSalaryFlows

def view

get user\_salary\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:employee)

assert\_not\_nil assigns(:employee\_salary)

assert\_not\_nil assigns(:adjustment)

assert\_not\_nil assigns(:basic\_pay)

end

end

end

E:\workspace\payroll\_rails\test\integration\user\_total\_work\_hours\_chart\_flows\_test.rb

require 'test\_helper'

class UserTotalWorkHoursChartFlowsTest < ActionDispatch::IntegrationTest

setup do

@user = user(:ben)

end

test 'browse chart' do

a = normal\_user(UserTotalWorkHoursChartFlows)

a.logs\_in @user.username, 'secret'

a.browse

a.logs\_out

end

private

module UserTotalWorkHoursChartFlows

def browse

get user\_workhours\_chart\_path

assert\_response :success

assert\_template 'index'

assert\_not\_nil assigns(:month\_hash)

get user\_workhours\_chart\_data\_path

assert\_response :success

assert\_not\_nil assigns(:data)

end

end

end