Rails in one lecture

How terribly impossible is that. Go read: http://guides.rubyonrails.org/
The Task

- Connect web clients to a database with Ruby!
Philosophy

- DRY - Don’t Repeat Yourself
- Convention over configuration - e.g., models are singular and CamelCased, tables are plural.
Model-View-Controller

- Model - provides an interface to the underlying data, independent of the action.
- View - presentation layer that formats the output
- Controller - bridges the two, contains the application logic
MVC, e.g.,

- class Student - decides how to access, defines relations such as courses or major
- class StudentController - defines actions such as enroll or change_major
- views/students/enroll.html.erb - formats a page for that action
<%= form_for(@student) do |f| %>
  <% if @student.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@student.errors.count, "error") %>
        prohibited this student from being saved:</h2>
      <ul>
        <% @student.errors.full_messages.each do |msg| %>
          <li><%= msg %></li>
        <% end %>
      </ul>
    </div>
  <% end %>
</div>
  <% end %>

<div class="field">
  <%= f.label :umid %><br />
  <%= f.text_field :umid %>
</div>
Routes

• How an HTTP request is converted into a controller action, formatted by a view.

• Explicit (for one)
  • match “posts/:id”, :to => posts#show

• Resource (for CRUD)
  • resources :students
Routes

- Can also be inverted, so that a view can generate links to actions, e.g.,
  - student_path(@stu)
  - new_student_path
- Can nest resources, e.g., comments within posts, questions within exercises.
• select * from students where major = “cs”

• In practice:
  • Queries join several tables, can become incomprehensible.
  • Search and sort criteria come from user input, inherently untrusted.
SQL injection

Hi, this is your son's school. We're having some computer trouble.

Oh, dear - did he break something? In a way-

Did you really name your son Robert'); DROP TABLE Students;--?

Oh, yes. Little bobby tables, we call him.

Well, we've lost this year's student records. I hope you're happy.

And I hope you've learned to sanitize your database inputs.
ActiveRecord (Models)

• Validations - Ensure that only “good” data is written to the database, e.g., that a field must be non-null.

• Associations - Relate one object to another via methods, e.g., a student “has_many :courses” provides a student.courses method

• Callbacks - Trigger events, such as to log changes.
ActiveRecord Queries

- @majors = Student.where(:major => "cs").includes(:courses)
- @majors = Student.includes(:courses).where(major:"cs")
ActiveRecord Scopes

class Student < ActiveRecord::Base
  scope :current, where(:graduated => false, :enrolled => true)
end

class StudentsController < ApplicationController
  def index
    @majors = Student.current.where(:major => "cs")
    my_scope = Student.current
    @majors = my_scope.where(:major => "cs")
  end
end
Migrations

• Define a database-independent language for creating or modifying the database schema.

class AddNewFlagToStudents < ActiveRecord::Migration
def change
  add_column :students, :is_new, :boolean
end
end
Associations in the DB

- “Foreign Key” - Each record has a numeric id; one record can reference by its id.

- In a blog example, a comment table might have columns: id, post_id, user_id, text, and timestamp.

- post_id and user_id are “foreign keys”. The id column is the “primary key”.

- Rails expects this convention, but you can specify the key.
Associations in Rails

- `has_many, has_one` - A blog Post has_many Comments.
- `belongs_to` - A Comment belongs to a Post and belongs to a User.
- `has_many :through` - A Post has_many Users through Comments.
- `has_andbelongs_to_many "habtm"` - Same as `has_many :through`, only the “join table” has no model.
Generators

• bundle exec rails generate model student
  major:string name:string

• bundle exec rails generate controller
  student enroll change_major

• bundle exec rails generate scaffold post
  title:string body:text published:boolean
bundle exec?

• The bundler executes code within an environment of specific versions of plugins that your rails app uses.

• These plugins are listed in Gemfile.

• Think of it as per-application apt-get: if you want to use a new plugin, add it to the Gemfile, run bundle update.
Like irb?

- bundle exec rails console
- Create objects, set the admin flag on user records, destroy records that were mistakes, debug actions, etc.
Like testing?

- You should... since it is ridiculously easy to leave a syntax error in Rails code.
- Generators create simple unit tests and functional tests, as well as “fixtures” that seed a test database.
Environments

• Rails defaults to providing three, each has configuration settings, each has a DB:
  • Production - all caching on, debugging off.
  • Development - show stack traces on exceptions, log sql queries.
  • Test - skip some checks
Do.

- http://rubyonrails.org/download
- railscasts.com - teaches me.
- railsforzombies.org
- Agile Web Development with Rails (4ed)