Software Engineering
ECE444

Rails

Rails is…

- a framework to develop, deploy and maintain Web applications
  - start with a working application that you then change
  - testing support baked in
  - built in support for
    - Ajax
    - REST interfaces
  - all implemented in Ruby

Two key principles to apply…

… when writing Rails applications:

- DRY: Don't repeat yourself
  - Every piece of knowledge is expressed in just one place
- Convention over Configuration
  - sensible defaults everywhere

➔ follow Rails conventions and you write much less code

It's easy!

1. create a directory called, say, 'RoR'; cd into it

Command-line tool that comes with Rails

2. `> rails new demo` # create new app called demo

   .
   .
   . <outputs a lot of stuff, lists many files and directories>
   .

   # a fully function Web application called demo now exists

3. `> rake about` # tests various things

like Unix make

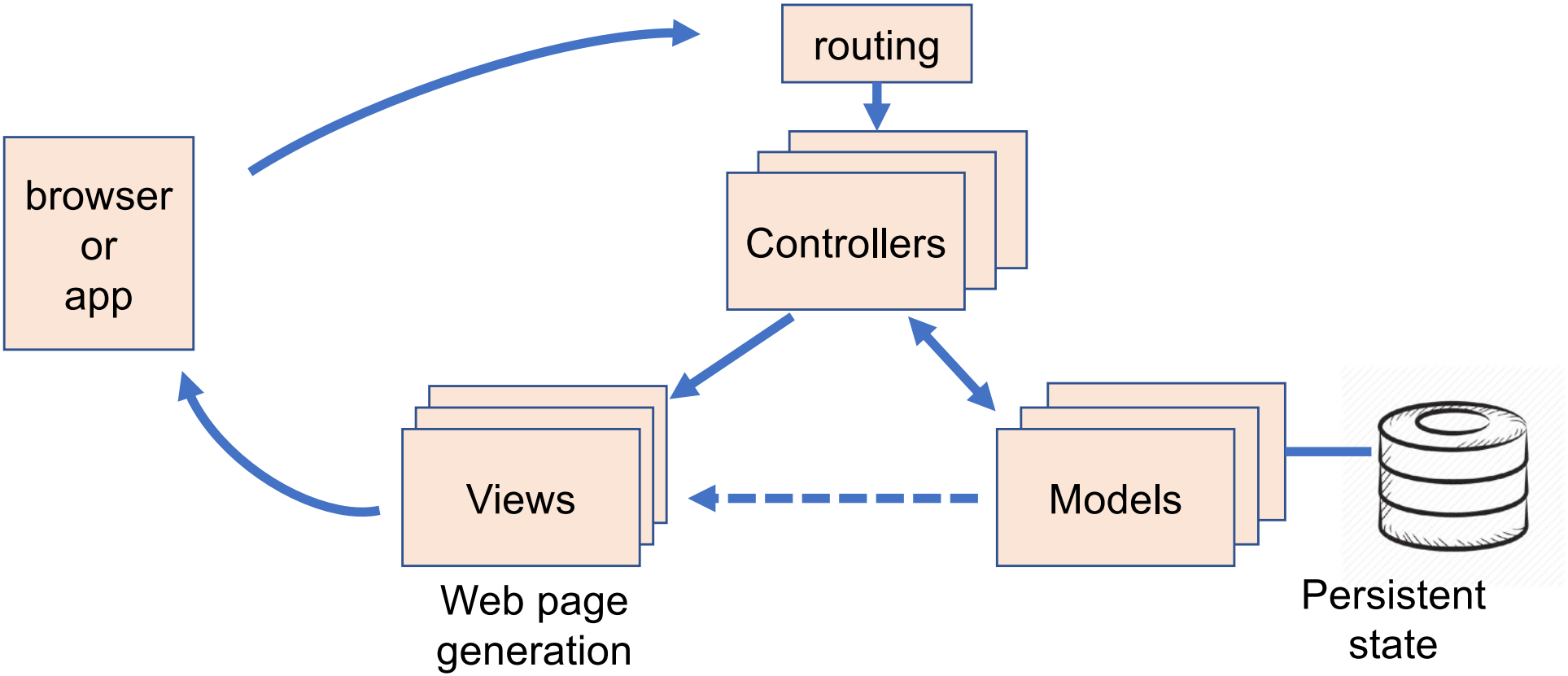
4. > rails server # starts up a Web server (WeBrick)

5. use a browser and go to http://localhost:3000

port

Architecture of Rails applications

- Architectural pattern: Model-View-Controller (MVC) structure for organizing your code



- Developers write code for all three independently
  Rails takes care of the independent plumbing

Directory hierarchy automagically created

- demo
  - gemfiles
  - app
    - controllers
      - . . .
    - models
      - . . .
    - views
      - . . .
  - config
  - db
  - lib
  - test
  - …

Get busy…

Homework:

1. Install Ruby on Rails
   (from https://rubyonrails.org)

2. create the "demo" app as per the above

3. make it work and test it

Model View Controller

Model:

- responsible for maintaining persistent state of the app
- maps DB into objs that can be accessed by Ruby
- hides complexity of DB & DB operations
- enforces business rules that apply to this data

View:

- responsible for generating user interface at times based on data in Model

Controller:

- orchestrates the application
- receives events from outside and then takes appropriate action
  - interacting with the model
  - displaying a appropriate view

Generate controller

```
> rails generate controller say hello goodbye
```

name of controller

actions the controller will support

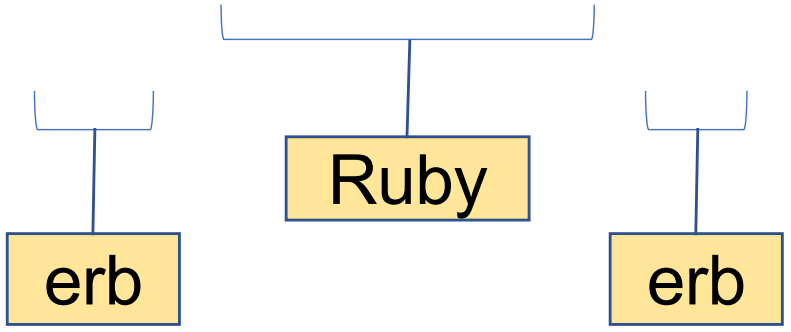
- creates all sorts of files, including:
  - app/controllers/say_controller.rb
  - app/views/say/hello.html.erb
  - app/views/say/goodbuy.html.erb

embedded ruby templating lang

- a scaffolding that works:
  - URL: localhost:3000/say/hello.html
  - URL: localhost:3000/say/goodbye.html
  - but localhost:3000/say/hi.html → error: "no route"

Modify generated hello.html.erb

```
<html>
    <head> <title>Hello!</title> </head>
    <body>
        <h1>Hello from Say!</h1>
        <p>
            It is now <%= Time.now %>
        </p>
    </body>
</html>
```



- Ruby code can also make calls to model

Generated controller

```
class SayController < ApplicationController

    def hello
    end

    def goodbye
    end

end
```

Superclass

after execution: automagically serves hello.html back

- You typically add code
- Can have any view returned, depending on computation. Default is `<name_of_action>.html`

Where to place code for logic in MVC

- Different styles spread code over models, controllers and views differently

- My recommendation:
  - Start by defining model/DB data and how it is organized

    ➔ will dominate design, and if designed well (and naturally), the rest will become evident….

Brief intro to routing: Background I

- Controller actions process (HTTP) requests (from browsers)
- Each HTTP request identified by:
  1. HTTP method/verb: `GET, POST, PUT, DELETE`
  2. URI; e.g., `localhost:3000/say/hello`
- Each action is handled by a Ruby method in a controller
- Hence, incoming HTTP requests must be mapped to appropriate controller and action
- This mapping is called a route defined in `config/routes.rb`

Brief intro to routing: Background II

- Rails expects you to follow "REST" architectural design principles (but does not mandate it)
- REST: Representational State Transfer (2000)
  - simple, but powerful
    - more scalable systems
    - easier to write applications
    - simpler interfaces
  - consistent way to map requests to actions so that each request is self-contained, containing all info need for the action;

    i.e., no sessions
    - server does not maintain client context
    - server is "stateless" (➔ no get first; get next; …)
    - client responsible for maintaining state

Brief intro to routing: Background III

- An organizing principle for SAAS apps:
  - identify entities manipulated by the app as resources
    (recall previous recommendation ☺)

    ➔ app = set of resources + actions on them
    (OO way of organizing a program)

- URIs specify resource:

```
localhost:3000/people
localhost:3000/people/34
```

- use HTTP methods to identify operation on a resource
  - POST : to Create a resource
  - GET : to Retrieve a resource
  - PUT : to Update/modify a resource
  - DELETE : to Delete a resource

CRUD:
standardized use
of HTTP action verbs

Brief intro to routing: Background IV

- Problem: browsers don't all support these HTTP methods

  ➔ if not, Rails uses POST and annotates Web forms to identify the method under the covers and then adjusts the HTTP method for the controller as if the HTTP method had been submitted

- RESTful interfaces are powerful
  ➔ you should use them
  ➔ Amazon: all services (even internal ones) use them

Brief intro to routing: Configuring routes

- to get HTTP request routed to right controller
- Add to `config/routes.rb`:

```
Rails.application.routes draw do
    resources  :people
end
```

This automagically generates many routes.

To list them, run: `rake routes`

Brief intro to routing: Routes table

- `rake routes` outputs:

Prefix	Verb	URI	Controller#action	Used for
	GET	/people	people#index	display list of people
	POST	/people	people#create	create a new person
	GET	/people/new	people#new	get form for new person
	GET	/people/:id	people#show	disply specific person
	GET	/people/:id/edit	people#edit	get form for editing person
	PATCH	/people/:id	people#update	update a person
	PUT	/people/:id	people#update	update a person
	DESTROY	/people/:id	people#destroy	destroy a person

In controller `PeopleController`
you need to define action

Brief intro to routing: I

- This means, e.g., `GET /people/new` HTTP request
  - causes `new` action (method) in `PeopleController` to be called

- And `GET /people/17` HTTP request
  - causes `show` action in `PeopleController` to be called
  - and an automagically generated hash called `params` available to the controller will include the key-value pair `{id: 17}`.

Brief intro to routing: II

- Most general route specification:
  ```
  get ':controller/:action/:id'
  ```
  - will route `GET /xxx/yyy/17` to controller `XxxController`'s `yyy` action
  - and `17` is available as `params[:id]`
- hash params will also include all parameters from query string; e.g.,
  ```
  GET xxx/yyy/17?zzz=2
  ```
  will also place `zzz: '2'` in params.
- Finally: route
  ```
  root  'welcome#index'
  ```
  (GET /)
  is also useful…
- More details to come…