INFO 2310

Topics in Web Programming
Ruby on Rails

Week 1

Introduction &&
Tools &&
Development Environment

Course Caveats

- This is my first teaching experience
- I have professional Rails experience, but I would hardly consider myself an expert

So why are you here instead of learning online on your own?

- Hopefully our course is at least as good as the materials
- You have a classroom full of people, myself, and Professor Williamson to learn from
- A set meeting time every week keeps you on track
- You get credit!

Alternatives

* Ample online tutorials
    * specifically, ruby.railstutorial.org
* See also
    * curails.herokuapp.com/resources

Pre-requisites

INFO-2300 or equivalent

You should have experience building a database backed web application, writing SQL queries, and writing HTML/CSS.

Some comfort with object oriented programming will help as well.

If you are not sure, feel free to chat with me after class.

Course Structure

* 1 credit S/U
* 10 weeks, 1 meeting a week
* To pass, attend 8/10 classes and have a working website that we will build together in class.

Tentative Syllabus

- Week 1 : Introduction, Tools and Development Environment
- Week 2 : Static Pages && Rendering / Ruby
- Week 3 : Users && ActiveRecord
- Week 4 : Signing up Users
- Week 5 : Signing in / Signing out
- Week 6 : Microposts
- Week 7 : More User Pages
- Week 8 : Following Users
- Week 9 : Stress Testing && Tools
- Week 10 : TBD

Course Materials

* No required textbook
* The course is based heavily off of ruby.railstutorial.org, which is a Rails tutorial available for free online.
* Course website is curails.herokuapp.com

Instructors

* Matt Goggin (mg343@cornell.edu)
    * OH - Weds 2:40 - 4pm (after class)
        - Monday 7-8pm
        - by appointment
* Professor Williamson (dpw@cs.cornell.edu)

What is Ruby on Rails?

Ruby is a programming language (like PHP, Java, Python).

Rails is a web framework, written in Ruby.

Essentially, it's a big piece of software designed to help us make websites.

Why use Rails?

It facilitates tasks common to developing web sites. E.g.

* Interacting with a database
* Accepting and validating user input (forms)
* Security
* Routing
* Testing

Philosophies

Agile Software

The old way...
* gather requirements from client
* build and test everything
* show the client
* client requests changes
* rinse and repeat

Agile Manifesto

http://agilemanifesto.org

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Agile Software in Rails

No tool is in and of itself agile

But Rails gives us the ability to very quickly get a simple system working.

This facilitates making progressive changes, getting feedback, and iterating on that feedback

Philosophies

Convention over Configuration

In Rails, sensible defaults for most things are provided; this allows you to accomplish simple tasks very quickly, only overriding the defaults when you need to do something more complex or unusual.

* If you have a Post table, and a Post has many Comments, you can tell Rails about this association with

has_many :comments

* Rails will infer table and column names, but you can override them if yours are different

Philosophies

Don't Repeat Yourself (DRY)

Define something, ANYTHING, in one and only one place. If you are storing the same information in more than one place, you're doing it wrong.

This applies to any part of your app, including both data and behavior.

When data or behavior changes it only needs to happen in ONE place

DRY Examples

Your database connection information is stored in a single file, not scattered among scripts that connect to a database.

Defining pieces of your layout that are replicated across many different views, e.g. a header or sidebar, in a single file and including them in other views

Philosophies

Model-View-Controller (MVC)

Model-View-Controller is a (now) standard architecture in software engineering. It separates the data and operations on the data from code that actually displays it.

The central idea behind MVC is code reusability and separation of concerns.

Separation means that we can make changes to one part without affecting what happens to the other.

Not MVC

```
<form method="GET" action="thispage.php"> <table>
  <?php
    if (isset($_GET['sort'])) {
      $query = "SELECT * FROM Movies ORDER BY ".$_GET['sort'];
    } else {
      $query = "SELECT * FROM Movies";
    }

    $results = mysql_query($query,$link);

    while ($row = $mysql_fetch_array($results, MYSQL_ASSOC)) {
      print("<tr>");
      foreach ($row as $item) {
        print ("<td>$item</td>");
      }
      print("</tr>");
    }
  ?>

  <input type="submit" name="sort" value="title" />
```

Model

The objects which store and manipulate the data for an applications. They encapsulate all of the business logic of an MVC application.

E.g. a User object, or a Post object

Controller

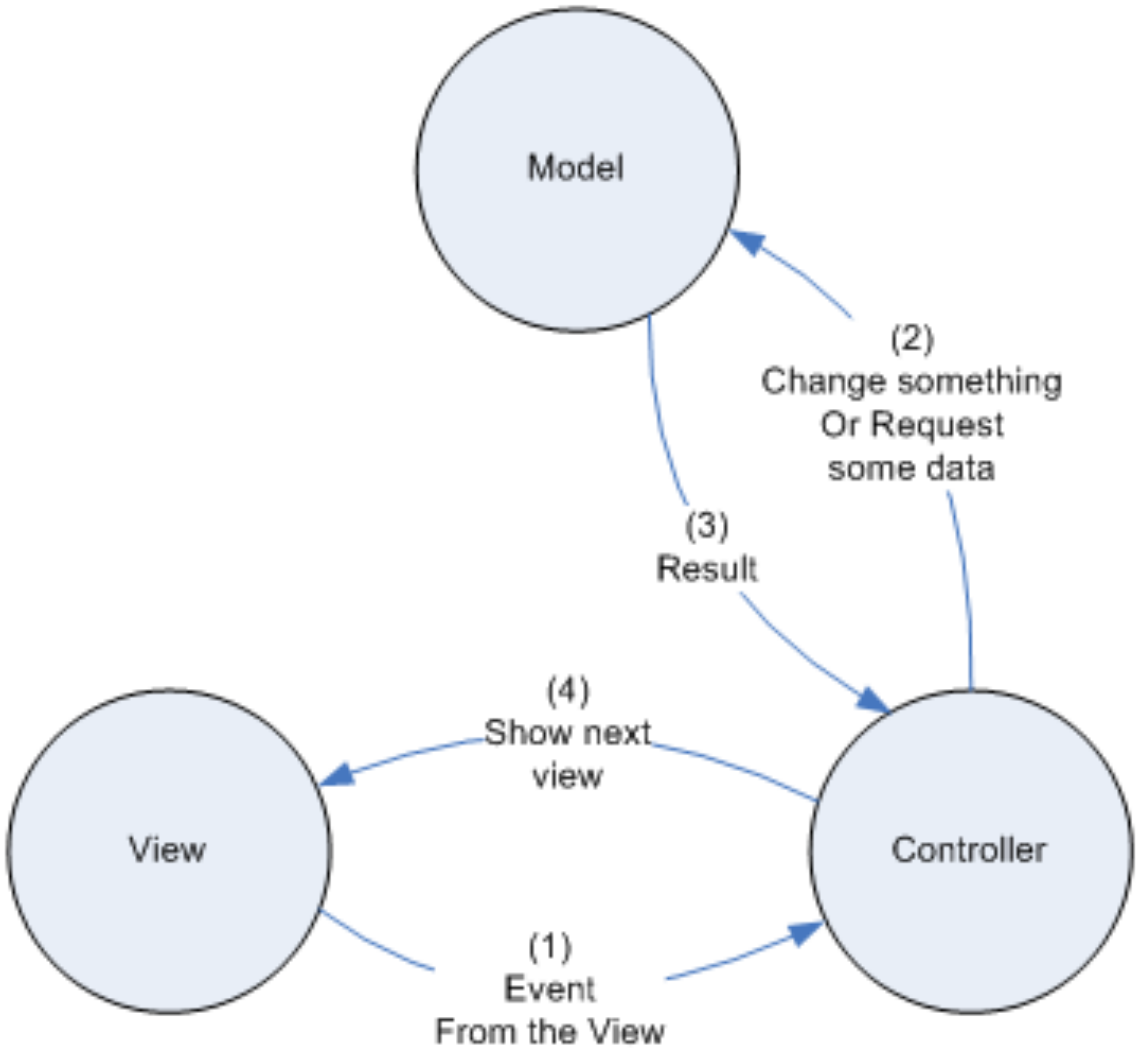
Accepts incoming requests (generated by a user), summons and potentially manipulates the relevant models, and then passes them onto the view for rendering.

View

The presentation layer, which accepts input from the user passes it back to the controller, and displays the result again to the user.

In a web application the views will be primarily HTML (and some embedded Ruby), but they could also be JSON, or XML responses.

Model-View-Controller



Development Environment

Amazon EC2 (elastic compute cloud)

* on demand virtual machines from Amazon Web Services

You should have already set up the EC2 image we provided, which is pre-installed with

* Ruby 1.9.3
* Rails 3.2.11
* Git
* Heroku

We will develop "in the cloud" using tools on your local machine to connect to and manipulate an EC2 virtual machine.

* WinSCP allows us to easily edit files on the EC2 instance
* PuTTY gives us an SSH connection to the EC2 instance so that we can issue terminal commands (e.g. starting and stopping our Rails servers).

Setting up our Development Environment

Log into the Amazon Web Services (AWS) console.

https://console.aws.amazon.com/console/home

Start your instance

Setup

Open up your email and download the *.pem file that you created when you first set up your EC2 instance.

We need to convert the type of key generated by Amazon (*.pem) into the type of key required by PuTTY and WinSCP.

* Open PuTTYgen
* Click "Load"
* Choose your *.pem file.
* Click "Save Private Key", and name it with a *.ppk extension
* Close PuTTYgen

Save your converted *.ppk key on your flash drive, or email it to yourself so you don't need to do this every class.

Logging in via PuTTY

- open PuTTY
- On the left panel, navigate to Connection->SSH->Auth
  - Next to "Private key file for authentication:", click "Browse", and select the *.ppk file you created on the previous step.
- Then, navigate to Connection->Data
  - For "Auto-login username", type "ec2-user"
- Navigate to "Session" (the very top)
  - Copy the "Public Domain Name" of your EC2 instance; you can see this on the instances page of the Amazon console, when an instance is selected
  - Paste it into the "Host Name (or IP address)" field
- Click "Open"

Setup

- Open WinSCP
  - Paste in your domain to "Host name", as you did in PuTTY
  - Type "ec2-user" for the "User name"
  - Click "..." to select your private key file
  - Click "Login"
- Set NotePad++ as the default editor.
  - Click Options->Preferences
  - Select "Editors" from the left tab
  - Click Add
  - Select "External Editor"
  - Find NotePad++ (C:\Program Files (x86)\Notepad++)
  - Click "Open", then "Okay"
  - Drag it to the top of the editor list

And finally....

```
cd ~/apps                 # change directories into apps

# create a brand new rails app
rails new info2310 --skip-test-unit

cd info2310               # change directories into your app
rails server              # start the Rails webserver (aliased to rails s)
```

Visit
your_ec2_domain_name.com:3000
to see your first Rails app

Open up WinSCP to see all the files generated.

The file being shown in the browser is at public/index.html

Go ahead and edit it, then refresh your browser to see the changes.

Git && Github

Git is a distributed version control system (like SVN).

Version control systems allow us to track changes to our source code, and facilitate working with other developers on the same code base.

Github offers free Git hosting for public repositories (and some free private ones if you're a student).

Set up your credentials for Git

In your PuTTY terminal, set the global git credentials to point to your name, and the email you used to set up your Github account

First you'll have to press ctrl-c to stop your rails server

git config --global user.name "Your Name Here"
git config --global user.email "your_email@youremail.com"

Now when you push to Github your commits will be attributed to your Github user.

Go ahead and log into Github, and create a new repository for our rails app.

Push your code to Github

```
git init                                    # initializes an empty git repository
git add -A                                  # add all of the application files to the git repo
git commit -m "initial commit"    # commit all the files that you just added

# tell your local git where the central git repository is
# be sure to copy the HTTP url and NOT the SSH URL
git remote add origin <your URL from github>

# push the code on your current branch to the remote server
git push origin master

# Sleep better now that our code is backed up and version controlled
```

Heroku

Heroku is a service for hosting Rails (among others) applications. They offer plug and play functionality for many common web application needs (databases, caching, email server, DNS setup...).

There offer a free tier for applications that do not require many resources.

The "heroku" command allows you to interact with the heroku service. It is already installed on your EC2 instance.

Before we deploy...

a few changes to our Gemfile

replace the line
gem 'sqlite3'

with

```
group :development, :test do
  gem 'sqlite3'
  gem 'rspec-rails'
  gem 'capybara'
end

group :production do
  gem 'pg'
end
```

What did we just do?

The Gemfile contains the list of "gems" our Rails app depends on. Gems are third party ruby libraries packaged up for inclusion in other projects.

The Gemfile has groups for which "environment" needs which gems. Some gems are only needed when we are developing, or testing. Some are needed everywhere.

Heroku use PostgresSQL as it's relational database. Installing postgres can be a pain, we will use SQLite to develop locally, and Postgres when we are in production on heroku

Install our new gems

Now that we've updated the Gemfile, we need to install our new Gems.

Running this command

bundle install --without production

will install all the gems except the production group, since we don't want to deal with installing Postgres.

We should update Git with our changes.

git add -A

git commit -m "add testing gems to Gemfile, and add pg for production database"

git push origin master

Now we can deploy to Heroku

in your putty terminal...

heroku login # you will be prompted for your heroku credentials
heroku keys:add # create a new SSH key and upload it to Heroku

create a new heroku app: use the name curails-NETID
heroku create curails-mg343

You deploy to heroku by using Git to push to a different remote server than before. Remember we pushed to Github with the command "git push origin master".

git push heroku master # deploy your code to heroku

Once the push completes, you can visit your app at curails-NETID.herokuapp.com

Stop your instances

Remember to stop (but NOT terminate) your EC2 instance by right clicking on it, and choosing "stop".

You will be able to reboot next time from where we left off, and you won't be charged while the instance is not running.

Today we...

Discussed some of the philosophies behind Rails:

* Agile
* Convention over Configuration
* MVC
* DRY

Set up our development environment

* An EC2 virtual machine
    * Ruby, Rails, Git, Heroku
* WinSCP and PuTTY on the lab machines to connect to it

Made the initial commit to our version control system on Github

Deployed our first Rails app to Heroku :)

See you next week!

My first office hours are right now, so come chat with me if you have any questions