CS169 Week 3 Section

9/17/19

Attendance

http://j.mp/cs169section

Slides thanks to: Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Discussion Feedback

tinyurl.com/169disc

Administrivia & Agenda

- HW2 is due this Friday 9/20
- HW3 to be released this week
- Project groups/lists to be released later this week

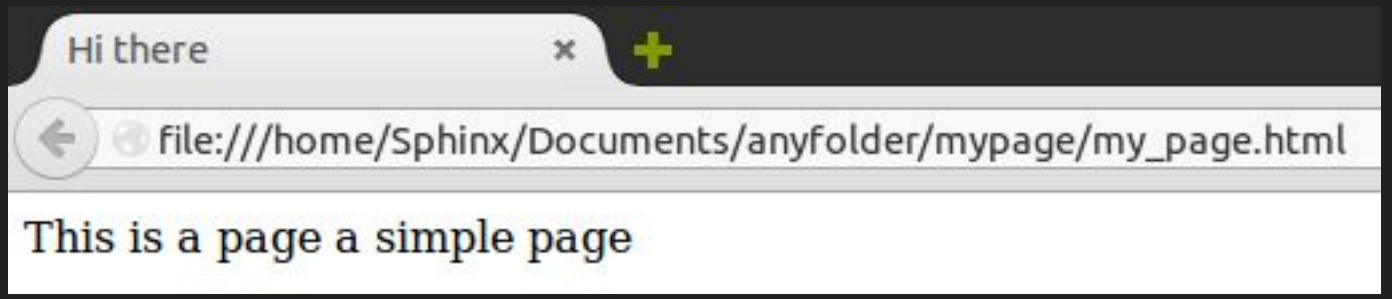
Today's goal: Explain the concepts of MVC by going through a basic TODO app (in Sinatra)

Slides thanks to: Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

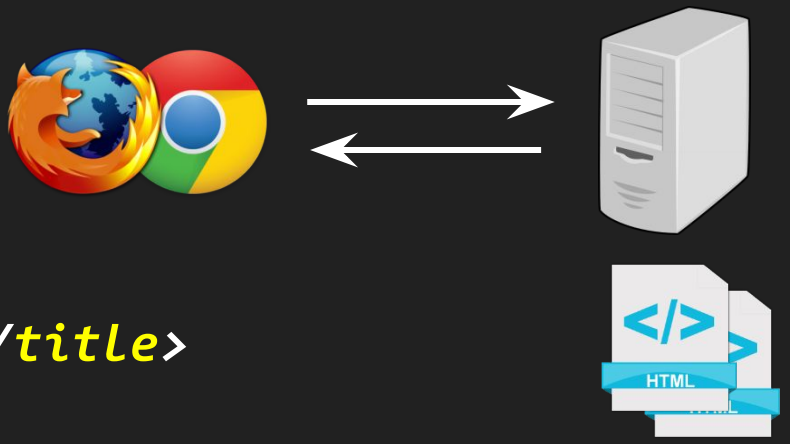
Template

HTML

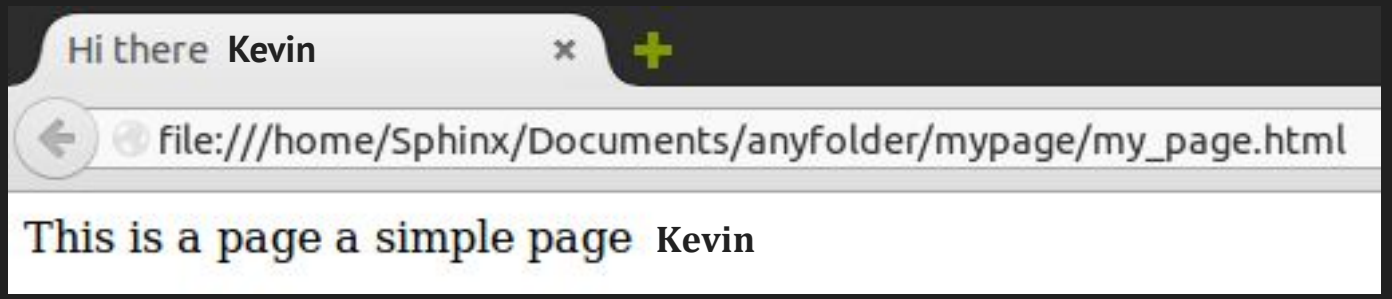
```
<!DOCTYPE html>
<html>
    <head>
        <title>Hi there</title>
    </head>
    <body> This is a page a simple page </body>
</html>
```



HTML

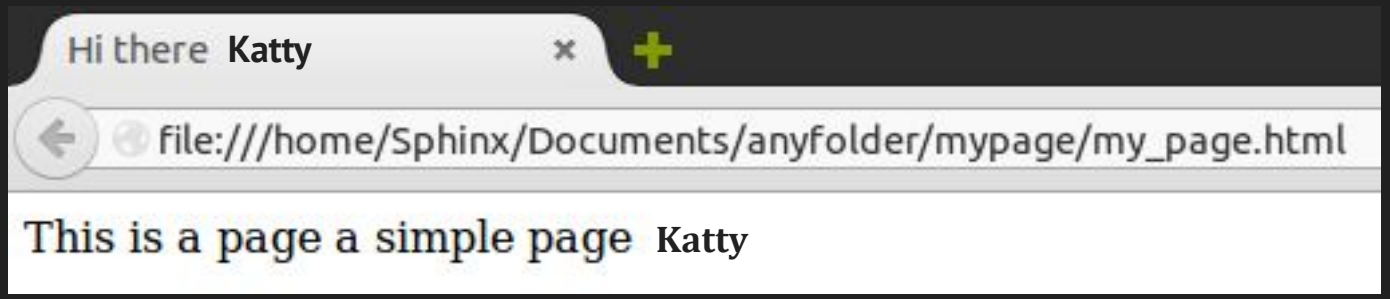



```
<!DOCTYPE html>
<html>
    <head>
        <title>Hi there Kevin</title>
    </head>
    <body> This is a page a simple page Kevin </body>
</html>
```

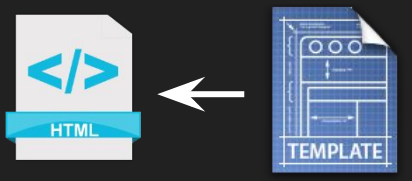


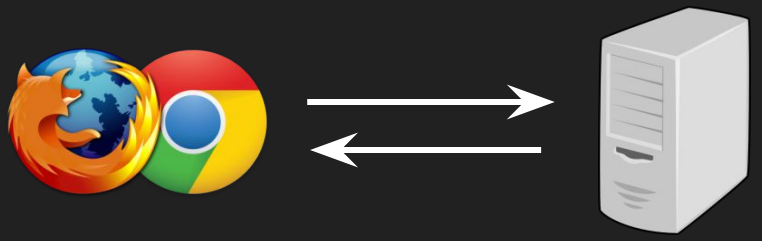
HTML

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hi there Katty</title>
    </head>
    <body> This is a page a simple page Katty </body>
</html>
```

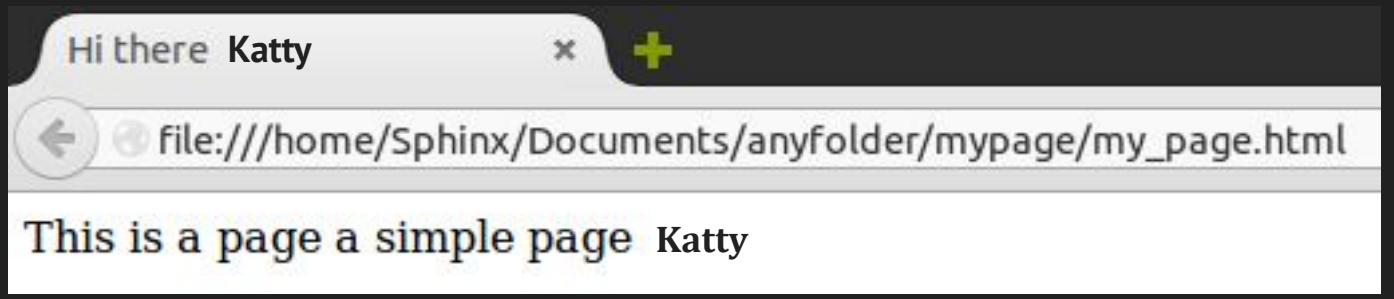


HTML



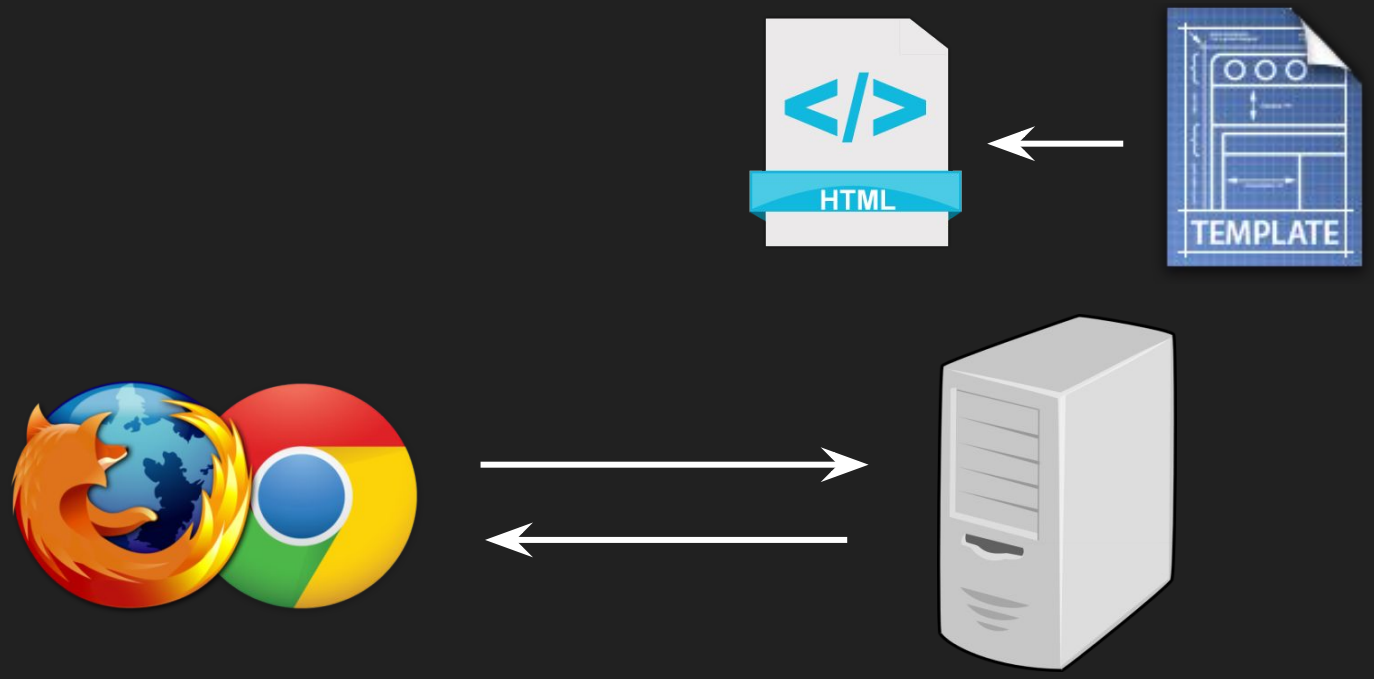


```
<!DOCTYPE html>
<html>
    <head>
        <title>Hi there <%= @name %></title>
    </head>
    <body> This is a page a simple page <%= @name %>
    </body>
</html>
```



Web Programming

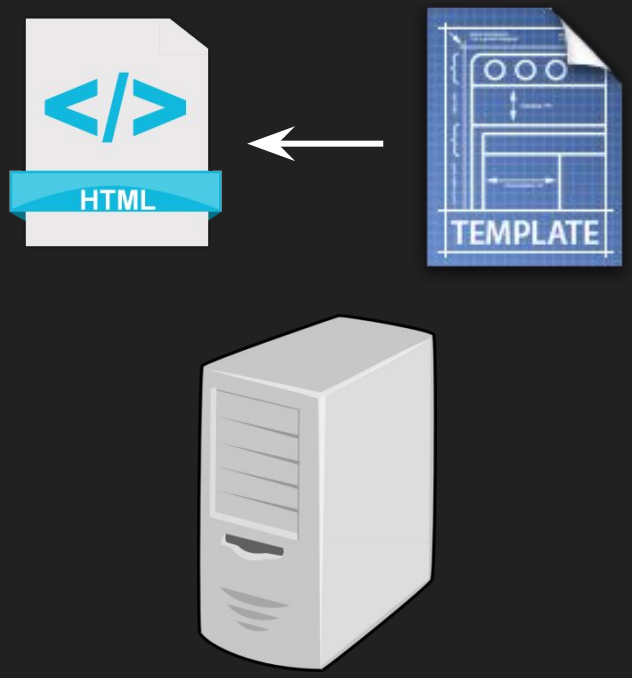
Dynamically generate HTML from template according to user's request.



Web Programming

Dynamically generate HTML from template according to user's request.

Ruby (Sinatra, Rails)
PHP (Wordpress)
Python (Django)
Java → JSP
C# → ASP.NET
JavaScript → Node.js
C++ → CGI
…



Do not put everything into the template!

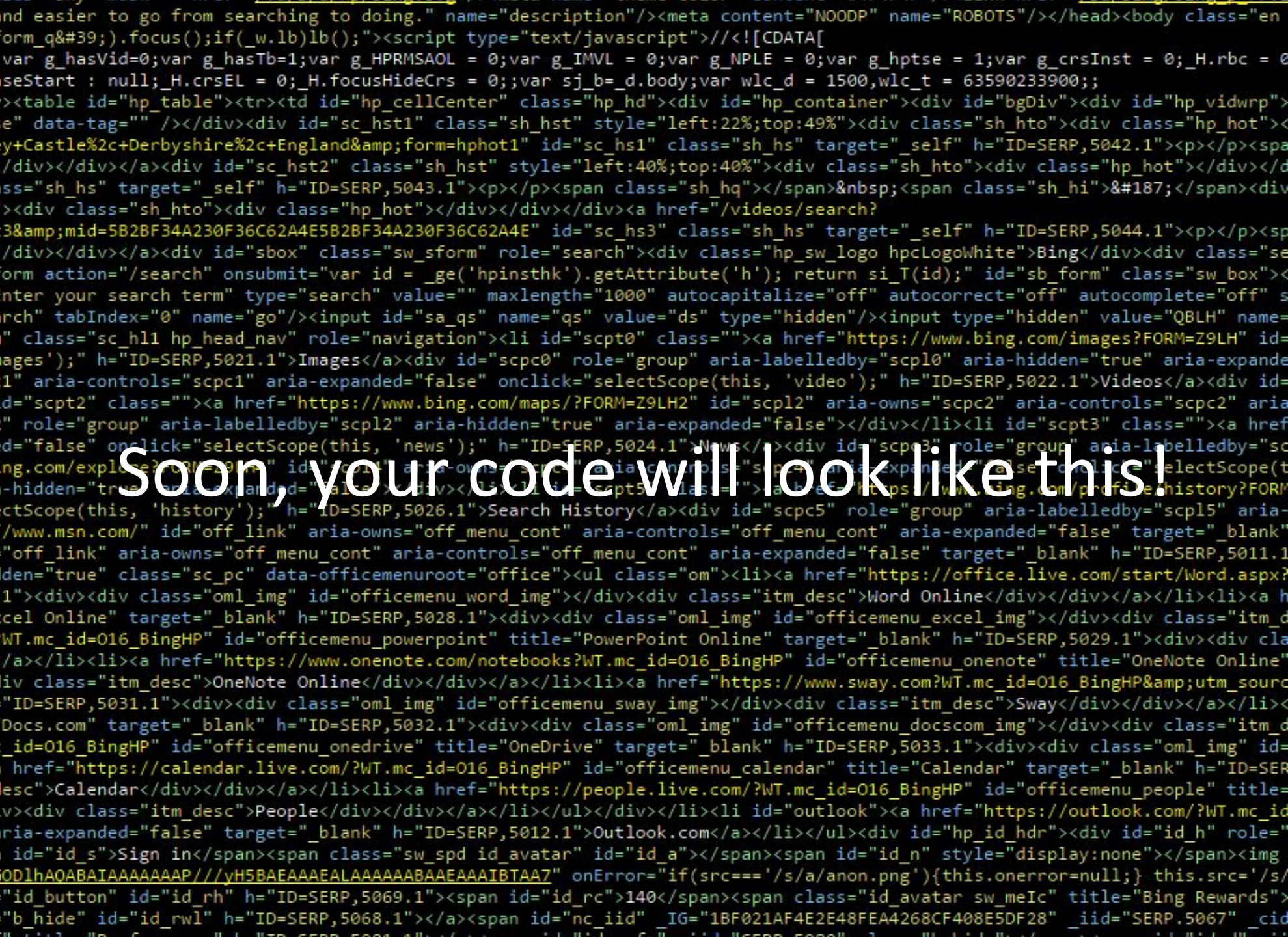
```
<!DOCTYPE html>
<html>
    <%
    @user = User.find(params[:id])
    %>
    <head>
        <title>Hi there <%=@user.name%></title>
    </head>
    <body> This is a page a simple page </body>
</html>
```

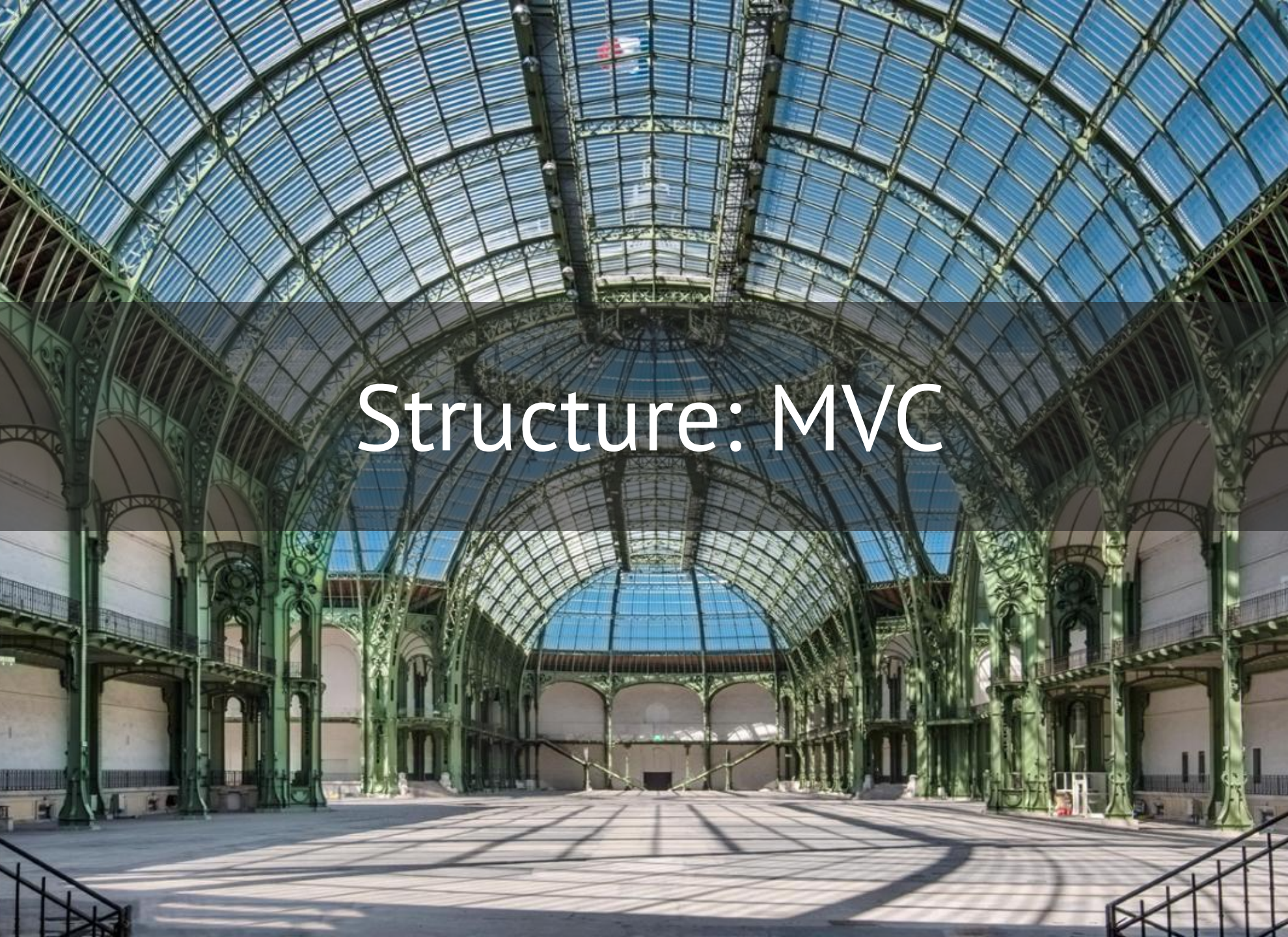
Do not put everything into the template!

```
<!DOCTYPE html>
<html>
    <%
    @user = User.find(params[:id])
    @movies = Movies.find_by_user_id(@user.id)
    %>
    <head>
        <title>Hi there <%=@user.name%></title>
    </head>
    <body> This is a page a simple page
    <%= @movies... %>
    </body>
</html>
```

Do not put everything into the template!

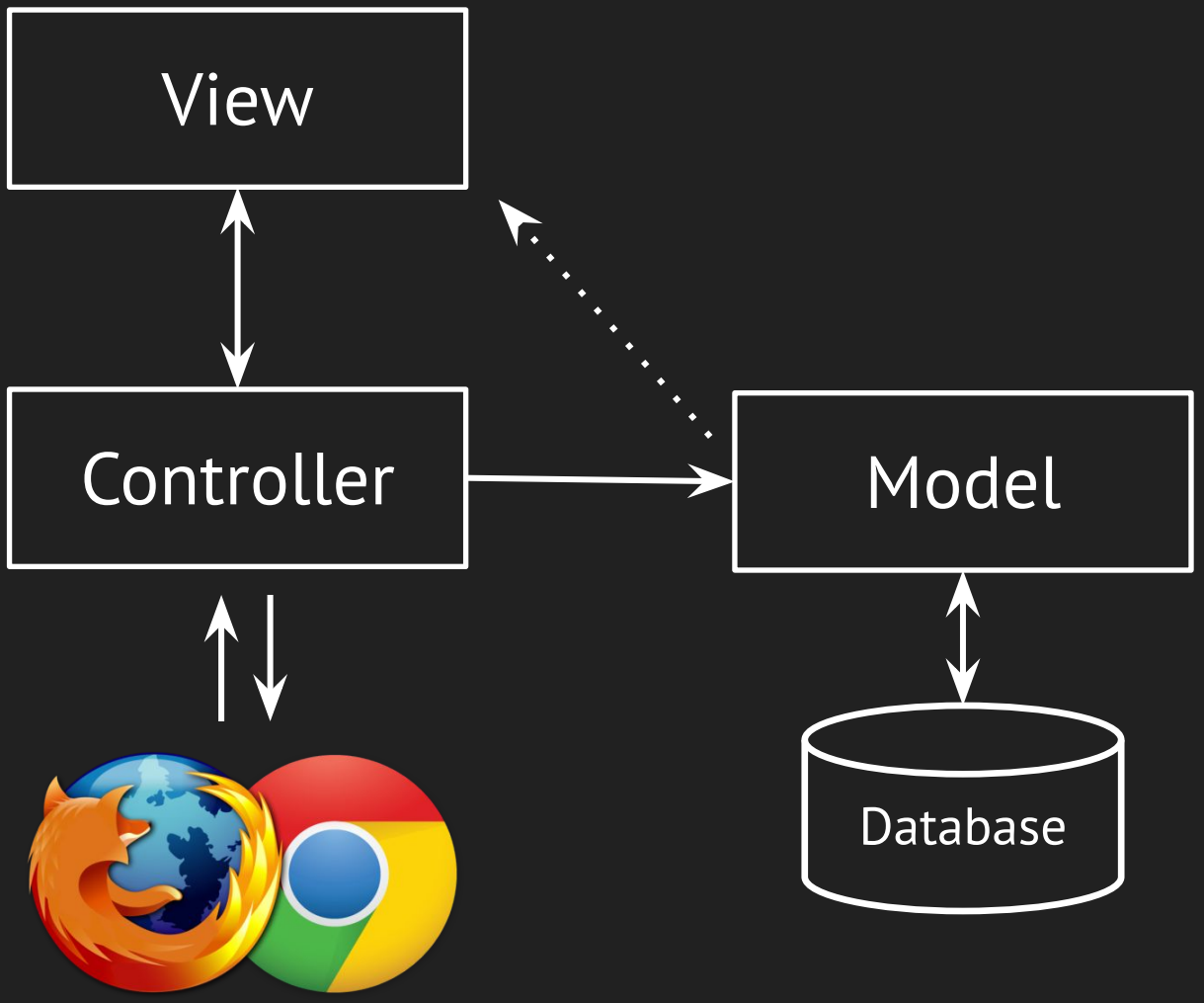
```
<!DOCTYPE html>
<html>
    <%
    @user = User.find(params[:id])
    @movies = Movies.find_by_user_id(@user.id)
    config.active_record.raise_in_transactional_callbacks = true
    ...
    %>
    <head>
        <title>Hi there <%=@user.name%></title>
    </head>
    <body> This is a page a simple page
    <%= @movies... %>
    </body>
</html>
```


Soon, your code will look like this!

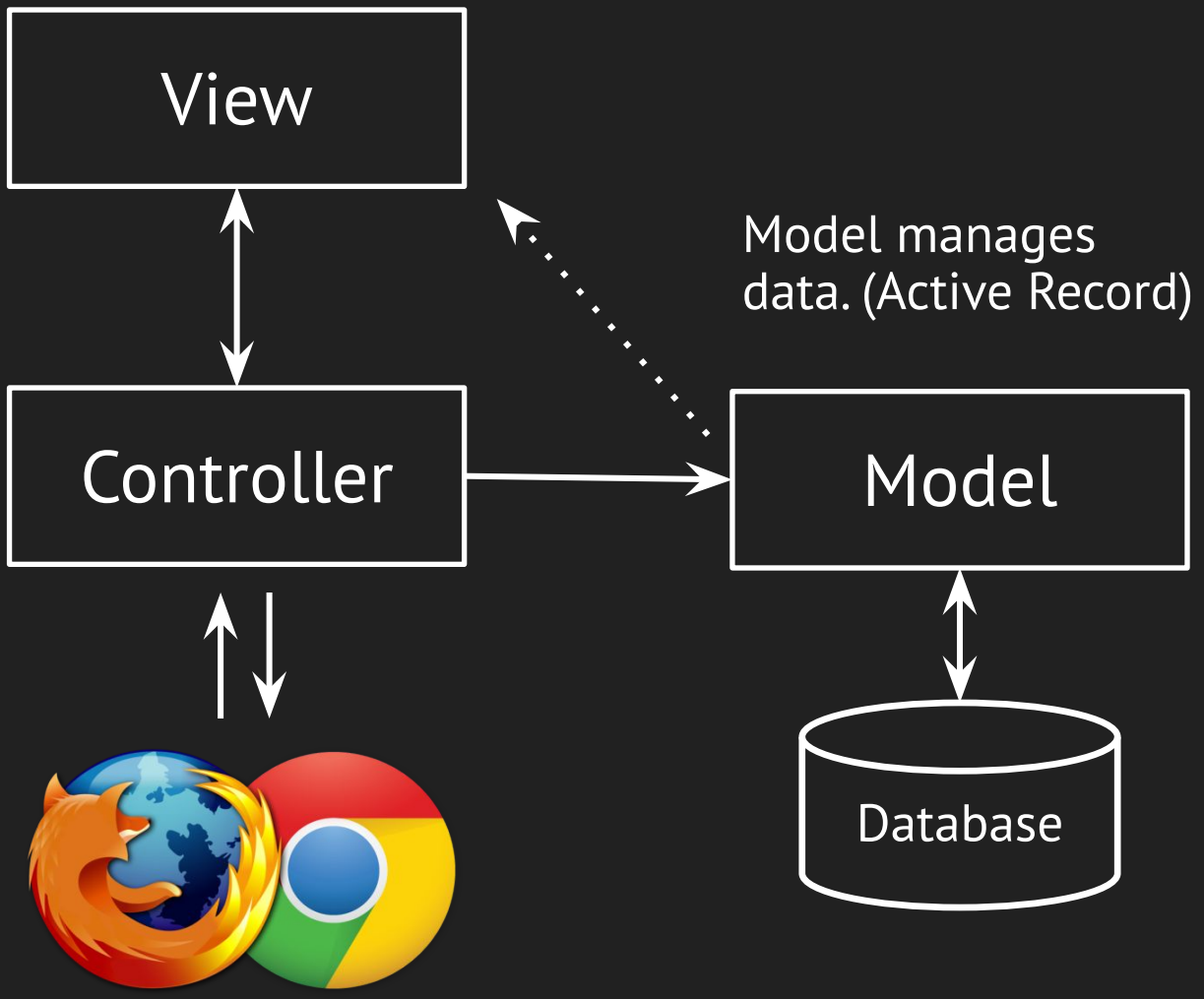


Structure: MVC

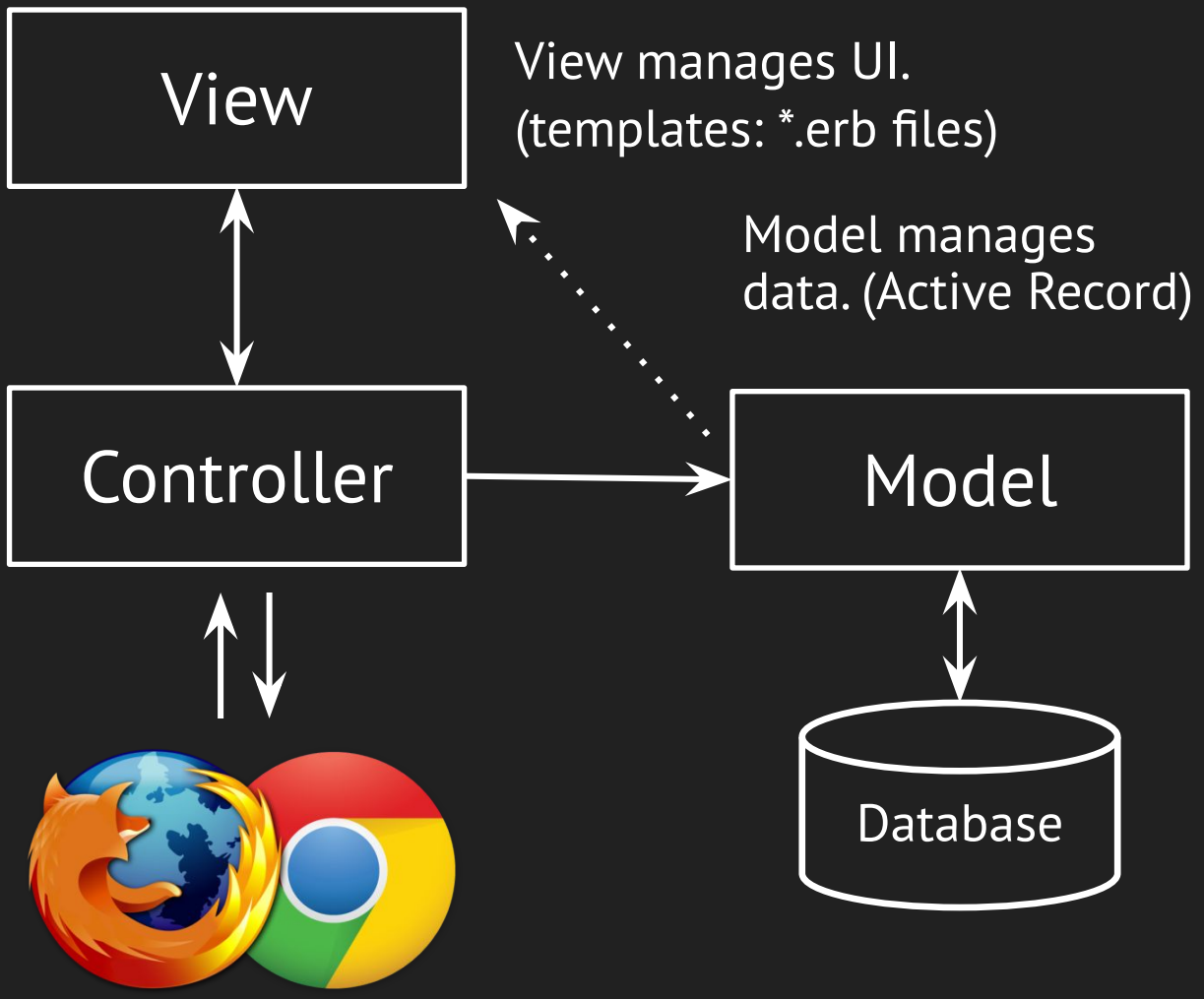
MVC



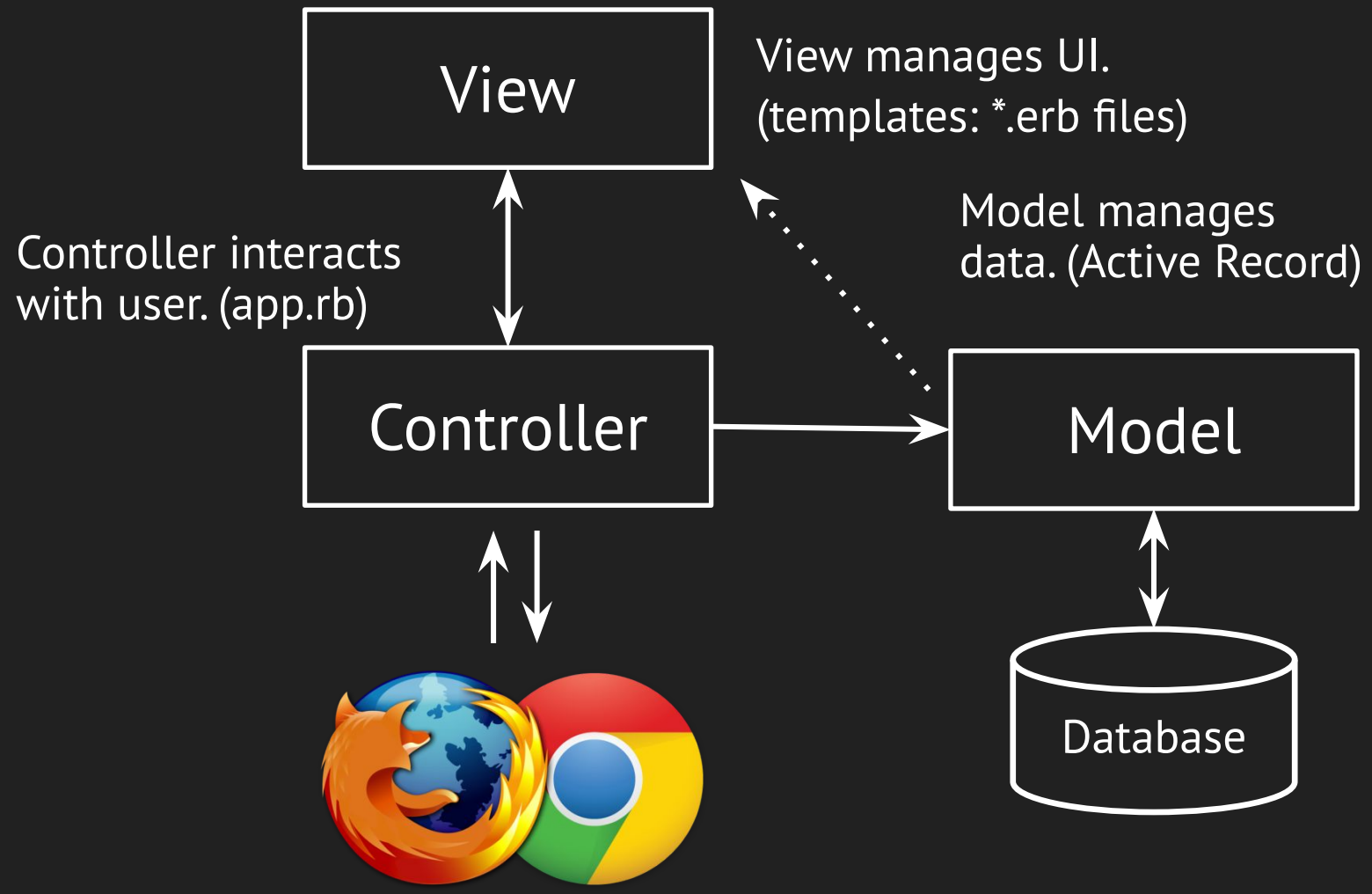
MVC


View
Model manages
data. (Active Record)
Controller
Model
Database

MVC


View
View manages UI.
(templates: *.erb files)
Model manages
data. (Active Record)
Controller
Model
Database

MVC


View
View manages UI.
(templates: *.erb files)
Controller interacts
with user. (app.rb)
Model manages
data. (Active Record)
Controller
Model
Database

Sinatra vs Rails

Sinatra is a library

- dealing with HTTP from the server side.
- Your code invokes Sinatra's code
- Sinatra does not contain MVC, you can use MVC with Sinatra though.

Rails is a MVC framework

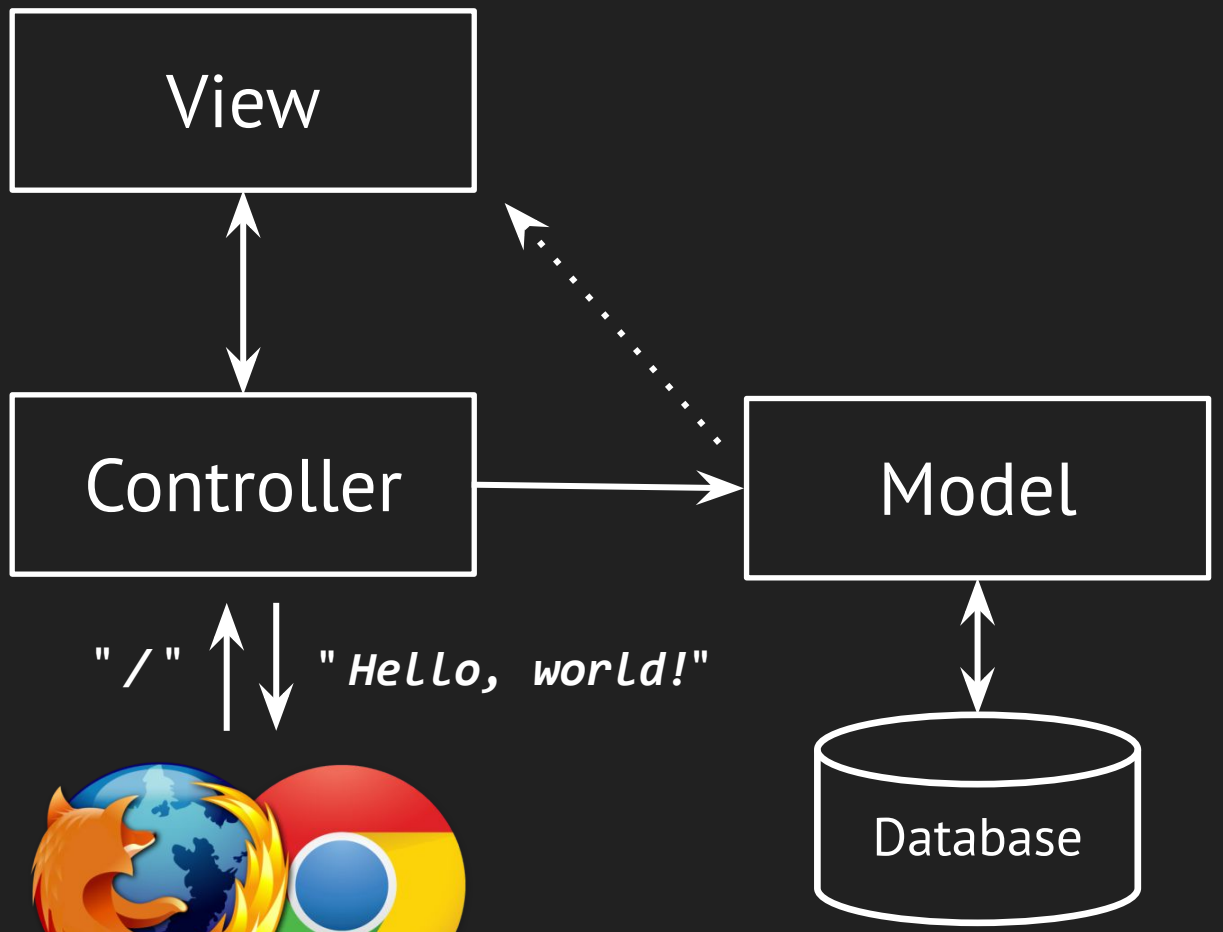
- boost productivity
- Rails generates code that invokes your code
- Rails supports MVC

Sinatra

```
require "sinatra"

get "/" do
  "Hello, world!"
end
```

MVC




```
get "/" do
    "Hello, world!"
end
```

Sinatra

```
require "sinatra "

get "/getname/:name" do
   "#{params[:name]}"
end
```

Sinatra

```
get "/form" do
   erb: form
end

post "/form" do
    " You said #{params[:message]}"
end
```

Sinatra

```
get "/form" do
  erb: form
end

post "/form" do
  " You said #{params[:message]}"
end
```

```
<form action="/form" method="post">
    <input type="text" name="message">
    <input type="submit">
</form>
```

Sinatra

```
get "/form" do
  erb: form
end
```

```
<form action="/form" method="post">
    <input type="text" name="message">
    <input type="submit">
</form>
```

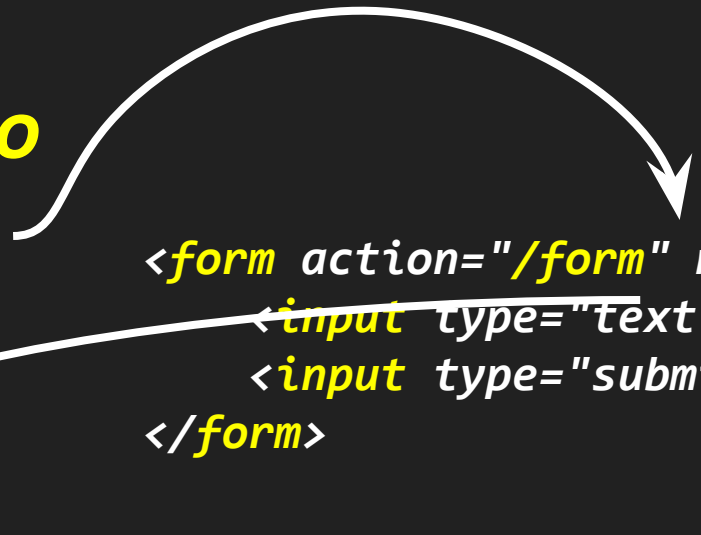
```
post "/form" do
  " You said #{params[:message]}"
end
```

Sinatra

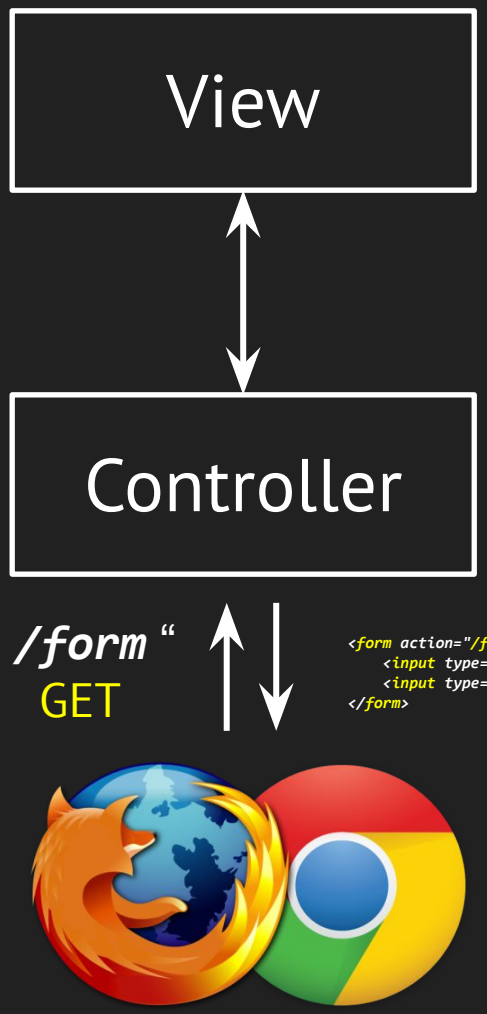
```
get "/form" do
  erb: form
end
```

```
<form action="/form" method="post">
    <input type="text" name="message">
    <input type="submit">
</form>
```

```
post "/form" do
  " You said #{params[:message]}"
end
```



MVC

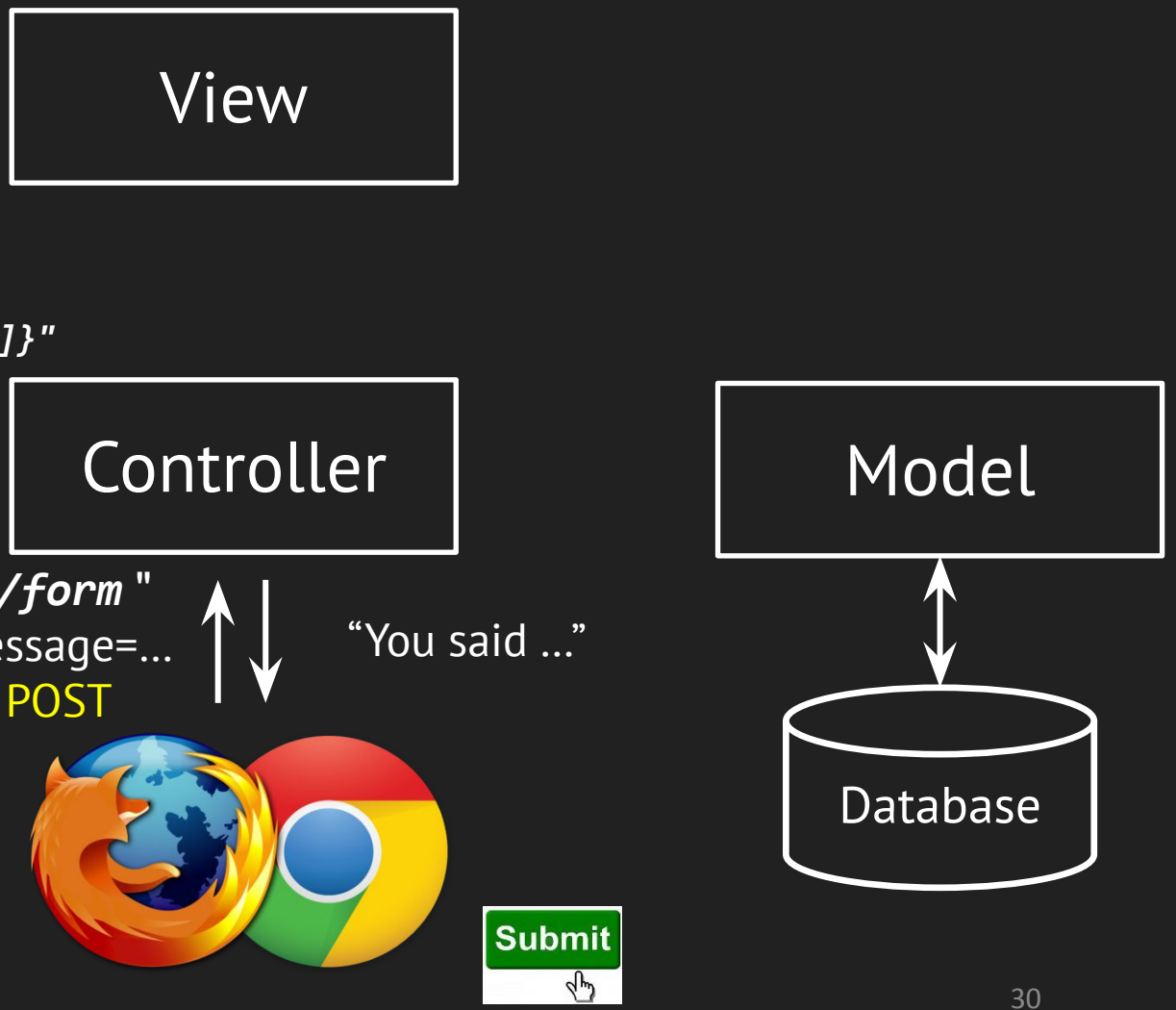



```
<form action="/form" method="post">
    <input type="text" name="message">
    <input type="submit">
</form>
```

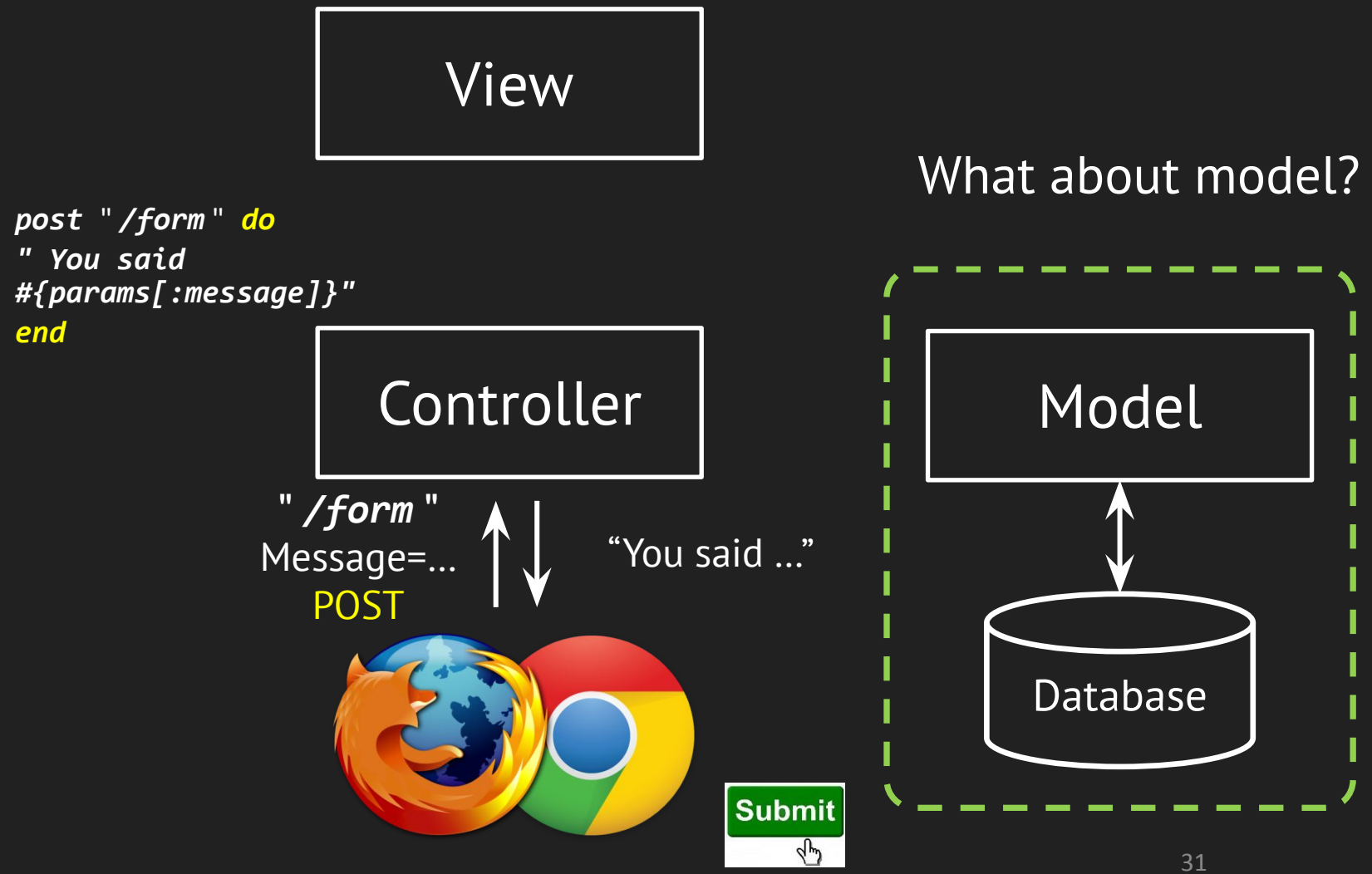
```
get "/form" do
    erb: form
end
```

MVC

```
post "/form" do
" You said
#{params[:message]}"
end
```



MVC


View
What about model?
post "/form" do
" You said
#{params[:message]}"
end
Controller
Model
"/form"
Message=...
POST
"You said ..."
Database
Submit

Active Record

- An Implementation of the object-relational mapping (ORM) pattern.
- Automated mapping between classes and tables, attributes and columns
  - Basic operations on object: CRUD
  - (Create, Read, Update, Delete)
- Associations between objects defined by simple class methods (will be covered later)

Example

```
class Article < ActiveRecord::Base {
  :id      => :integer,
  :title   => :string,
  :content => :text
}
```

AR automatically handles the mapping between:

- objects in memory
- Records in database

id	title	content
1	First record	Hello world
2	Week 3 section	Active record etc.
3	Week 4 section	Rails etc.

Example

```
class Article < ActiveRecord::Base {
  :id      => :integer,
  :title   => :string,
  :content => :text
}
```

AR automatically handles the mapping between:

- objects in memory
- Records in database

```
a = Article.new
a.title = "Week 5"
a.save
```

id	title	content
1	First record	Hello world
2	Week 3 section	Active record etc.
3	Week 4 section	Rails etc.
4	Week 5	

Example

```
class Article < ActiveRecord::Base {
  :id      => :integer,
  :title   => :string,
  :content => :text
}
```

AR automatically handles the mapping between:

- objects in memory
- Records in database

```
Article.create(
  :title => "Week 5"
)
```

id	title	content
1	First record	Hello world
2	Week 3 section	Active record etc.
3	Week 4 section	Rails etc.
4	Week 5	

ActiveRecord & SQL

AR automatically handles the mapping between:

- objects in memory
- Records in database

AR will translate the query API call to SQL commands:

```
Article.where(:title => "Week 5")
> select * from Article where title = "Week 5"
```

The result will be put into objects in memory.

id	title	content
1	First record	Hello world
2	Week 3 section	Active record etc.
3	Week 4 section	Rails etc.
4	Week 5	

ActiveRecord Methods

```
class Article < ActiveRecord::Base {
  :id      => :integer,
  :title   => :string,
  :content => :text
}
```

ActiveRecord will dynamically "generate" methods based on the attributes in the table:

- `Article.find_by_title("Week 5")`
- `Article.find_by_title_and_content(`
  `"First Record", "Hello world")`

Use ActiveRecord in Sinatra

```
require 'sinatra'
# db configuration
db_options = {adapter: 'sqlite3', database: 'todos_db' }
# establish connection
ActiveRecord::Base.establish_connection(db_options);
# write migration
class CreateTodos < ActiveRecord::Migration
  def change; create_table :todos ... end
end
# enforce migration
createTodos.new.change
# define the active record
class Todo < ActiveRecord::Base
  ...
end
```

No need to memorize this.
Rails generates this for you.

However, it is good to know how things works under the hood!
A small Sinatra-with-ActiveRecord Demo:

http://bit.ly/2kkfYJu