CS 416
Web Programming

Ruby on RAILS
Chapter 2

Dr. Williams
Central Connecticut State University

Getting started – Blog app

- Create new app:

```
rails new toy_app
```

- Copy Gemfile from Listing 2.1 from book

- Run bundle

```
bundle install --without production
```

- Create git repo

```
echo "# toy app" >> README.md
git init
git add .
git commit -m "Initialize repository"
git remote add origin yourRepoURL
git push -u origin master
```

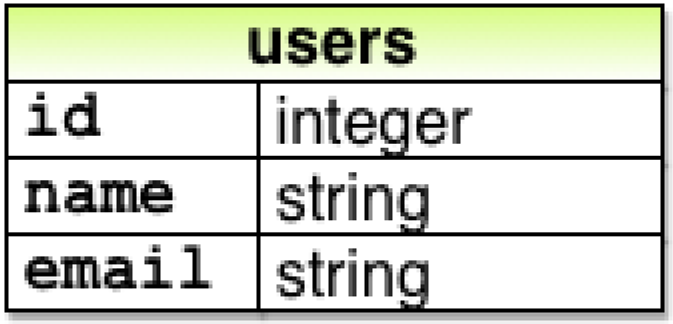
Push to heroku

- Create new app at Heroku
  - `heroku create`
- Push from your git repository to Heroku
  - `git push heroku master`
- Done!

(Note in general if a push to heroku fails, the first thing to check is make sure everything has been committed to git and pushed to origin)

Creating models in Rails

- Just like we saw in Java, think about fields needed and data types



users	
id	integer
name	string
email	string

- In rails by defining attributes and types it will generate both the object model, data model, and tying one to the other

Id field is meant as primary key

Create model using scaffolding

- In rails the idea of scaffolding is to create a model and create basic interfaces for create, read, update, and delete, that can then be overridden – Creates them as REST interface
- Creating the scaffolding:

```
rails generate scaffold User
name:string email:string
```

Specify business object name (capitalized), then each attribute name along with its type

Note the primary key will be created automatically

Rails DB migrations

- A central concept behind Rails is that in addition to versioning business objects as they change through out the project, since tied directly to DB also must version DB
- Each time you modify object, if data model must also change new DB migration is created (how to modify DB to meet new object model)
- To apply migrations:

```
rails db:migrate
```

Scaffolding tour

URL	Action	Purpose
/users	index	page to list all users
/users/1	show	page to show user with id 1
/users/new	new	page to make a new user
/users/1/edit	edit	page to edit user with id 1

RESTful routes by Users resources

HTTP request	URL	Action	Purpose
GET	/users	index	page to list all users
GET	/users/1	show	page to show user with id 1
GET	/users/new	new	page to make a new user
POST	/users	create	create a new user
GET	/users/1/edit	edit	page to edit user with id 1
PATCH	/users/1	update	update user with id 1
DELETE	/users/1	destroy	delete user with id 1

REpresentational State Transfer (REST)

- Application components modeled as *resources*
  - **C**reated, **R**ead, **U**pdated, and **D**eleted (CRUD)
  - HTTP requests: `POST`, `GET`, `PATCH`, and `DELETE`

- Idea is for all components in application to be thought of in this way to guide choices of what type of calls and controllers to provide

Rails routes

- *To list current routes in rails run:*

```
rails routes
```

- Modify root route to point to users index

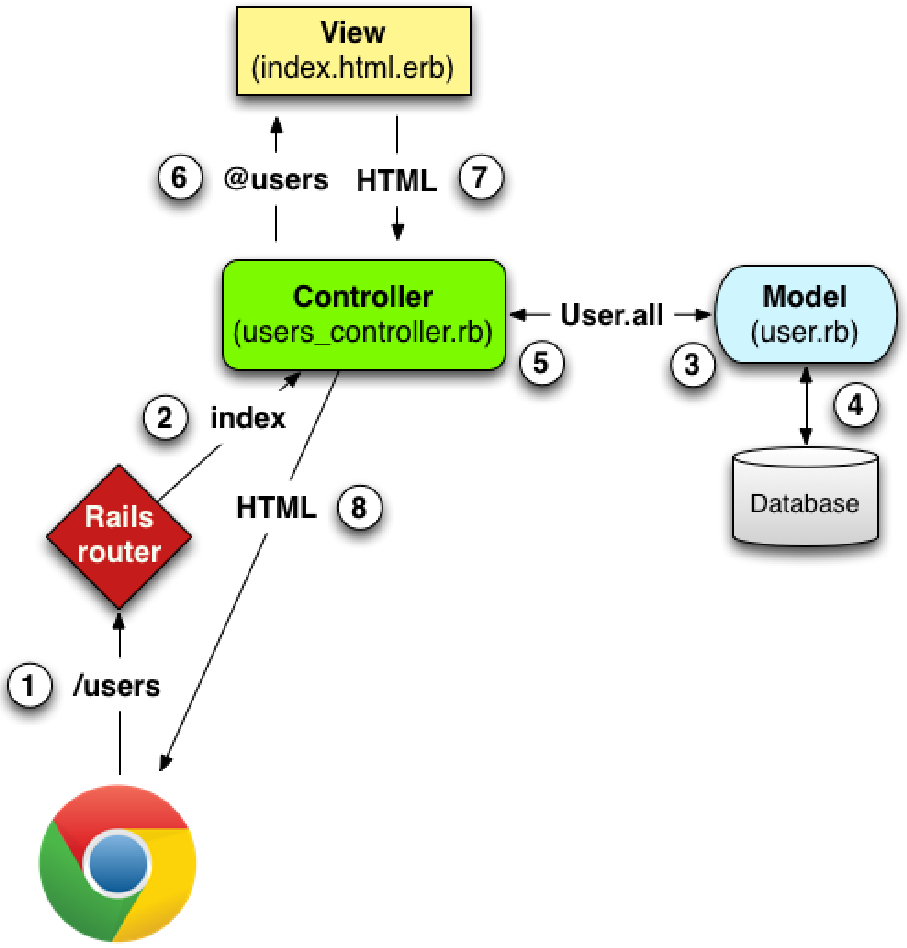
In config/routes.rb

```
root 'users#index'
```

- Display change in routes:

```
rails routes
```

MVC in action



MVC in action – User scaffolding

- The User model

In app/models/user.rb

```
class User < ApplicationRecord
end
```

- Represents *Active Record* – object tied directly to database record
- Prebuilt marshalling of all database fields
- Prebuilt querying by object

Controller view interaction Users controller

In app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  .
  def index
    @users = User.all
  end
  .
end
```

Breakdown:

- `User.all` – Calls *Active Record* method on `User` to return all users in the database
- `@users` – creates a variable `@users`, all variables with `@` sign are automatically available in the view
- View – Once the `index` function ends the controller framework calls the view with the matching name: `app/views/users/index.html.erb`

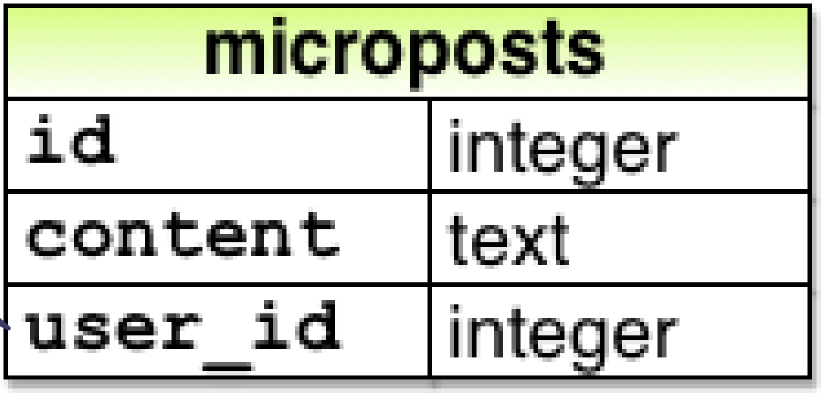
Controller view interaction
Users view

- *In app/views/users/index.html.erb*

```
<% @users.each do |user| %>
  <tr>
    <td><%= user.name %></td>
    <td><%= user.email %></td>
    <td><%= link_to 'Show', user %></td>
    <td><%= link_to 'Edit', edit_user_path(user) %></td>
    <td><%= link_to 'Destroy', user, method: :delete,
                data: { confirm: 'Are you sure?' } %></td>
  </tr>
<% end %>
```

Create microposts resource

This will be our foreign key to our users



microposts	
id	integer
content	text
user_id	integer

```
rails generate scaffold Micropost
content:text user_id:integer
```

Then update data model:

```
rails db:migrate
```

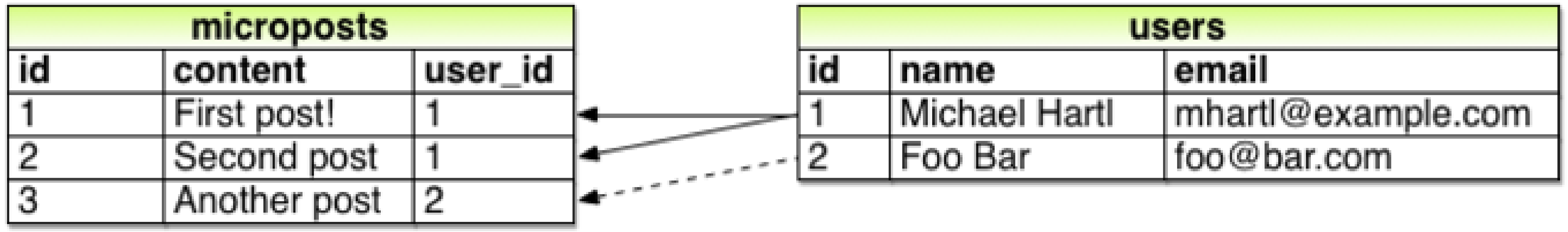
Then check the generated routes

Adding constraints/validation to model

- In app/models/micropost.rb

```
class Micropost < ApplicationRecord
  validates :content, length: { maximum: 140 }
end
```

Creating data associations



microposts		
id	content	user_id
1	First post!	1
2	Second post	1
3	Another post	2

users		
id	name	email
1	Michael Hartl	mhartl@example.com
2	Foo Bar	foo@bar.com

Need to tell objects about relationship

- In app/models/user.rb

```
class User < ApplicationRecord
  has_many :microposts
end
```

- In app/models/micropost.rb

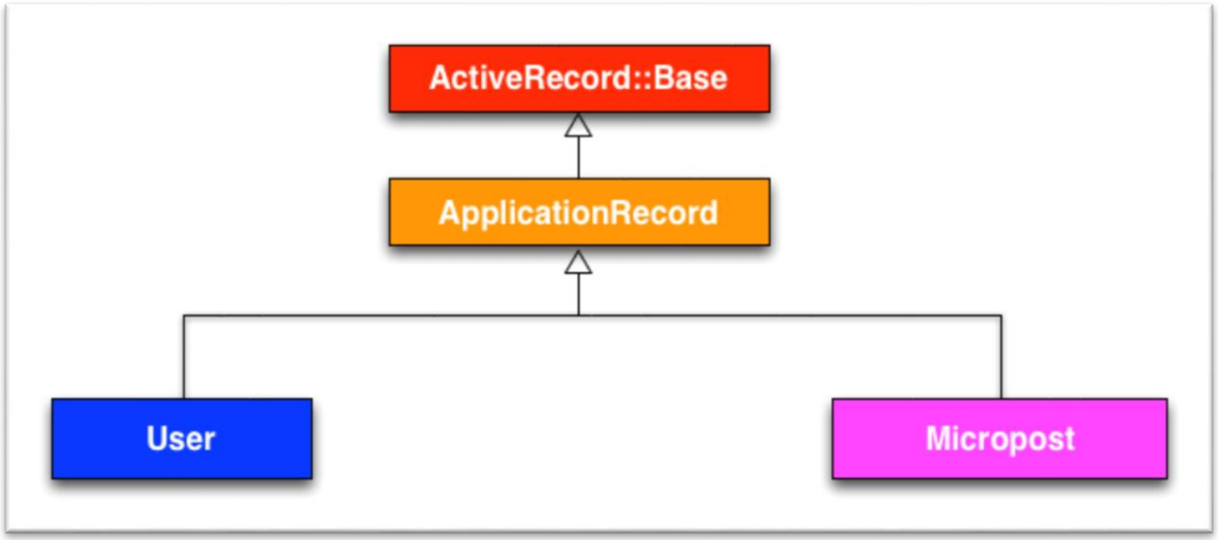
```
class Micropost < ApplicationRecord
  belongs_to :user
  validates :content, length: { maximum: 140 }
end
```

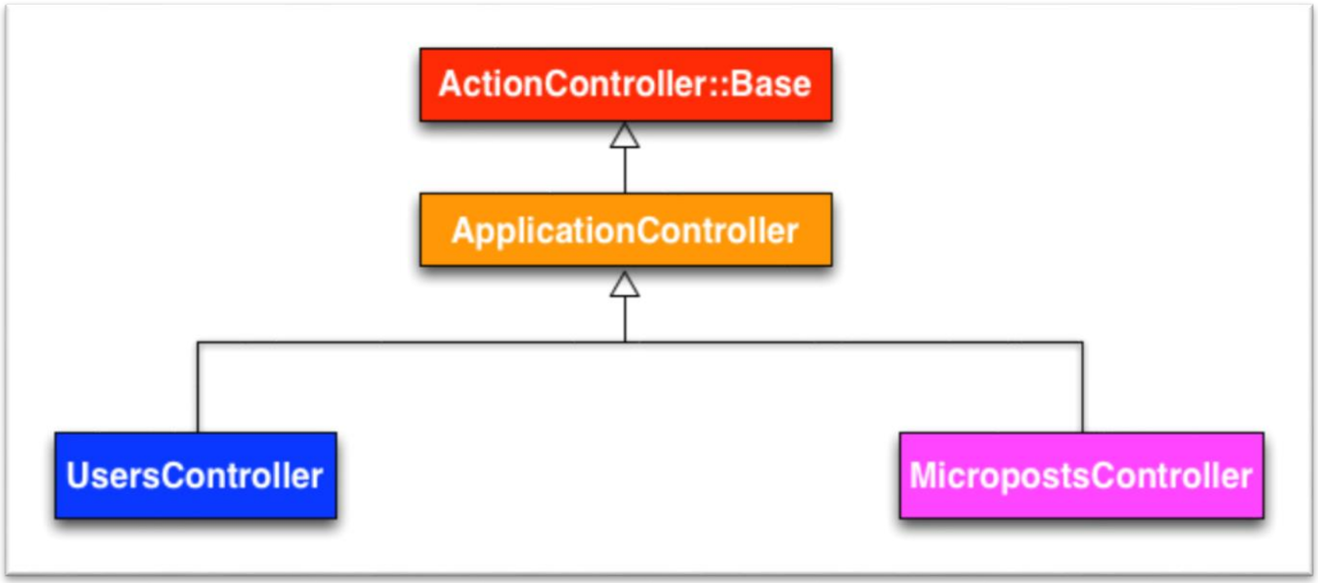
Rails console

- Although not necessary for development, the rails console can be useful for testing/visualize code especially data results

```
rails console
>> first_user = User.first
>> first_user.microposts
>> micropost = first_user.microposts.first
>> micropost.user
>> exit
```

Inheritance hierarchy so far…





Deploy to cloud and migrate DB

- Add changes to git

```
git add -A
git commit -m "Add micropost"
git push
```

- Push to Heroku

```
git push heroku
```

- Migrate DB changes to Heroku as well

```
heroku run rails db:migrate
```

Summary

- Scaffolding can be used to automatically create
  - Model – Object specified and matching DB
  - View – Template for displaying object
  - Controller – Default interactions populating/retrieving/persisting model elements and connecting to views
- What is REST architecture – standard set of URLs and controller actions for interacting with data models
- Routes – `rails routes`
- Easy to add data model constraints/validations
- Easy to add associations in data model
- `rails console`