Limiting Controller Access:

```ruby
class ApplicationController < ActionController::Base
  #Functions for the application controller.
  protect_from_forgery with: :exception
  #Include SessionHelpers. Allows helpers to be executed.
  include SessionsHelper

    private
    def logged_in_user
      unless logged_in?
        store_location
        flash[:danger] = "Please log in."
        redirect_to login_url
      end
    end

    def correct_user
      @user = User.find(params[:id])
      redirect_to(root_url) unless current_user?(@user)
    end

    def admin_user
      redirect_to(root_url) unless current_user.admin == true
    end

    def teacher_user
      redirect_to(root_url) unless current_user.teacher?
    end

    def student_user
      redirect_to(root_url) unless !current_user.teacher? && !current_user.admin?
    end
end
```

Figure 1. Definition of functions used to determine before actions

```ruby
class FreePlayRoomsController < ApplicationController

  before_action :logged_in_user
  before_action :student_user, :except => [:index]

```

Figure 2. Before actions for Free Play Rooms Controller

```ruby
class InterviewsController < ApplicationController

  before_action :logged_in_user
  before_action :admin_user
```

Figure 3. Before actions for Interviews Controller

```ruby
class SkillsController < ApplicationController

  before_action :logged_in_user
  before_action :teacher_user
```

Figure 4. Before actions for skills Controller

Limiting controller access is a fundamental aspect of the project. As defined in our users' stories: confidentiality and integrity of users' information are three major security assets that we strive to maintain. In order to insure the confidentiality of students, we want to limit who is able to view certain information. As seen in figure 2, logged in users are unable to view any aspect of the free play room. Unlogged in users are unable to view the times and dates of when students will be in the room. Additionally, teacher and admins are only able to view the index of the free play room. Thus, are unable to edit or delete student bookings – ensuring user integrity of their information. If an unlogged in user attempts to view the free play room controller, or in fact any controller except static pages and new user page, they will be redirected to the login page. Just like with the free play room, any logged in user who attempts to view information not relevant to themselves (unauthorised) will be redirected to the home page. This functions responsible for redirecting certain users are found in the application controller as seen in Figure 1, therefore all controllers in the project inherit these functions.

In addition to the control above, I have implemented other additional control methods to prevent information about users being disclosed to unauthorised users. An example of this is the show page of users. If the correct users is logged in they can see personal information such as email, parent name, etc. However, users are only able to see name, birth date and Facebook address of other users.

Test fixtures and Helpers:

```
# Test users currently used for website.
mika:
  id: 1
  name: Mika
  last_name: Williams
  email: mika@music.com
  facebook_ID: www.facebook.com/mikaMusic
  address: '450 Queens St, QLD'
  dob: 11/12/1980
  gender: Female
  password_digest: <%= User.digest('password') %>
  admin: true

teacher1:
  id: 2
  name: Teahcer
  last_name: num1
  email: teachernum1@example.gov
  facebook_ID: www.facebook.com/teachernum1
  address: '40283 Dunbar St, Grafton, QLD'
  dob: 20/08/1989
  gender: Male
  password_digest: <%= User.digest('password') %>
  teacher_qualifications: Graduated Queensland University of Technology
  user_recieve_emails: true
  teacher: true
  admin: false

teacher2:
  id: 3
  name: Teahcer
  last_name: num2
  email: techteacher@teacher.org
  facebook_ID: www.facebook.com/teacher1
  address: '100 Railway Ave, Tingalpa, QLD'
  dob: 05/08/1991
  gender: Female
  password_digest: <%= User.digest('password') %>
  teacher_qualifications: Graduated University Queensland
  user_recieve_emails: false
  teacher: true
  admin: false

student1:
  name: student
  last_name: number1
  email: student@example.gov
  facebook_ID: www.facebook.com/studentNum1
  address: '15 Collins St, Boondal, QLD'
  dob: 10/01/1995
  gender: Male
  password_digest: <%= User.digest('password') %>
  user_recieve_emails: false
  teacher: false
  admin: false
```

Figure 5. Test fixture for users

availability_test.r × | test_helper.rb | availabilities.yml ×

```
one:
  user_id: 2
  user_email: teachernum1@example.gov
  day: Monday
  time: 09:30
  duration: 2

two:
  user_id: 3
  user_email: techteacher@teacher.org
  day: Friday
  time: 09:30
  duration: 5
```

Figure 6. Test fixture for availabilities

availability_test.r × | test_helper.rb | av

```
one:
  user_id: 4
  user_email: student@example.gov
  preferred_day: Monday
  preferred_time: 12:30
  instrument: Guitar
  preferred_teacher_language: English
  preferred_teacher_gender: Male

two:
  user_id: 5
  user_email: num2@student.gov
  preferred_day: Friday
  preferred_time: 30:11
  instrument: Drums
  preferred_teacher_language: English
  preferred_teacher_gender: None
```

Figure 7. Test fixture for preferences

availability_test.r × | test_helper.rb

```
ENV['RAILS_ENV'] ||= 'test'
require File.expand_path('../../config/environment', __FILE__)
require 'rails/test_help'
require "minitest/reporters"
Minitest::Reporters.use!
class ActiveSupport::TestCase
  fixtures :all
  include ApplicationHelper

  def setup
    @admin = users(:mika)
    @teacher1 = users(:teacher1)
    @teacher2 = users(:teacher2)
    @student1 = users(:student1)
    @student2 = users(:student2)
    @availability1 = availabilities(:one)
    @availability2 = availabilities(:two)
    @free_play_room1 = free_play_rooms(:one)
    @free_play_room2 = free_play_rooms(:two)
    @interview1 = interviews(:one)
    @interview2 = interviews(:two)
    @preference1 = preferences(:one)
    @preference2 = preferences(:two)
    @skill1 = skills(:one)
    @skill2 = skills(:two)
    @base_title = 'Mika Music School'
  end

  # Returns true if a test user is logged in.
  def is_logged_in?
    !session[:user_id].nil?
  end
  # Log in as a particular user.
  def log_in_as(user)
    session[:user_id] = user.id
  end
end

class ActionDispatch::IntegrationTest
  # Log in as a particular user.
  def log_in_as(user, password: 'password', remember_me: '1')
    post login_path, params: { session: { email: user.email,
                                          password: password,
                                          remember_me: remember_me } }
  end
end
```

Figure 8. Test Helper

Above are some examples of test fixtures used for this project. As seen in Figure 5 compared to artefact 5, the user fixtures have been significantly modified and updated to reflect plans for release 2. The admin user has been named as Mika and her test credentials have been updated to reflect her. There are two fixtures for teachers, each with slightly different, yet both valid, credentials. This is the same two student fixtures.

Two test fixtures were created for each table/controller created. An example of this can be seen in Figure 6 and 7. Both fixtures for each table/controller are valid however, each fixture belongs to a separate individual. This is so it can be tested whether or not a user can manipulate an entry that does not belong to themselves. The test helper then assigns each of these fixtures to an object, thus allowing each test controller to access these objects without repetitive code.

Test controllers:

```ruby
require 'test_helper'

class InterviewsControllerTest < ActionDispatch::IntegrationTest

  test "admin should get index" do
    log_in_as(@admin)
    get my_interviews_path
    assert_response :success
    assert_select "title", "My Interviews | #{@base_title}"
  end
  test "admin should get new" do
    log_in_as(@admin)
    get new_interview_path
    assert_response :success
    assert_select "title", "Book Interview | #{@base_title}"
  end
  test "admin should get edit" do
    log_in_as(@admin)
    get edit_interview_path(@interview1)
    assert_response :success
    assert_select "title", "Edit Interview | #{@base_title}"
  end
  test "admin should get delete" do
    log_in_as(@admin)
    get delete_interview_path(@interview1)
    assert_response :success
    assert_select "title", "Delete Interview | #{@base_title}"
  end

  test "teacher should not get index" do
    log_in_as(@teacher1)
    get my_interviews_path
    assert_redirected_to root_url
  end
  test "teacher should not get new" do
    log_in_as(@teacher1)
    get new_interview_path
    assert_redirected_to root_url
  end
  test "teacher should not get edit" do
    log_in_as(@teacher1)
    get edit_interview_path(@interview1)
    assert_redirected_to root_url
  end
  test "teacher should not get delete" do
    log_in_as(@teacher1)
    get delete_interview_path(@interview1)
    assert_redirected_to root_url
  end

  test "student should not get index" do
    log_in_as(@student1)
    get my_interviews_path
    assert_redirected_to root_url
  end
```

Figure 9. Test Controller for Interviews part 1

```ruby
  test "student should not get index" do
    log_in_as(@student1)
    get my_interviews_path
    assert_redirected_to root_url
  end
  test "student should not get new" do
    log_in_as(@student1)
    get new_interview_path
    assert_redirected_to root_url
  end
  test "student should not get edit" do
    log_in_as(@student1)
    get edit_interview_path(@interview1)
    assert_redirected_to root_url
  end
  test "student should not get delete" do
    log_in_as(@student1)
    get delete_interview_path(@interview1)
    assert_redirected_to root_url
  end

  test "unlogged user should not get index" do
    get my_interviews_path
    assert_redirected_to login_url
  end
  test "unlogged user should not get new" do
    get new_interview_path
    assert_redirected_to login_url
  end
  test "unlogged user should not get edit" do
    get edit_interview_path(@interview1)
    assert_redirected_to login_url
  end
  test "unlogged user should not get delete" do
    get delete_interview_path(@interview1)
    assert_redirected_to login_url
  end
end
```

Figure 10. Test Controller for Interviews part 2

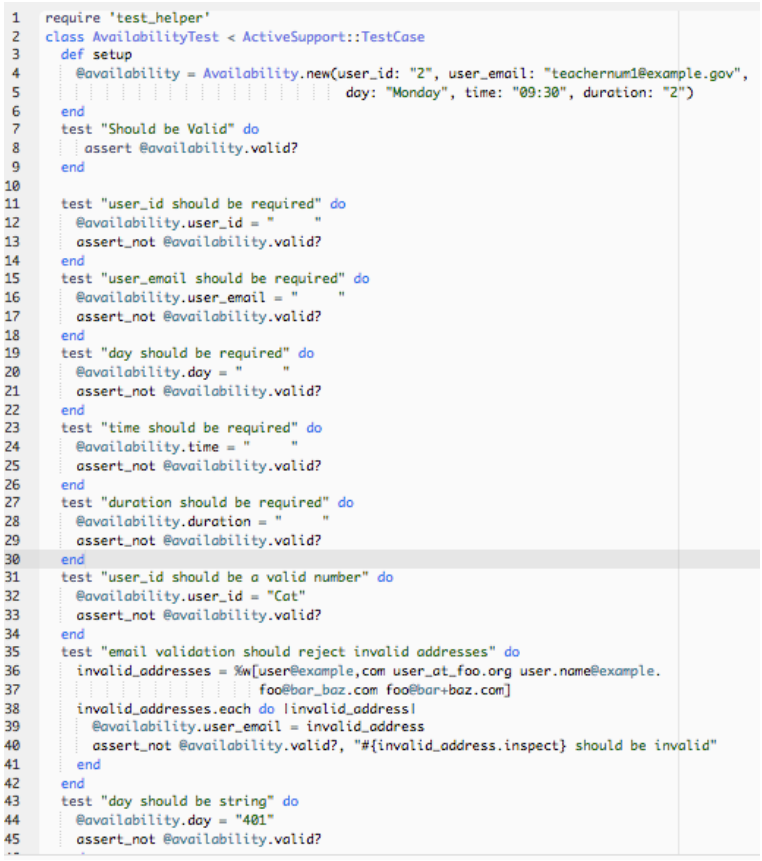
```ruby
require 'test_helper'

class AvailabilitiesControllerTest < ActionDispatch::IntegrationTest
  test "should get index" do
    log_in_as(@teacher1)
    get my_availabilities_path
    assert_response :success
    assert_select "title", "My Availabilities | #{@base_title}"
  end
  test "should get new" do
    log_in_as(@teacher1)
    get new_availability_path
    assert_response :success
    assert_select "title", "Add New Availability | #{@base_title}"
  end
  test "should get edit" do
    log_in_as(@teacher1)
    get edit_availability_path(@availability1)
    assert_response :success
    assert_select "title", "Edit Availability | #{@base_title}"
  end
  test "should get delete" do
    log_in_as(@teacher1)
    get delete_availability_path(@availability1)
    assert_response :success
    assert_select "title", "Delete Availability | #{@base_title}"
  end

  test "should redirect student to home page on index attempt" do
    log_in_as(@student1)
    get my_availabilities_path
    assert_redirected_to root_url
  end
  test "should redirect student to home page on new attempt" do
    log_in_as(@student1)
    get new_availability_path
    assert_redirected_to root_url
  end
  test "should redirect student to home page on edit attempt" do
    log_in_as(@student1)
    get edit_availability_path(@availability1)
    assert_redirected_to root_url
  end
```

Figure 11. Test Controller for Availabilities

As displayed in Figures 9 and 10, the test controller for Interviews has 16 tests and 20 assertions. The first four tests check to make sure that the admin is able to access all intended pages of the interview controller – on successful access then tests to make sure that the title is correctly displaying the appropriate information. There are then eight tests to test if both teachers and students are unable to access the interview controller and are redirected to the root url. Lastly are four tests, testing if un logged users are redirected to the sign up page.

This pattern is used almost identically for all test controllers, only changing the users that should be able to access that specific controller.

Model Tests:



```ruby
require 'test_helper'
class AvailabilityTest < ActiveSupport::TestCase
  def setup
    @availability = Availability.new(user_id: "2", user_email: "teachernum1@example.gov",
                                     day: "Monday", time: "09:30", duration: "2")
  end
  test "Should be Valid" do
    assert @availability.valid?
  end

  test "user_id should be required" do
    @availability.user_id = "     "
    assert_not @availability.valid?
  end
  test "user_email should be required" do
    @availability.user_email = "     "
    assert_not @availability.valid?
  end
  test "day should be required" do
    @availability.day = "     "
    assert_not @availability.valid?
  end
  test "time should be required" do
    @availability.time = "     "
    assert_not @availability.valid?
  end
  test "duration should be required" do
    @availability.duration = "     "
    assert_not @availability.valid?
  end
  test "user_id should be a valid number" do
    @availability.user_id = "Cat"
    assert_not @availability.valid?
  end
  test "email validation should reject invalid addresses" do
    invalid_addresses = %w[user@example,com user_at_foo.org user.name@example.
                           foo@bar_baz.com foo@bar+baz.com]
    invalid_addresses.each do |invalid_address|
      @availability.user_email = invalid_address
      assert_not @availability.valid?, "#{invalid_address.inspect} should be invalid"
    end
  end
  test "day should be string" do
    @availability.day = "401"
    assert_not @availability.valid?
```

Figure 12. Model Test for Availabilities

Model tests were added for all models generated by myself. Each test follows the same structure as shown above. First a valid entry is defined and then tested for validity. Then each field is left blank and tested if the updated entry is invalid. Lastly, specific fields are tested at the bottom of the model test file. This includes testing for format and data entry type.

Final Tests and Assertions:

```
bradz:~/workspace/IFB299/IFB299-Music-School (master) $ rails test
Running via Spring preloader in process 14286
Started with run options --seed 50785

  160/160: [============================================================] 100% Time: 00:00:04, Time: 00:00:04

Finished in 4.30766s
160 tests, 270 assertions, 0 failures, 0 errors, 0 skips
```

Figure 13. Final Test and Assertion Count

By the end of the addition of all tests mentioned above, there were a total of 160 tests and 270 assertions. This was last tested by myself on the 26/10/16.