Update Action (TimeEntry)

The common way to implement an Update

1. A user visits a page that contains a form with a resource loaded

2. The user fills the form and submits it

3. An action receives that information and processes it:

    1. if it's valid, then the entity is updated

    2. if it's invalid, then renders the template with the form and some validation errors

1. A user visits a page that contains a form with a resource loaded

Http Method: **GET**

URL path: **/<resource>/<id>/edit**

HTML template: **<form ...>**

2. The user fills the form and submits it

- Nothing we can do in the backend -

3. An action receives that information a process it

Http Method: **PUT / PATCH**

URL path: **/<resource>/<id>**

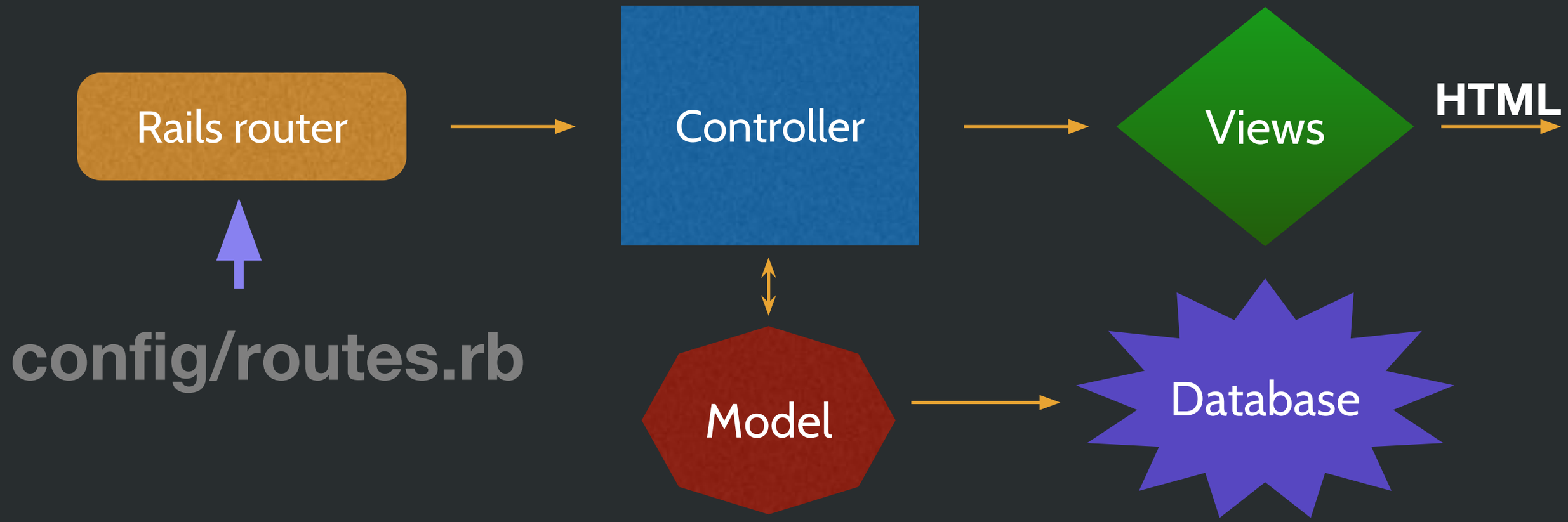
HTML template: **(no template)**

Implementing the “form” action

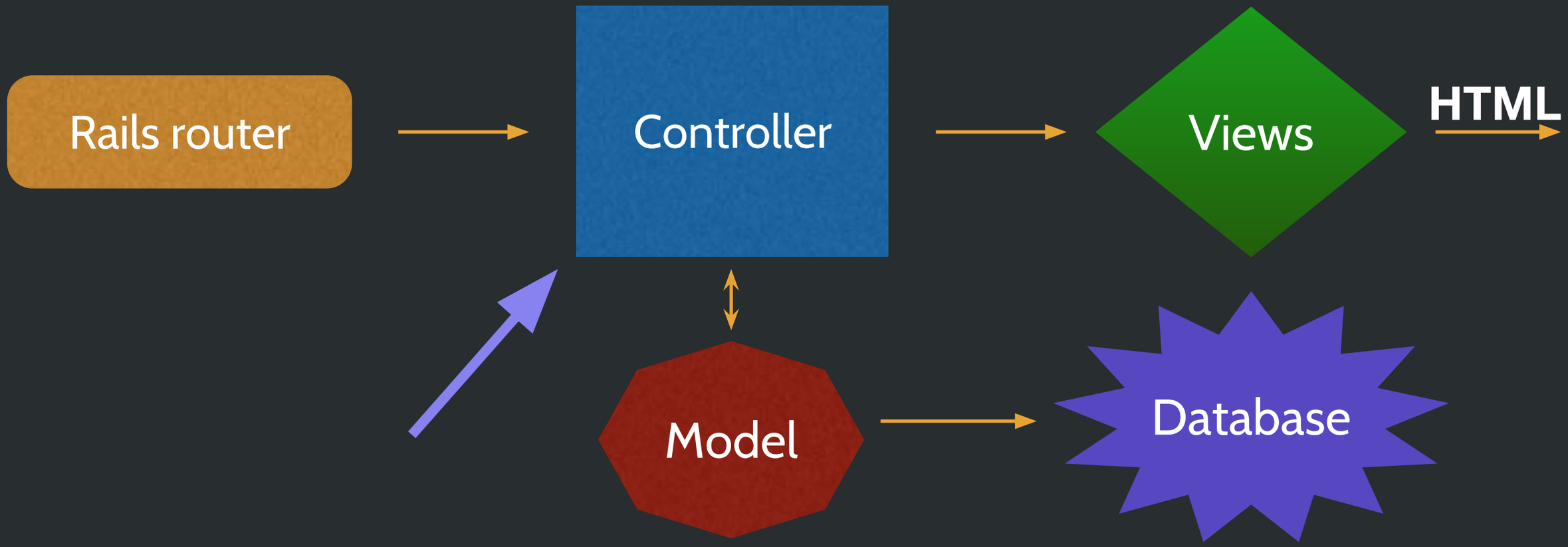
Implementing the “form” action

GET

/projects/:project_id/entries/3/edit

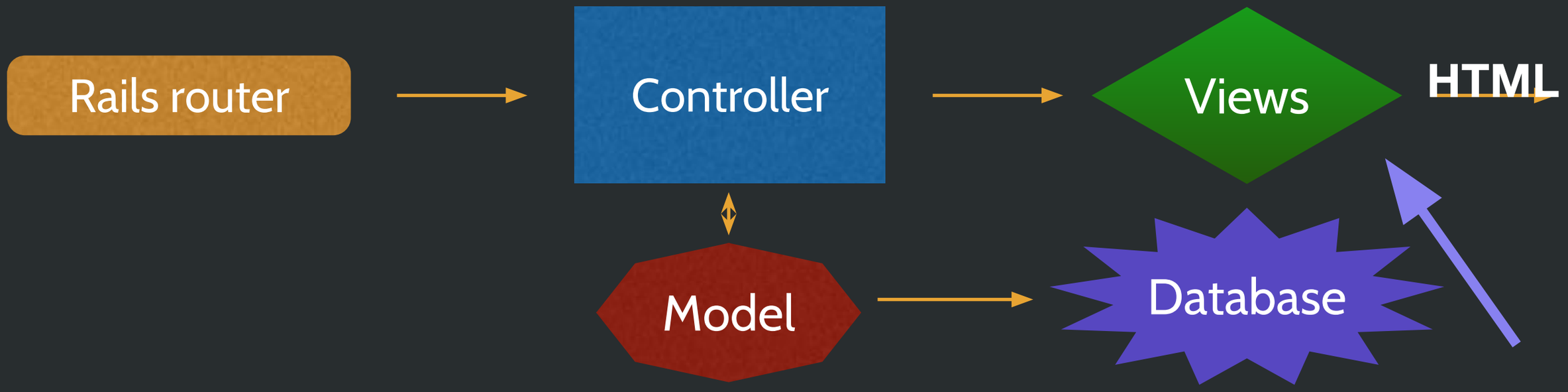



```
Timetracking::Application.routes.draw do
  # [...]
  get '/projects/:project_id/time_entries/:id/edit', to: 'time_entries#edit'
end
```



app/controllers/time_entries_controller.rb

```ruby
class TimeEntriesController < ApplicationController
  def edit
    @my_project = Project.find params[:project_id]
    @my_entry = @my_project.time_entries.find params[:id]
  end
end
```



app/views/time_entries/edit.html.erb

```erb
<%= form_for [@my_project, @my_entry] do |f| %>
  <%= f.label :hours %>
  <%= f.text_field :hours, size: 4 %>
  <br>
  <%= f.label :minutes %>
  <%= f.text_field :minutes, size: 4 %>
  <br>
  <%= f.label :date %>
  <%= f.date_field :date, size: 4 %>
  <br>
  <%= f.label :comments %>
  <%= f.text_area :comments %>
  <br>
  <%= f.submit 'Save' %>
  <%= link_to 'Cancel', project_time_entries_path(@my_project) %>
<% end %>
```

Time tracking tool

—

Hours 1

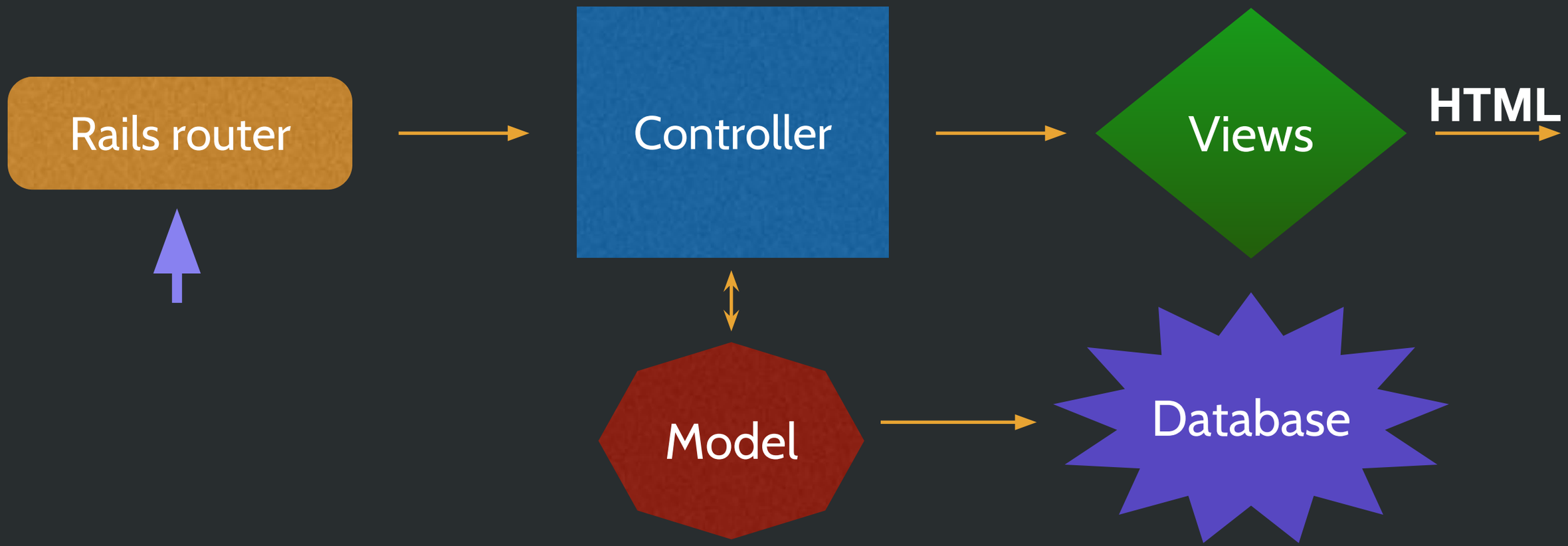
Minutes 24

Date 14/02/2014

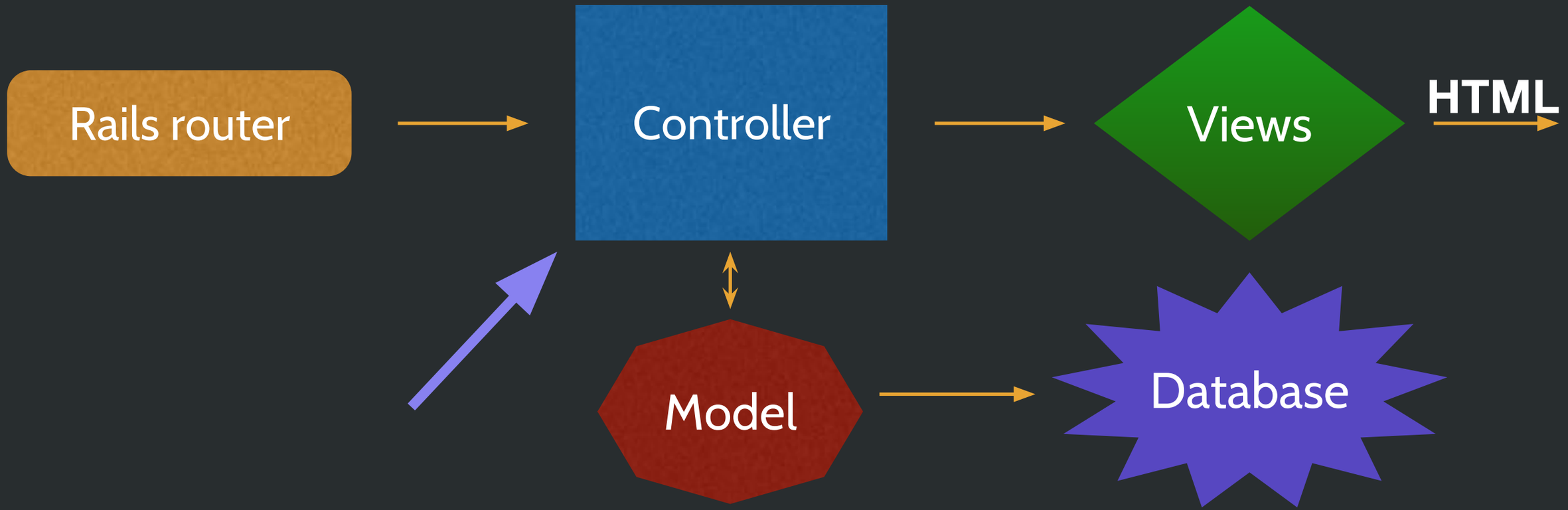
Save Cancel

Unknown action

The action 'update' could not be found for EntriesController




```
Timetracking::Application.routes.draw do
    # [...]
    get '/projects/:project_id/time_entries/:id/edit', to: 'time_entries#edit'
    patch '/projects/:project_id/time_entries/:id', to: 'time_entries#update'
end
```



app/controllers/time_entries_controller.rb

```ruby
class TimeEntriesController < ApplicationController
  def update
    @my_project = Project.find_by(id: params[:project_id]
    @my_entry = @my_project.time_entries.find_by(id: params[:id])

    if @my_entry.update(hours: params[:time_entry][:hours],
                        minutes: params[:time_entry][:minutes],
                        date: params[:time_entry][:date])
      redirect_to action: "index", controller: "time_entries",
project_id: @my_project.id
    else
      render "edit"
    end
  end
end
```

```ruby
class TimeEntriesController < ApplicationController
  def update
    @my_project = Project.find_by(id: params[:project_id]
    @my_entry = @my_project.time_entries.find_by(id: params[:id])

    if @my_entry.update(hours: params[:time_entry][:hours],
                        minutes: params[:time_entry][:minutes],
                        date: params[:time_entry][:date])
      redirect_to action: "index", controller: "time_entries",
project_id: @my_project.id
    else
      render "edit"
    end
  end
end
```

```ruby
class TimeEntriesController < ApplicationController
  def update
    @my_project = Project.find_by(id: params[:project_id]
    @my_entry = @my_project.time_entries.find_by(id: params[:id])

    if @my_entry.update(hours: params[:time_entry][:hours],
                        minutes: params[:time_entry][:minutes],
                        date: params[:time_entry][:date])
      redirect_to action: "index", controller: "time_entries",
project_id: @my_project.id
    else
      render "edit"
    end
  end
end
```

We've seen this before in our create action

Flashback to create

```
@my_entry = @my_project.time_entries.new(
    hours: params[:time_entry][:hours],
    minutes: params[:time_entry][:minutes],
    date: params[:time_entry][:date])
```

This isn’t DRY (Don’t repeat yourself)

Rails provides us a nicer syntax, and a safer way to do this.

It’s called strong parameters.

Strong Parameters

Strong parameters are a way to keep your Rails app safe

Strong Parameters

Coincidentally they also make your code easier to read.

Strong Parameters

Let's see them in action

Strong Parameters

Say we submit our edit / new form

Understanding mass assignments

Params

```
"time_entry"=>
 {"hours" =>"3",
  "minutes" => "45",
  "date" => "2014-02-06"},
"project_id" => 2
```

Understanding Mass Assigments

We *could* create a new entry like this:

```
@my_entry = @my_project.time_entries.new(params[:time_entry])
```

Understanding Mass Assigments

ActiveModel::ForbiddenAttributesError in EntriesController#create

ActiveModel::ForbiddenAttributesError

Extracted source (around line **#14**):

```
12    def create
13      @project = Project.find params[:project_id]
14      @entry = @project.entries.new params[:entry]
15      if @entry.save
16        redirect_to action: 'index', controller: 'entries', project_id: @project.id
17      else
```

`Rails.root: /Users/fernando/proyectos/ironhack/timetracking`

Application Trace | Framework Trace | Full Trace

`app/controllers/entries_controller.rb:14:in `create'`

Request

Understanding Mass Assigments

Rails says that we explicitly have to tell it what is allowed into a controller and what is not

Strong Parameters

Why? Mass Assignment

Strong Parameters

If we don't explicitly specify what is allowed into our controller a user could enter anything

Strong Parameters

You can easily add inputs in the chrome inspector that *WILL* be submitted to the server

Strong Parameters

```
<label for="entry_date">Date</label>
<input value="2016-03-09" type="date" name="entry[date]" id="entry_date">
<br>
<input value="HACKED!" type="text" name="entry[some_secret_field]">
<input type="submit" name="commit" value="Save">
<a href="/projects/1/entries">Cancel</a>
</form>
```

Strong Parameters

```
<label for="entry_date">Date</label>
<input value="2016-03-09" type="date" name="entry[date]" id="entry_date">
<br>
<input value="HACKED!" type="text" name="entry[some_secret_field]">
<input type="submit" name="commit" value="Save">
<a href="/projects/1/entries">Cancel</a>
</form>
```

Strong Parameters

```
"time_entry"=>
 {"hours" =>"3",
  "minutes" => "45",
   "date" => "2014-02-06"
   "some_secret_field" => "HACKED!"},
"project_id" => 2
```

Strong Parameters

At best, this can cause an error. Entry doesn't have a field called `some_secret_field`.

Strong Parameters

At worst this could allow some hacker complete access to your website.

Strong Parameters

Be safe! Be DRY! Let's implement strong params.

Strong Parameters

```ruby
class TimeEntriesController < ApplicationController
  #[...]
  private

  def entry_params
    params.require(:time_entry).permit(:hours, :minutes, :date)
  end
end
```

Strong Parameters

What's going on?

Strong Parameters

Our params must have a key called entry.

Strong Parameters

```ruby
class TimeEntriesController < ApplicationController
  #[...]
  private
  def entry_params
    params.require(:time_entry).permit(:hours, :minutes, :date)
  end
end
```

Strong Parameters

```
"time_entry"=>
 {"hours" =>"3",
  "minutes" => "45",
  "date" => "2014-02-06"
  "some_secret_field" => "HACKED!"},
"project_id" => 2
```

Strong Parameters

And I will only permit the fields of hours, minutes, and date.

Strong Parameters

```
"time_entry"=>
 {"hours" =>"3",
  "minutes" => "45",
  "date" => "2014-02-06"
   "some_secret_field" => "HACKED!"},
"project_id" => 2
```

Strong Parameters

```
"time_entry"=>
 {"hours" =>"3",
  "minutes" => "45",
  "date" => "2014-02-06"
  "some_secret_field" => "HACKED!"},
"project_id" => 2
```

Strong Parameters

This method now returns our params hash sanitized and clear of any excess information

Update

```
class TimeEntriesController < ApplicationController
  def update
    @my_project = Project.find_by(id: params[:project_id])
    @my_entry = @my_project.time_entries.find_by(id: params[:id])
    if @my_entry.update(entry_params)
      redirect_to action: "index", controller: "time_entries",
project_id: @my_project.id
    else
      render "edit"
    end
  end
end
```

Create

```
class TimeEntriesController < ApplicationController
  def create
    @my_project = Project.find params[:project_id]
    @my_entry = @my_project.time_entries.new(entry_params)
    if @my_entry.save
      redirect_to action: "index", controller: "time_entries",
project_id: @my_project.id
    else
      render 'new'
    end
  end
end
```

We're not totally DRY yet!

Where else are we repeating code?

We have the same form in both our new and edit views.

Let's put that in a partial called `_form.html.erb`, and then use that in both of our views.

```erb
<% if @my_entry.errors.any? %>
  <% if flash[:error] %>
    <%= flash[:error] %>
  <% end %>
  <ul>
    <% @my_entry.errors.full_messages.each do |error_msg| %>
      <li><%= error_msg %></li>
    <% end %>
  </ul>
<% end %>
<%= form_for [@my_project, @my_entry] do |f| %>
    <%= f.label :hours %>
    <%= f.text_field :hours %>
    <br>
    <%= f.label :minutes %>
    <%= f.text_field :minutes %>
    <br>
    <%= f.label :date %>
    <%= f.date_field :date %>
    <br>
    <%= f.submit "Save" %>
    <%= link_to("Cancel", project_time_entries_path) %>
<% end %>
```

app/views/time_entries/_form.html.erb

app/views/time_entries/edit.html.erb

```
<h2> Edit time entry for <%= @my_project.name %> </h2>

<%= render "form" %>
```

app/views/time_entries/new.html.erb

```
<h2>Creating a new time entry in <%= @my_project.name %> project </h2>

<%= render "form" %>
```

Any time you see code being repeated in your view, extract it into a partial.

Rails figures out if `@my_entry` persisted (saved in the database) and then determines what the forms method will be. (post or patch/put)

Exercise

In the time entries index, add a link to each entry to go to it's edit page.

Remember to commit your changes in git