CSCI 446 Web Applications: Midterm Exam

Trevor Whitney

March 10, 2012

1. PUT can modify in addition to create, whereas POST will merely create. Both tell the server to accept the entity enclosed within the request as a new subordinate of the request URI. Meaning, if the URI was '. . . `/artists`', the entity enclosed in the request would be considered an `artist`. However, if the entity already exists, PUT assumes that the entity enclosed in the request is an updated or modified version of the original entity, and will update/replace the existing entity on the server with the new one enclosed in the request.

2. The target URL is relative.

3. An absolute URL starts with a '`/`', and is appended to the root domain. So, in the above request, if the link was to '`/images/new`. . . ', this link would always go to '`domain.com/images/new`', where `domain.com` is the root domain of the site, no matter what the current URL is. However, as it is now, as a relative url, it will simply append '`images/new`. . . ' to the current URL. So, for example, from '`domain.com/artists`', it would go to '`domain.com/artists/images/new`. . . '.

4. This link would generate a POST request.

5. Yes, the query string is '`request_type=PUT`'.

6. The link declaration should include a title attribute, which would provide a description of what the link is for or where it goes.

7. The web browser serves as the portal between the user and the data, it allows the user to make requests for data and then displays the servers response. The database is where the data is stored, when a browser makes a request, the web server processes it, and fetches any requested data from the database to return to the user.

8. Yes, this HTTP response would usually contain a body, but the content of that body will be dependent on the request method (ie. GET, POST, etc.). 200 is the HTTP code for OK. In other words, the request has succeeded, so the web server is saying "Your request was OK, it succeeded, so here is what you requested."

9. My troll class would look like:

```
class Troll
 attr_accessor :ugliness, :smelliness, :strength

  def initialize(grunt = "UNGAH")
    @grunt = grunt
  end
```

```
  def speak
    42.times do
      puts @grunt
    end
  end

  def reverse
   puts @grunt.reverse
  end

  def self.propogate
   troll = self.new "eegah"
  end

end
```

10. For `fred.respond_to?("fight")` to return true, I need to inlcude a method called fight:

```
  def fight
  end
```

11. Yes, this is an illustration of object-oriented polymorphism. This method, '`respond_to?`' is available to all objects in ruby regardless of type. Therefore, you do not need to know the object type to be able to call this method. Similarly, the method acts in a predictable manor on all objects, regardless of type, by returning a list of methods available on that object. I believe that ruby implements this by putting the '`respond_to?`' method in the Object class, which all Objects in ruby inherit from. This sub-classing of objects is possible because of object-oriented polymorphism.

12. A boolean value.

13. The method with the bang will alter the objectm whereas the method without the bang will not. Or, in an example like ActiveRecord's `save` vs. `save!`, the version with the bang will raise an exception.

14. Ruby is dynamically and strongly typed. Ruby is dynamically typed because type checking is done at run time as opposed to compile time. This is true of most interpreted languages, like ruby, because an interpreted script does not need to be compiled before it can be run. This is in contrast to a language like C, where types must be specified in the code, because type checking is done at compile time. For example, in ruby, you can store the integer 5 in the variable `x` like so:

```
x = 5
```

Ruby dynamically figures out `x` is an integer based on what you assign to it. This would not work in C, and you must instead define `x` like so:

```
int x = 5;
```

You must specify "int" because C is statically typed, meaning the types of objects are static and must be defined at compile time. In this case, `x` is an integer, and can and only will be an integer unless explicitly cast as something else. In ruby, however, since types are dynamic, while `x` may start as an integer, you may later make it a string with an assignment like:

```
x = "5"
```

Furthermore, ruby is also strongly typed. Strongly typed means that, once an object's type is known, the language will expect you to do something sensible with it. For example, in ruby, once it knows `x` is a string with the value of `"5"`, it will not let you add an integer to it like so:

```
x = "5"
y = x + 7
```

Ruby will complain about this, specifically by raising the error "can't convert Fixnum into String". However, if you were to attempt to concatenate another string to `x`, ruby would be happy to do so.

```
x = "5"
y = x + " Monkeys"
```

This will result in the string `"5 Monkeys"`. Strongly typed languages are in contrast to weakly typed languages, which don't care if you mix and match object types. If you try to add a Fixnum to a String in JavaScript for example, it will not raise an error, but you will most likely get weird results.

15. This expression will yield the following array of strings:

```
[ "master", "rails", "and", "then", "try", "another", "framework", "
  you'll", "never", "go", "back" ]
```

16. Yes, they would both print each `happy_place` in the `@happy_places` array.

17. No, a return statement is not required. Ruby by default will return what is returned by the last command in the function. So, for example, if the last line of a function was `@trevors_grade = "A+"`, then the function would return the string `"A+"`. However, return statements are still available in ruby, and are sometimes helpful to improve the readability or understanding of your code.

18. The `before_validation` callback can be used to attach a method to be called before object validation occurs. The `before_save` callback can be used to call a method before an object is saved. With these `before_*` callbacks, if the associated method returns false, the operation, such as validation or saving, is cancelled, along with all later callbacks. The `after_validation` callback can be used to call a method after validation, and the `after_save` callback can be used to call a method after an object is saved. With the `after_*` callbacks, if the associated method returns false, all later callbacks are cancelled.

19.

HTTP method	**controller action**	**CRUD operation**
GET	index	read
GET	new	read
POST	create	create
GET	edit	read
PUT	update	update
DELETE	delte	delete

20. Not all webservers implement PUT and DELETE, thus rails "simulates" these methods to make it web server agnostic. Rails is attempting to revive HTTP in the way it was designed, to use more than just the GET and POST commands. Therefore, when the web server supports it, rails will accept a PUT and DELETE rquest to an update or delete url. However, rails "simulates" these methods so that they can also be called using a GET or POST request to the same URL when required.

21. The production enviornment is the actual, functioning application, as it exists in the real world. So, for example, I may have my site `reallycoolapp.com` hosted on a linux box at some hosting providers server farm. When people go to the url `reallycoolapp.com`, this is the version they see. However, I need to be able to make changes and updates to this site without distrubing the user experience by breaking something or bringing my site down. This is why rails provides the development enviornment, which usually resides on your local development machine. This enviornment is hosted on my local machine, by my local installation of apache, and the changes I make in the development enviornment are only sent to the production enviornment when I explicitly tell them to update.

22. A contoller should have a singular name, like in the example of `GeocodingController` when the thing it is "controlling" is a verb, like "geocoding". In this case, the controller likely is not directly related to a model, since models are usually things, and therefore nouns.

23. A helper method is a function that returns a block of markup for your view. Helper methods are used to keep your views clean, tidy, easy to read, and focused on the actual **markup**. They should be defined to refactor unruly code. For example, if you have a long `link_to` call in your view, the specifics of the link, such as title, image, etc. are not important to the markup of the page. For example, this link may be in a list of links. What is important to the markup, the structure of the document, is that there is a list of links. Therefore, you can move the long, unruly `link_to` call to an appropriately named helper method, which you can then call in your view. This cleans up the view by distracting less from the actual markup. When the view is rendered, your helper method will be called, inserting the necessary code.

24. The database schema must include a `bees_flowers` table that contains records with a bee and flower id, which are foreign keys to entities in the bees and flowers tables, respectively.

25. No, these queries are not ok since they are causing an unecessary decrease in the performance of our application. You could reduce the number of queries to the database by using a feature in rails called "eager loading". To implement this, you would change how the `FlowersController#show` action populates its bees. You could add a `@bees` collection to the `FlowerController` that utilizes eager loading of the hives:

```
@flower = Flower.find(params[:id])
@bees = Bee.find_all_by_flower_id(@flower.id, :include => :hive)
```

Then modify your view to something like the following:

```
- @bees.each do |b|
  %h1= b.name
  %p= b.hive.name
```

The `:include => :hive` parameter is what is responsible for the eager loading. Thus the `find_all_by` call will not only fetch the appropriate bees, but will also fetch their related hives, eliminating the plethora of unecessary SQL queries executed by the old view.