Week 4 Lecture 10

Applied

What’s in this lecture?

- Data Models (focusing on Ruby on Rails)
- REST (software architecture)

Problems:

- Logically organizing data is hard
- Structure conflicts with implementation
- Lines between objects are blurry
- Why?
    - Data overlaps, and is often dual use

So where do we start?

- Take a business object:
    - account, photo album, blog, post
- List all of its attributes
- Group by singular and multiple
- Group by hierarchy

Thinking about Data I

- What should be part of a User model?
  - email? (work? personal? other?)
  - phone number?
  - address? GPS location?
  - second address? Zip code?
  - *Hard*: usage statistics?

Thinking about Data II

- What types of data are each of these?
- Can you store GPS coordinates as integers?
- Can you store article text as a string?
- Do you need to search on zip code?

Thinking about Data III

- Creating a good data model is an art, rather than working from a rubric
- Goal is to insulate your data behind a logical interface

World’s Simplest Model

```
class Person
  attr_accessor :name

  def initialize(name)
    @name = name
  end

end
```

What does it do?

- Represents a single person
- Is initialized with a parameter 'name'
- 'name' can be **read** and **updated**
- **Is the mold from which we cast new people!**
- ...but that's about it

The code in action:

```
>> p1 = Person.new("Kip")
  => #<Person:0x1006096a8 @name="Kip">

>> p2 = Person.new("George")
  => #<Person:0x1005ff8b0 @name="George">

>> p2.name
  => "George"

>> p2.name = "Jerry"
  => "Jerry"

>> p2
  => #<Person:0x1005ff8b0 @name="Jerry">
```

What is going on?

- Person isn't actually data, its just structure!
- We're creating **one** instance of Person, assigned to **p1** with an object id of **0x1006096a8**
- We're creating a **second** instance of Person, assigned to **p2** with an object id of **0x1005ff8b0**
- **p2**'s name is updated from 'George' to 'Jerry'
- Tip: Google 'ruby object id' if you are confused

Let’s Extend It!

- Goal:

  People are social, and like to communicate. Jerry needs the ability to say ‘Hi’ to other people!

Poor Execution:

```
class Person
  attr_accessor :name

  def initialize(name)
    @name = name
  end

  def say_hi
    puts “hi kip!”
  end
end
```

Poor execution in action:

```
>> p1.say_hi
  => "hi kip!"

>> p2.say_hi
  => "hi kip!"
```

Better Execution

```
class Person
  attr_accessor :name

  def initialize(name)
    @name = name
  end

  def say_hi(first_name)
    puts "hi #{first_name}!"
  end
end
```

Better execution in action:

```
>> p1.say_hi("Jerry")
  => "hi Jerry!"

>> p2.say_hi("Kipling")
  => "hi Kipling!"
```

Adding ‘business logic’

- How the model behaves in relation to its **current state** and the state of **related objects**
- Create new user => business logic
  User wins contest => application logic

Let’s extend it further!

```
class Person
  attr_accessor :name

  def initialize(name)
    @name   = name
    @energy = 5
  end

  def say_hi(first_name)
    puts “hi #{first_name}!”
  end

  def start_dancin
    unless @energy < 4
      5.times {|i| puts “DANCE”[i] }
      @energy = @energy - 1
    else
      puts “ehh... more like time for bed...”
    end
  end

  def go_to_bed
    @energy = @energy + 5
  end
end
```

Hints

- Don’t store hard code data in your models
- Question isn’t *just* ‘What is our data?’ but ‘How can we represent our data?’
- Always consider what actions your model performs, and how they affect their data

RESTful Architecture

The Basics

- **RE**presentational **S**tate **T**ransfer
- An object can be represented by an ID
- State is the information needed to completely describe an object
- Transfer refers to the HTTP method used

REST'n'HTTP

- An object's state can be entirely controlled through the HTTP methods:

GET	--> index, show, new, edit
POST	--> create
PUT	--> update
DELETE	--> delete

Question:
What must a ‘Blog Post’ be able to do?

REST and Models

- Think of a model as a resource with a controller that can:
  - give all instances
  - display specific instance
  - create new instance
  - edit existing instance
  - delete existing instance

Hands On

- Run application ‘Graphr’ on your local machine:

```
$ bundle install
$ rake db:create
$ rake db:migrate
$ rails s
```

Create a new Point

```
curl -v -X POST --data "{\"graph_point\":{\"x_coord\":
\"2.05\",\"y_coord\":\"4.13\"}}" -H "Content-Type:
application/json" -H "Accept: application/json" "http://
localhost:3000/graph_points.json"
```

Get All Points

```
curl -v -X GET "http://localhost:3000/graph_points.json"
```

Update Point with ID

```
curl -v -X PUT --data "{\"graph_point\":{\"x_coord\":
\"122.23\",\"y_coord\":\"56.78\"}}" -H "Content-Type:
application/json" "http://localhost:3000/graph_points/1.json"
```

Show point by ID

```
curl -v -X GET "http://localhost:3000/graph_points/1.json"
```

Destroy Point

```
curl -v -v -X DELETE "http://localhost:3000/1.json"
```

Understanding it

- not RESTful:
  - PUT http://localhost:3000/start_party
- RESTFUL:
  - PUT http://localhost:3000/party/34

Putting it all together

- Models represent our data structures
- Instances of these models are our data
- A controller exposes a RESTful endpoint to access and control these models
- REST gives a pattern for data management

Exercises

- Use Rails to generate a scaffold for a BlogPost model. Examine the controller code generated.
- Implement a simple ROR blogging application: use a single Post model.