Build and deploy a web application

What will it do?

* Add topics

* Vote on a topic

Let’s Build a Web Application

```
rails suggestorama -m http://gist.github.com/194076.txt
```

File/Folder	Purpose
README	This is a brief instruction manual for your application.
Rakefile	This file contains batch jobs that can be run from the terminal.
app/	Contains the controllers, models, and views for your application. You will do most of your work here.
config/	Configure your application’s runtime rules, routes, database, and more.
db/	Shows your current database schema, as well as the database migrations.
doc/	You would add documentation for your application here
features/	Added by the template. This is from cucumber, not part of core Rails.
lib/	Extended modules for your application (not covered today).
log/	Application log files.
public/	The only folder seen to the world as-is. This is where your images, javascript, stylesheets (CSS), and other static files go.
script/	Scripts provided by Rails to do recurring tasks. We'll use some today.
test/	Unit tests, fixtures, and other test apparatus.
tmp/	Temporary files
vendor/	A place for third-party code.

Run the web app:

```
ruby script/server
```

Point your web browser at http://localhost:3000

Make it your own

Modify public/index.html

Run it again

Awesome! Let's ship it!

Create a local git repository:

```
git init
git add .
git commit -m 'basic web application'
```

Then deploy to heroku:

```
heroku create
git push heroku master
```

A Closer Look at the Features

The features have been defined in the /features directory. These were written for you in advance as a specification of the application.

One of the basic features that we might build first is defined in the topics.feature file in the features directory:

```
Feature: Topics
  In order to see a list of potential topics for meetings
  people need to be able to create and edit them

  Scenario: Getting to the new topic page
    When I go to topics
    And I follow "New topic"
    Then I should see a "Create" button
```

You can run all of the features in that file with:

```
cucumber features/topics.feature
```

Of course, the feature fails because we haven't written any code yet! These feature descriptions are tests as well as documentation. Typically we run the test, watch it fail, then implement a feature, then run the test again to see if it passes. We'll be doing that today.

The topics feature relies on some basic elements of a web app, which is what we'll build first. As we get features to pass, we'll look further at the features definitions to see what needs to be built

To run a single feature scenario, you can provide its name:

```
cucumber features/topics.feature -n 'Getting to the new topic page'
```

Adding topics

We can look through the features and screen shots and see how a topic is defined and how people expect to interact with it. We will use rails "scaffolding" to generate some pages (and code).

* topic will have a title and a description
* we will enable basic "CRUD" actions
    - Create - enter a new topic
    - Read - see everyone's topics
    - Update - change the topics you entered
    - Delete - remove your topics from the system

```
ruby script/generate scaffold topic title:string description:text
```

Holy generated files, batman!

Open the migration file (yours will have a different number):

db/migrate/20091014021209_create_topics.rb

See how it contains Ruby code to set up the database table. Migrations can also be used for modifying tables (add/remove/rename columns) and even modifying data

Now let's set up the database for the test by running the migration file on the test database:

```
rake db:migrate RAILS_ENV=test
```

Run the cucumber feature again (and the first one should pass)

```
cucumber features/topics.feature
```

Now let's look at the feature we created. We will typically do so in the development environment:

```
rake db:migrate
```

then start your server:

```
ruby script/server
```

and point your browser at: http://localhost:3000/topics

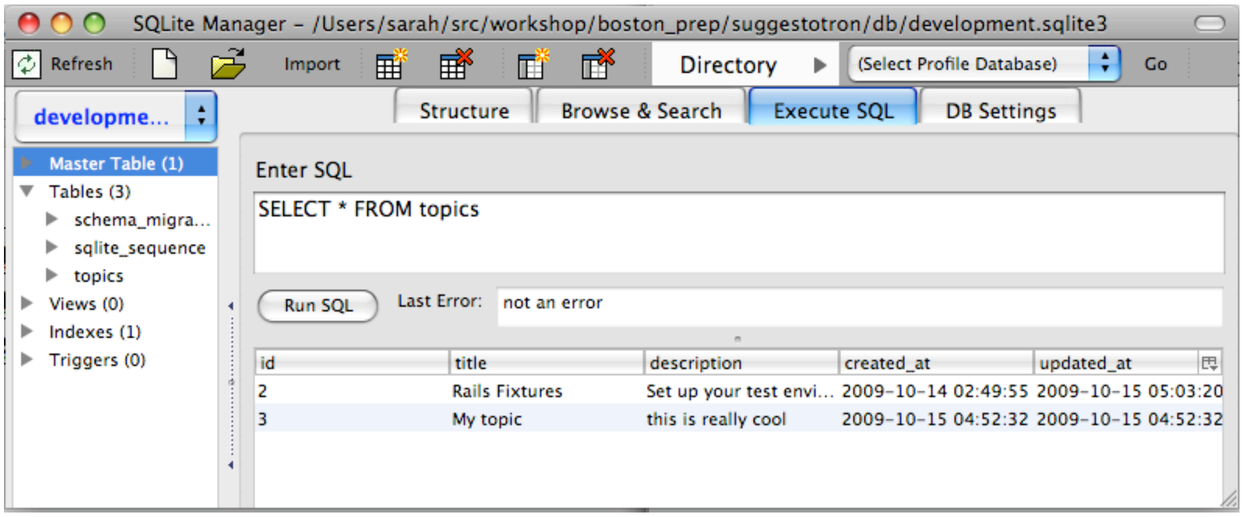
Congratulations! You have built a web application that works with a relational database.

A Quick Look at Your Database

You can access the database directly on your local machine. For this class we're using SQLite and you have installed a GUI tool that let's you inspect the database. In Firefox select:

Tools -> SQLiteManager

Then click the open folder icon or from the menu choose Database -> Connect Database and open suggestorama/db/ development.sqlite3



```
SELECT * FROM topics
```

Last Error: not an error

id	title	description	created_at	updated_at
2	Rails Fixtures	Set up your test envi...	2009-10-14 02:49:55	2009-10-15 05:03:20
3	My topic	this is really cool	2009-10-15 04:52:32	2009-10-15 04:52:32

You can choose "Browse & Search" to interactively explore the database or "Execute SQL" to type SQL commands.

You can also access the database through the command line:

```
ruby script/dbconsole
```

Common SQL Commands

	sqlite	MySql
list tables in current db	**.tables**	show tables;
show SQL for table create list columns	**.schema** **.schema people**	show create table topics; describe topics;
exit command line tool	**.quit**	exit
show all rows in table	**select * from topics;**	

show number of rows	**select count(*) from topics;**
show matching record	**select * from topic where title = "My Topic";**

Deploy Your Application

Don't forget commit early, deploy often:

```
git add .
git commit -m 'topic crud'
git push heroku master
heroku rake db:migrate
```

Congratulations! You have built and deployed a web application that works with a relational database.

What did we just do?

Rails implements a very specific notion of the Model-View-Controller pattern which guides how you build a web application.

Model

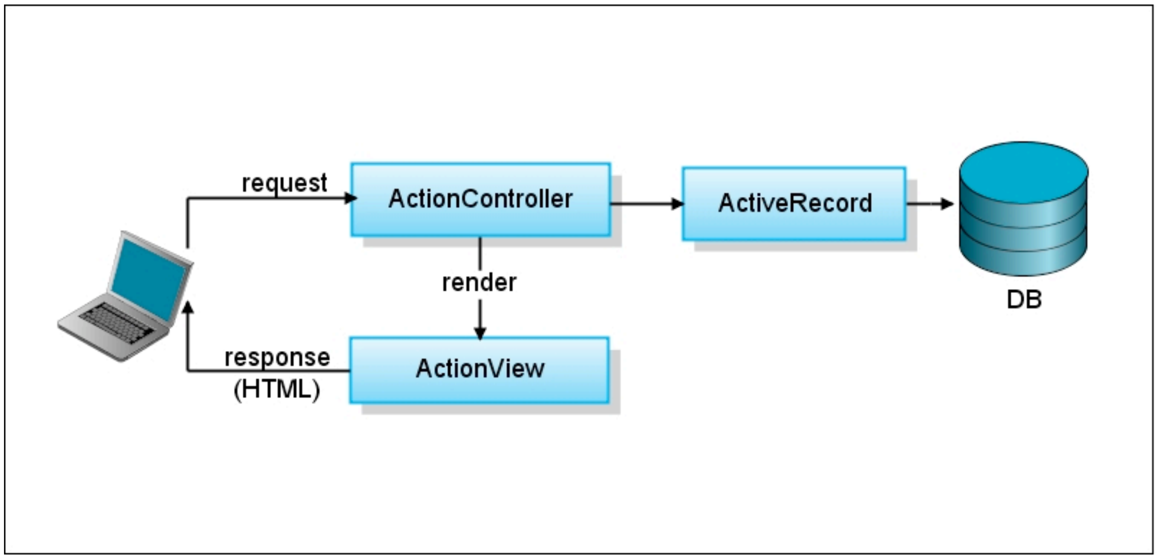
- o represents what is in the database
- o ActiveRecord

View

- o the model rendered as HTML
- o ActionView, erb

Controller

- o receives HTTP actions (GET, POST, PUT, DELETE)
- o decides what to do, typically rendering a view
- o ActionController



When you executed the script/generate command Rails generated files that implement a model, views and a controller for the topics featre.

The model

- o create app/models/topic.rb
- o create db/migrate/20090611073227createtopics.rb

4 views

- o create app/views/topics/index.html.erb
- o create app/views/topics/show.html.erb
- o create app/views/topics/new.html.erb
- o create app/views/topics/edit.html.erb

The controller

- o create app/controllers/topics_controller.rb
- o route map.resources :topics

A closer look

Rails allows you to easily invoke irb with all of the Rails libraries and your application code loaded:

```
ruby script/console
```

Let's look at the model that is defined here: app/models/topic.rb

```
>> t = Topic.new
=> #<Topic id: nil, title: nil, description: nil, created_at: nil, updated_at: nil>
>> t.title = "My topic"
=> "My topic"
>> t.description = "this is really cool"
=> "this is really cool"
>> t.save
```

Notice that the Topic class has title and description attributes which you did not need to explicitly declare in the class. This is handled by ActiveRecord which implements ORM (Object Relational Mapping) in Rails.

Controller

Rails routes control how URLs map to code

```
$ rake routes
    topics GET    /topics(.:format)                  {:action=>"index", :controller=>"topics"}
           POST   /topics(.:format)                  {:action=>"create", :controller=>"topics"}
new_topic GET     /topics/new(.:format)              {:action=>"new", :controller=>"topics"}
edit_topic GET    /topics/:id/edit(.:format)         {:action=>"edit", :controller=>"topics"}
    topic GET     /topics/:id(.:format)              {:action=>"show", :controller=>"topics"}
          PUT     /topics/:id(.:format)              {:action=>"update", :controller=>"topics"}
          DELETE  /topics/:id(.:format)              {:action=>"destroy", :controller=>"topics"}
                  /:controller/:action/:id
                  /:controller/:action/:id(.:format)
```

Each method in the controller will take an HTTP request, usually find some data in the database (via an ActiveRecord model) and render a view or re-direct to another action.

Next feature

Let's run the next feature and see what we need to build next

```
Scenario: Creating a topic
  Given I go to topics
  And I follow "New topic"
  When I fill in "Title" with "Rails Fixtures"
  And I fill in "Description" with "Introduce how to add test data with fixtures."
  And I press "Create"
  Then I should see "Rails Fixtures"
  And I should see a "New topic" link
```

Run the server, look at the app and see how the scaffold template differs from the desired application as we've described it using cucumber. In the desired behavior, after someone creates a topic, the expected behavior is for the application to display the list of topics; however, scaffold instead displays a page showing the individual topic that we just created

Controller: adjusting the flow of your application

Earlier we looked at the scaffold generated code a bit. To change the behavior and make the feature act as desired, we will look closely at the controller, which controls the general flow of your application (which page is displayed when someone clicks a link or a button)

In app/controllers/topics_controller.rb

Look at new and create actions

Notice that in create there is a redirect to

```
format.html { redirect_to(@topic) }
```

Instead we want to re-direct to the list of topics (look up the correct path using rake routes)

```
format.html { redirect_to(topics_path) }
```

Next feature: topics page

Note for this next set of feature scenarios we have a "Background" set of steps which will be executed before each scenario in the file. These rely on behavior that already works, so are green (pass).

```
 Scenario: Clicking on the topic title
  When I follow "Rails Fixtures"
  Then I should see "Introduce how to add test data with fixtures."
  And I should not see "add a topic"
```

Run the scenario and watch it fail:

```
cucumber features/topics_list_and_details.feature
```

in views/topics/index.html.erb

```
<td><%= link_to h(topic.title), topic %></td>
```

Change "Destroy" to "Delete" in views/topics/index.html.erb

Allow voting on a topic

```
Feature: Votes
  In order to determine which talk to give
  people need to be able to vote for the ones they like

  Background: Make sure that we have a topic
    Given I go to topics
    And I follow "New topic"
    When I fill in "Title" with "Rails Fixtures"
    And I fill in "Description" with "Introduce how to add test data with fixtures."
    And I press "Create"

  Scenario: viewing votes already cast
    When I go to topics
    Then I should see "0 votes"

  Scenario: voting on a topic
    When I follow "+1"
    Then I should see "1 vote"
```

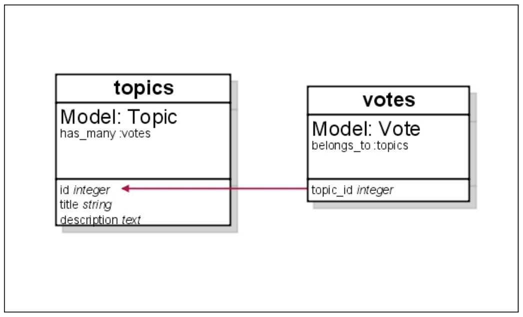
```
cucumber features/votes.feature
```

How will we build this feature?

* Each vote will be an object (row in database table)
* When someone votes on a topic, we'll create a new vote object and save it
* Each vote is associated with a specific topic

Rails associations

* Topic has_many :votes
* Vote belongs_to :topics



Add votes

We will use the resource generation script to create model & controller (no views)

```
script/generate resource vote topic_id:integer
```

the script creates files with:

- o model (including migration, unit, fixture) o controller (and route) with no code

Add code to to your models create the associations:

```
class Topic < ActiveRecord::Base
  has_many :votes
end

class Vote < ActiveRecord::Base
  belongs_to :topic
end
```

Check it out in irb (usually we would be test driving this, but since you are writing this code for the first time, we'll just explore)

```
>> t = Topic.new
=> #<Topic id: nil, title: nil, description: nil, created_at: nil, updated_at: nil>
>> t.votes
=> []
```

Now you can use it in your view (along with a handy view helper)

```
<td><%= pluralize(topic.votes.length, "vote") %></td>
```

Allow people to Vote

This is good to do one bit at a time and let the test failures drive what you do next. Check rake routes for figuring out the path.

```
<td><%= link_to '+1', votes_path(:topic => topic.id), :method => :post%></td>
```

Next we need to create the controller action

```
class VotesController < ApplicationController
   def create
     vote = Vote.new(:topic_id => params[:topic])
     vote.save

     if vote.save
       flash[:notice] = 'Vote was successfully created.'
     else
       flash[:notice] = 'Sorry we could not count your vote.'
     end
     redirect_to(topics_path)
   end
 end
```