

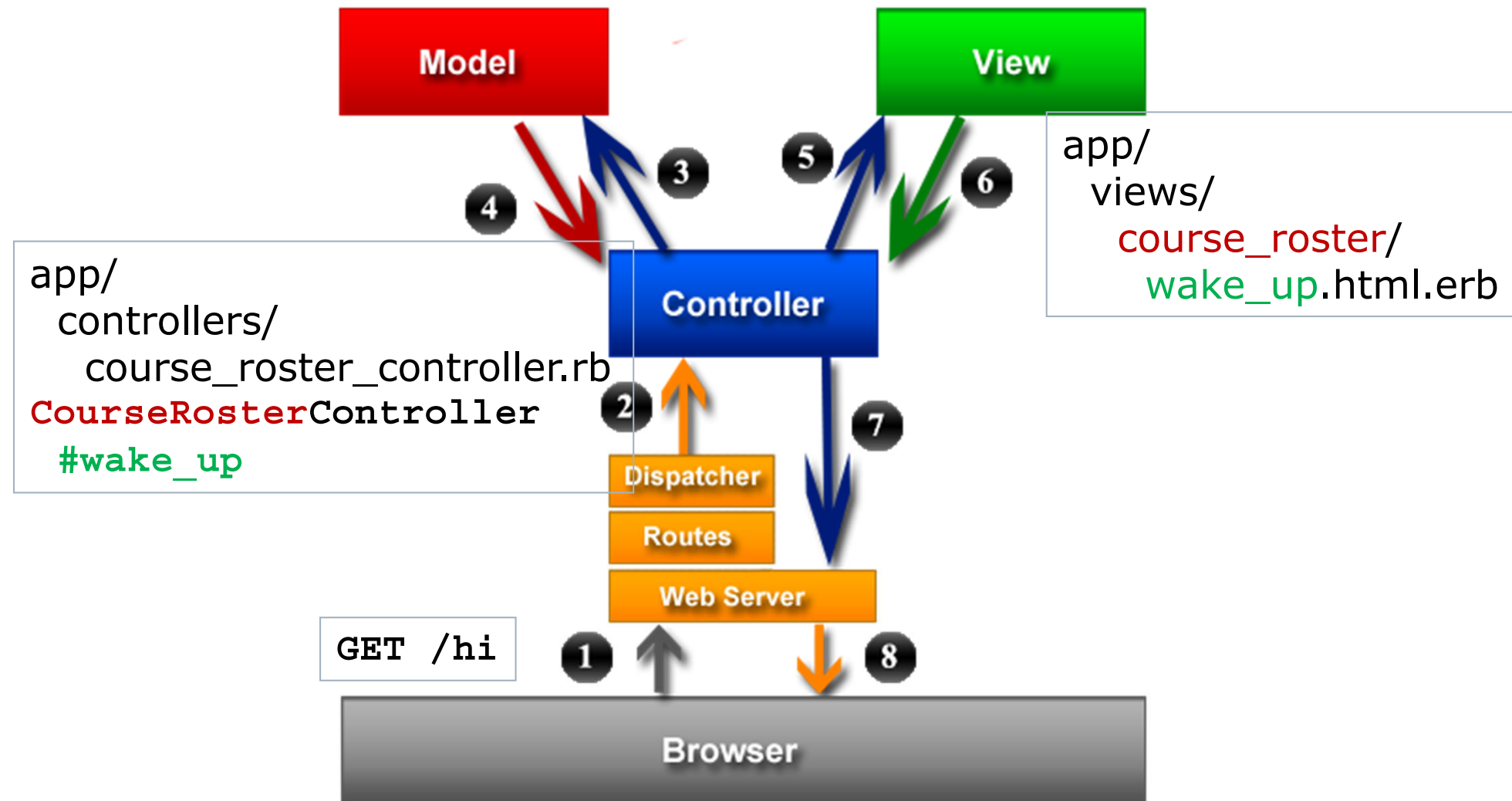
Rails: Views and Controllers II

Computer Science and Engineering ■ College of Engineering ■ The Ohio State University

Lecture 32

Recall: Rails Architecture; Files, Locations

Computer Science and Engineering ■ The Ohio State University



Recall: Wiring Views and Controllers

- A controller is just an ordinary Ruby class
 - Extends ApplicationController

```
class CourseRosterController <
  ApplicationController
```

 - Location: `app/controllers/`
 - Filename: `course_roster_controller.rb`
- Actions are methods in that class

```
  def wake_up
    ...
  end
```
- A view is an HTML page (kind of) that corresponds to that action
 - Location: `app/views/course_roster/`
 - Filename: `wake_up.html.erb`
 - Has access to *instance* variables (e.g., `@student`) of corresponding controller!

Recall: REST Routes and Wiring

- For a resource like `:students`, the action pack includes
 - 1 controller (`StudentsController`)
 - 7 routes (each with a method in controller)
 - 4 views (list all, show one, blank form, edit form)

HTTP Verb	URL	Resource	Method	Response (View)
GET	<code>/students</code>	Collection	index	list all
POST	<code>/students</code>	Collection	create	show one
GET	<code>/students/new</code>	Collection	new	blank form
GET	<code>/students/3</code>	Member	show	show one
GET	<code>/students/3/edit</code>	Member	edit	filled form
PUT	<code>/students/3</code>	Member	update	show one
DELETE	<code>/students/3</code>	Member	destroy	list all

Example: books/show.html.erb (Render)

```
<p style="color: green"><%= notice %></p>
```

```
<%= render @book %>
```

```
<div>
```

```
  <%= link_to "Edit this book",  
            edit_book_path(@book) %> |
```

```
  <%= link_to "Back to books", books_path %>
```

```
  <%= button_to "Destroy this book", @book,  
              method: :delete %>
```

```
</div>
```

Example: books/index.html.erb

```
<p style="color: green"><%= notice %></p>
```

```
<h1>Books</h1>
```

```
<div id="books">
```

```
  <% @books.each do |book| %>
```

```
    <%= render book %>
```

```
    <%= link_to "Show this book", book) %>
```

```
  <% end %>
```

```
</div>
```

```
<%= link_to "New book", new_book_path %>
```

Creating a Response

- There are 3 ways a controller action can create the HTTP response:
 1. Do nothing: defaults are used
 2. Call render method
 3. Call redirect method
- The first results in HTTP status 200 (OK)
 - Body of response is the HTML of the view
- The 3rd results in HTTP status 302 (temporary redirect)
- Different formats for body of response are possible (HTML, JSON, plain text...)

1: Default Response

- If the action does not call render (or redirect), then render is implicitly called on corresponding view

```
class BooksController <
  ApplicationController

  def index
    @books = Book.all
  end
end
```

- Results in call to render

```
app/views/books/index.html.erb
```

2: Explicitly Calling Render

- Argument: *action* whose *view* should be rendered

```
def wake_up
  render :show # or render "show"
end
def show ...
```

- Action (show) does *not* get executed

- Action could be from another controller

```
render 'products/show'
```

- Can return text (or json or xml) directly

```
render plain: "OK"
render json: @book # calls to_json
render xml: @book # calls to_xml
```

- Note: render *does not* end action, so don't call it twice ("double render error")

3: Calling Redirect

- ❑ Sends response of an HTTP redirect (3xx)
 - Default status: 302 (temporary redirect)
 - Override for permanent redirection (301)
- ❑ Consequence: client (browser) does *another* request, this time to the URL indicated by the redirect response
 - New request is a GET by default
- ❑ Need URL, can use named route helpers

```
redirect_to book_url(@book)
redirect_to books_path
redirect_to edit_book_path(@book)
redirect_to @book # calls url_for(@book)
```
- ❑ Or :back to go back in (client's) history

Redirect vs Render

□ Similarity

- Point to a different view
- Neither ends the action

`render...` and `return # force termination`

□ Difference

- Redirect entails 2 round-trips: request action response; request action response
- Redirect requires a URL as argument, Render takes a view (action)

□ Common usage for Redirect: POST-Redirect-GET pattern

GET Blank Form, POST the Form

The image shows two screenshots of a web browser (Chromium) displaying the Quickroster application. The top screenshot shows the 'Students' page with a list of students. The 'New student' link is circled in red. The bottom screenshot shows the 'New student' form with fields for 'Fname', 'Lname', and 'Buckid'. The 'Create Student' button is circled in red.

Students

Fname: Marco
Lname: Pantani
Buckid: 34822039
[Show this student](#)
[New student](#)

New student

Fname
Galileo
Lname
Buckid
Create Student
[Back to students](#)

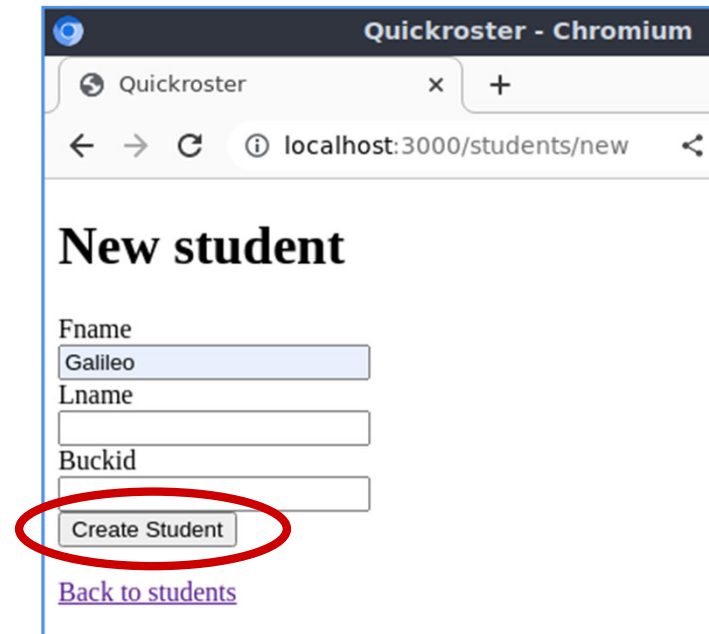
GET *"a blank form"*



POST /students
lname: ...etc



GET Blank Form, POST the Form (2)



Quickroster - Chromium

Quickroster x +

localhost:3000/students/new

New student

Fname
Galileo

Lname

Buckid

Create Student

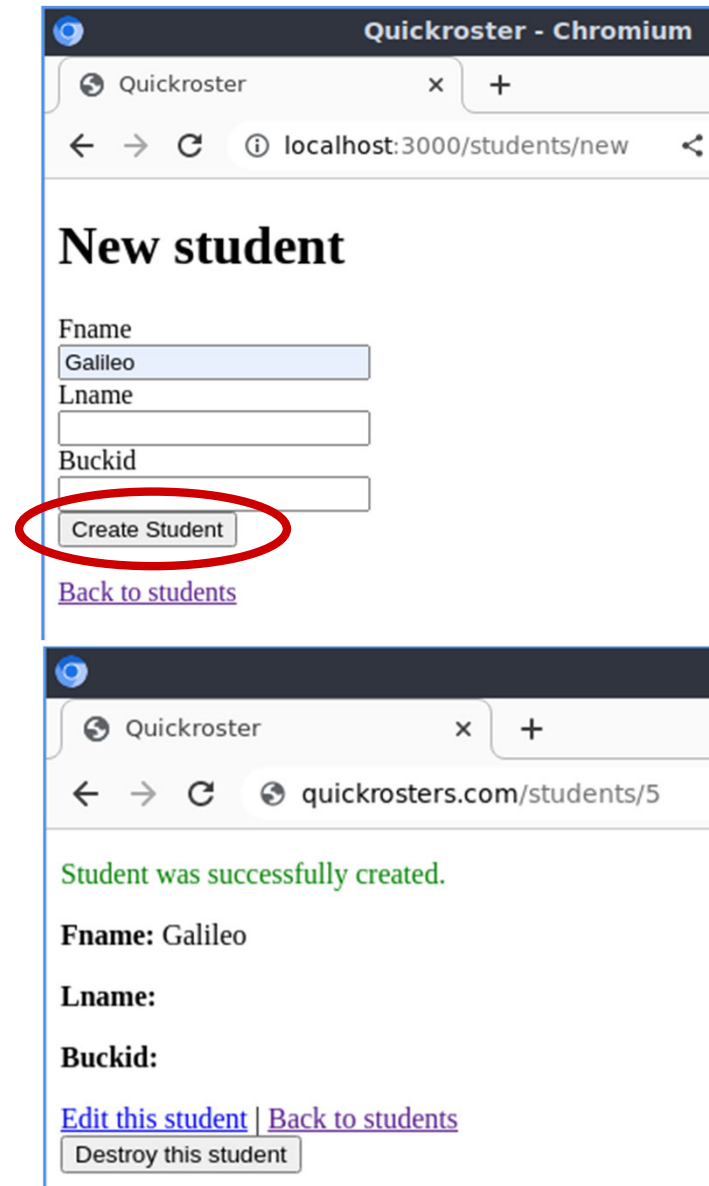
[Back to students](#)

POST /students
lname: ...etc



?

GET Blank Form, POST the Form (3)



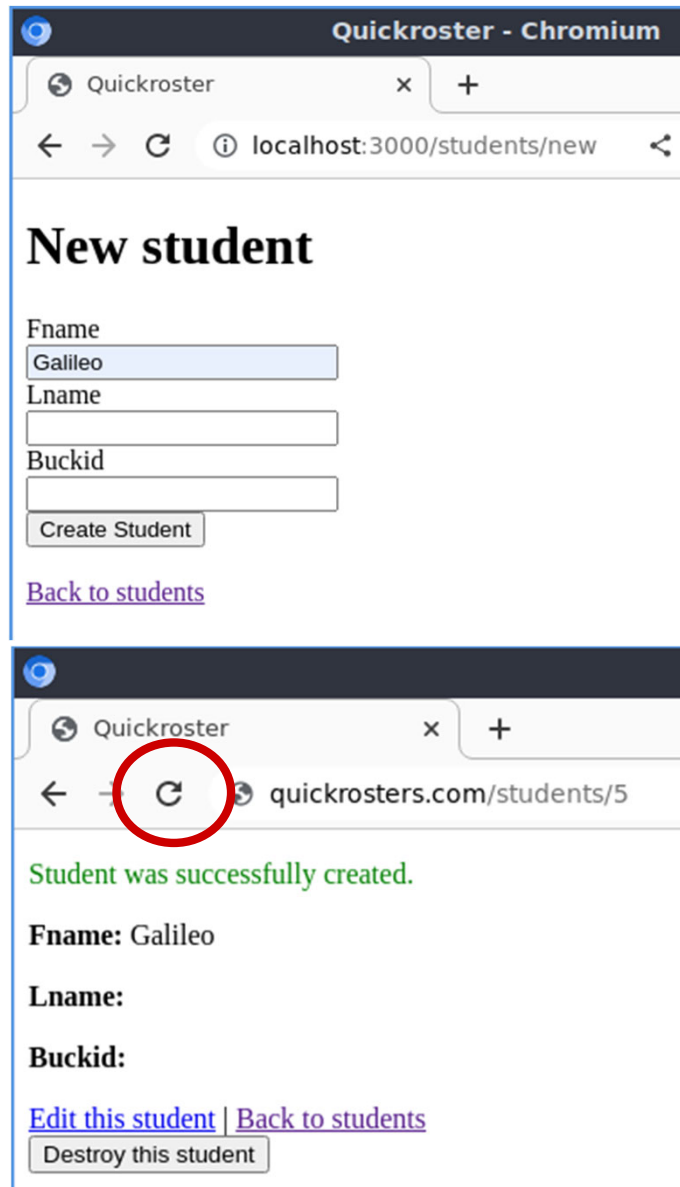
The top screenshot shows a web browser window titled 'Quickroster - Chromium' with the address bar displaying 'localhost:3000/students/new'. The page title is 'New student'. It contains a form with three input fields: 'Fname' (containing 'Galileo'), 'Lname' (empty), and 'Buckid' (empty). Below the fields is a button labeled 'Create Student', which is circled in red. A link 'Back to students' is at the bottom.

The bottom screenshot shows the same browser window after submission, with the address bar displaying 'quickrosters.com/students/5'. The page displays a green message: 'Student was successfully created.' Below this, the student details are listed: 'Fname: Galileo', 'Lname:', and 'Buckid:'. At the bottom, there are two links: 'Edit this student' and 'Back to students', and a button labeled 'Destroy this student'.

POST /students
lname: ...etc



GET Blank Form, POST the Form (4)

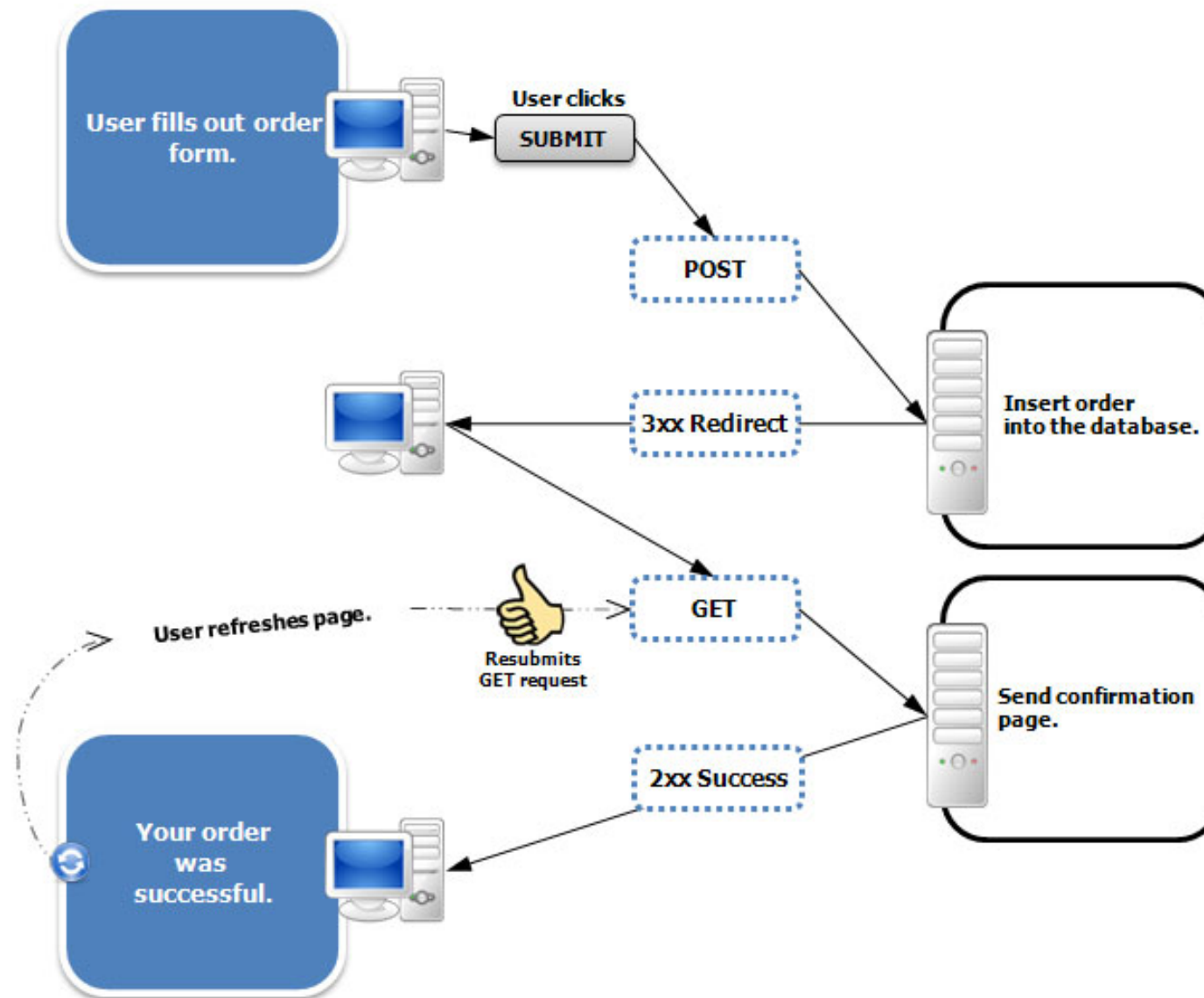


The image shows two screenshots of a web browser. The top screenshot displays the 'New student' form with fields for 'Fname' (containing 'Galileo'), 'Lname', and 'Buckid'. A 'Create Student' button is at the bottom. The bottom screenshot shows the confirmation page with the message 'Student was successfully created.' and displays the student details: 'Fname: Galileo', 'Lname:', and 'Buckid:'. There are links for 'Edit this student' and 'Back to students', and a 'Destroy this student' button. In the bottom screenshot, the refresh button in the browser's navigation bar is circled in red.

POST /students
lname: ...etc



POST-Redirect-GET Pattern



Example of POST-Redirect-GET

```
class BooksController <
  ApplicationController

  def create
    @book = Book.new(book_params)
    if @book.save
      redirect_to @book, # calls url_for()
      notice: 'Book successfully created'
    else
      render :new
    end
  end
end
```

Example of POST-Redirect-GET: Notice

```
class BooksController <
  ApplicationController

  def create
    @book = Book.new(book_params)
    if @book.save
      redirect_to @book,
        notice: 'Book successfully created'
    else
      render :new
    end
  end
end
```

Flash

- A hash returned with redirect response
 - Set by controller action issuing redirect

```
flash[:referral_code] = 1234
```
 - Common keys can be assigned in redirect

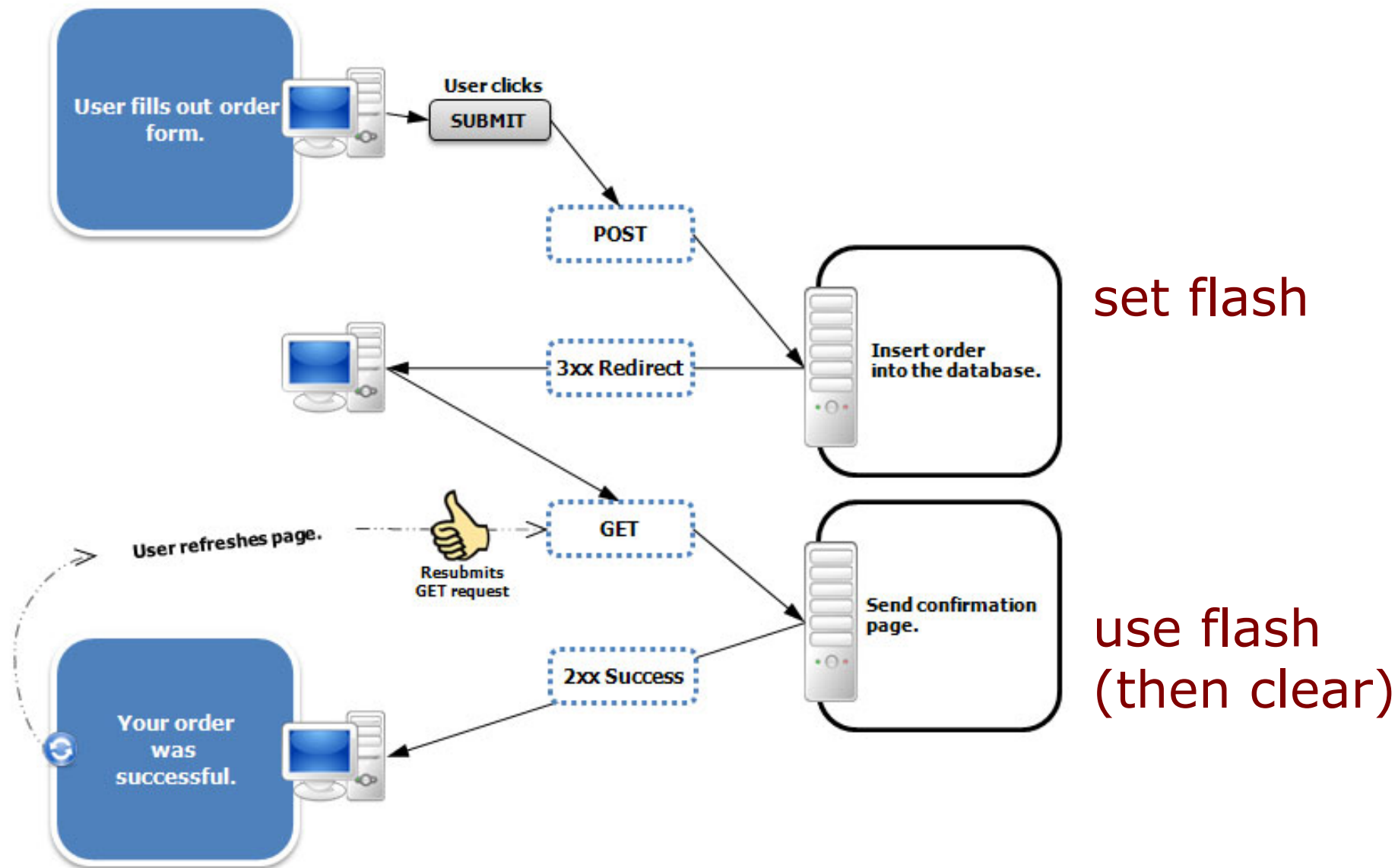
```
redirect_to @book notice: '...'
redirect_to books_path alert: '...'
```
- Flash included in client's *next* request
- Flash available to *next* action's view!

```
<p id="info"><%= flash[:warn] %>...
```

 - But: `flash.now` available to first view!

```
flash.now[:notice] = 'no such book'
```

Flash: Set, Use, Clear



Using Flash in View

```
# display just notice message
```

```
<p id="notice"><%= notice %></p>
```

```
# display all the flash messages
```

```
<% if flash.any? %>
```

```
  <div id="banner">
```

```
    <% flash.each do |key, message| %>
```

```
      <div class="flash <%= key %>">
```

```
        <%= message %>
```

```
      </div>
```

```
    <% end %>
```

```
  </div>
```

```
<% end %>
```

Example of Render vs Redirect

```
class BooksController <
  ApplicationController

def update
  @book = Book.find(params[:id])
  if @book.update(book_params)
    redirect_to @book,
      notice: 'Book successfully updated'
  else
    render :edit
  end
end
```

Why Is This Wrong?

```
class BooksController <
  ApplicationController

def update
  @book = Book.find(params[:id])
  if @book.update(book_params)
    redirect_to @book,
      notice: 'Book successfully updated'
  else
    render :edit,
      notice: 'Try again.'
  end
end
```

Fix: Use Flash.now

```
class BooksController <
  ApplicationController

def update
  @book = Book.find(params[:id])
  if @book.update(book_params)
    redirect_to @book,
      notice: 'Book successfully updated'
  else
    flash.now[:notice] = 'Try again.'
    render :edit
  end
end
```

Code Duplication

```
class BooksController < ApplicationController

  def show
    @book = Book.find(params[:id])
  end

  def edit
    @book = Book.find(params[:id])
  end

  def update
    @book = Book.find(params[:id])
    . . .
  end
```

DRY, aka Single-Point-of-Control

```
class BooksController < ApplicationController
  before_action :set_book,
                only %i[ show edit update destroy ]

  def show  # method is now empty!
  end

  def edit  # method is now empty!
  end

  # and other actions...

  private
    def set_book
      @book = Book.find(params[:id])
    end
end
```

Sanatizing Inputs

```
def update
  if @book.update(book_params)
    redirect_to @book, notice: 'Success!'
  else
    render :edit
  end
end
```

```
private
def set_book
  @book = Book.find(params[:id])
end

def book_params
  params.require(:book).permit(:title, :summary)
end
```

Recall Partial

- A blob of ERb used in multiple views
- Examples
 - Static header used throughout site
 - Dynamic sidebar used in many places
- Include in a template (or layout) with:
 - `<%= render 'menu' %>`
 - `<%= render 'users/icon' %>`
- Filename of partial has "_" prefix
 - Default location: app/views
`app/views/_menu.html.erb`
 - Organize into subdirectories with good names
`app/views/users/_icon.html.erb`

Example: views/layouts/applic...

```
<!DOCTYPE html>
<html>
  ... etc
<body>
  <%= render 'layouts/header' %>
  <div class="container">
    <%= yield %>
    <%= render 'layouts/footer' %>
  </div>
</body>
</html>
```

Example: views/layouts/_footer

```
<footer class="footer">
  <small>
    <a href="http://www.osu.edu">OSU</a>
  </small>
  <nav>
    <ul>
      <li><%= link_to "About",
                    about_path %></li>
      <li><%= link_to "Contact",
                    contact_path %></li>
    </ul>
  </nav>
</footer>
```

Recall: Tricks with Partials

- Content of partial can be customized with arguments in call

- In call: pass a hash called `:locals`

```
<%= render partial: "banner",  
      locals: { name: "Syllabus",  
                amount: @price } %>
```

- In partial: access hash with *variables*

```
<h3> <%= name %> </h3>  
<p> Costs <%= "$#{amount}.00" %></p>
```

Parameter Passing to Partials

- Partial also has one *implicit* local variable
- In the partial, parameter name *same* as partial

```
# in partial nav/_menu.html
```

```
<p> The price is: <%= menu %></p>
```

- Argument value assigned explicitly

```
<%= render partial: 'nav/menu',  
      object: cost %>
```

- Idiom: Begin partial by renaming this parameter

```
# in partial nav/_menu.html
```

```
<% price = menu %>
```

Example: books/show.html.erb

```
<p style="color: green"><%= notice %></p>
```

```
<%= render @book %>
```

```
<div>
```

```
  <%= link_to "Edit this book",  
            edit_book_path(@book) %> |
```

```
  <%= link_to "Back to books", books_path %>
```

```
  <%= button_to "Destroy this book", @book,  
              method: :delete %>
```

```
</div>
```

Partial: books/_book.html.erb

```
<div id="<%= dom_id book %>">
  <p>
    <strong>Title:</strong>
    <%= book.title %>
  </p>

  <p>
    <strong>Author:</strong>
    <%= book.author %>
  </p>
</div>
```

Demo: Scaffolding

- Generate many things at once
 - Migration for table in database
 - Model for resource
 - RESTful routes
 - Controller and corresponding methods
 - Views for responses
- Command

```
$ rails g scaffold Student lname:string buckid:integer
$ rails db:migrate
$ rails server
```

Summary: Rendering, Scaffolding

Computer Science and Engineering ■ The Ohio State University

- Controller generates a response
 - Default: render corresponding view
 - Explicit: `render` some action's view
 - Explicit: `re-direct`
 - POST-redirect-GET (aka “get after post”)
 - Flash passes information to next action
- Reuse of views with partials
 - Included with `render` (e.g., `<%= render...>`)
 - Filename is prepended with underscore
 - Parameter passing from parent template
 - Can iterate over partial by iterating over a collection