COM 1001

Introduction to Software Engineering

Professor Phil McMinn

Forms and Models

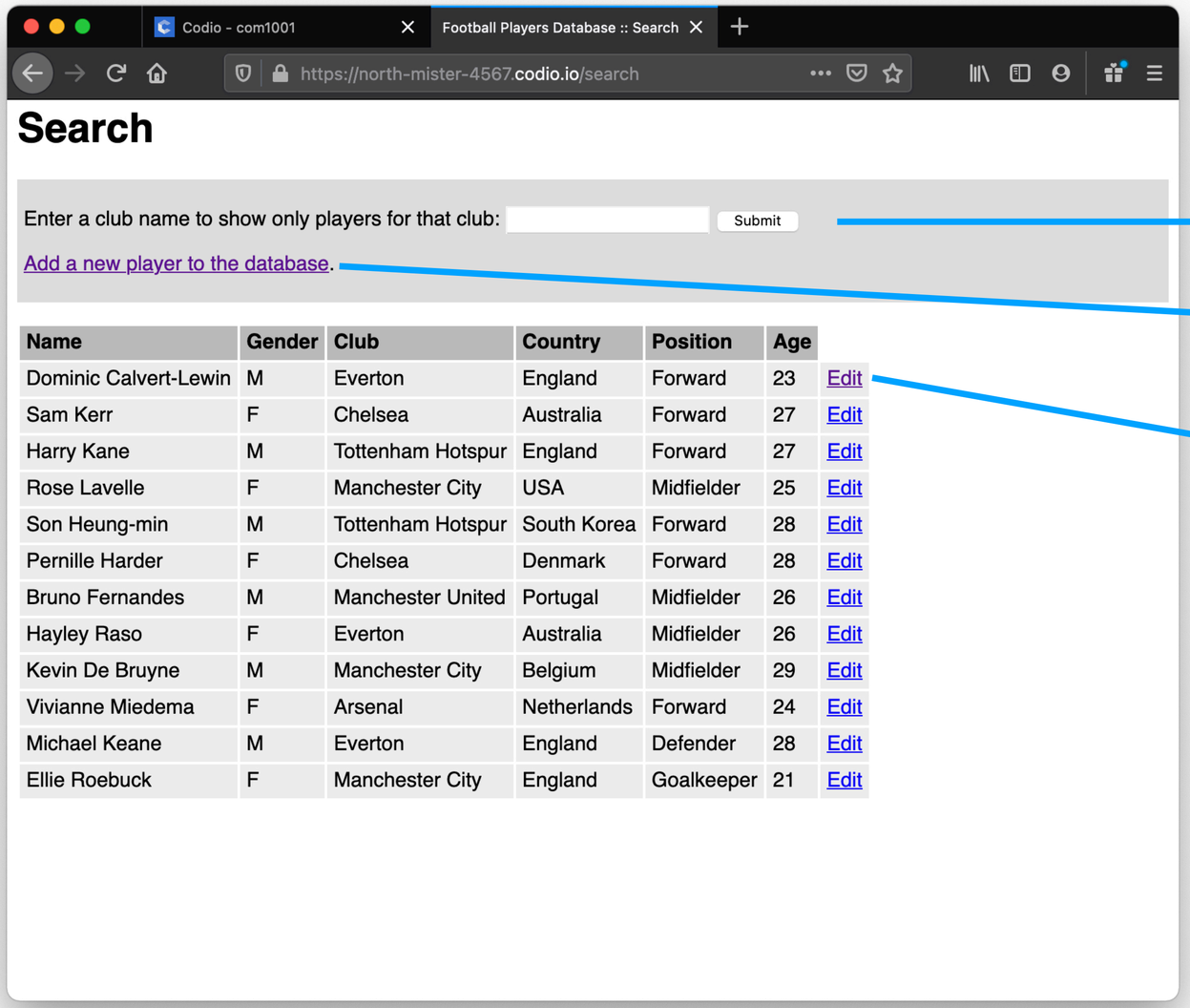
Forms and Models

With web applications, forms often provide a means to manipulate database data – that is, adding, updating, and deleting records.

Sequel models provide functionality for easily doing this, including a means for validating values before they go into the database.

As usual, we will do this by example, by means of extending the Football Players application. See `forms/football_players` in the code repository.

A Tour – Starting with the Search Page

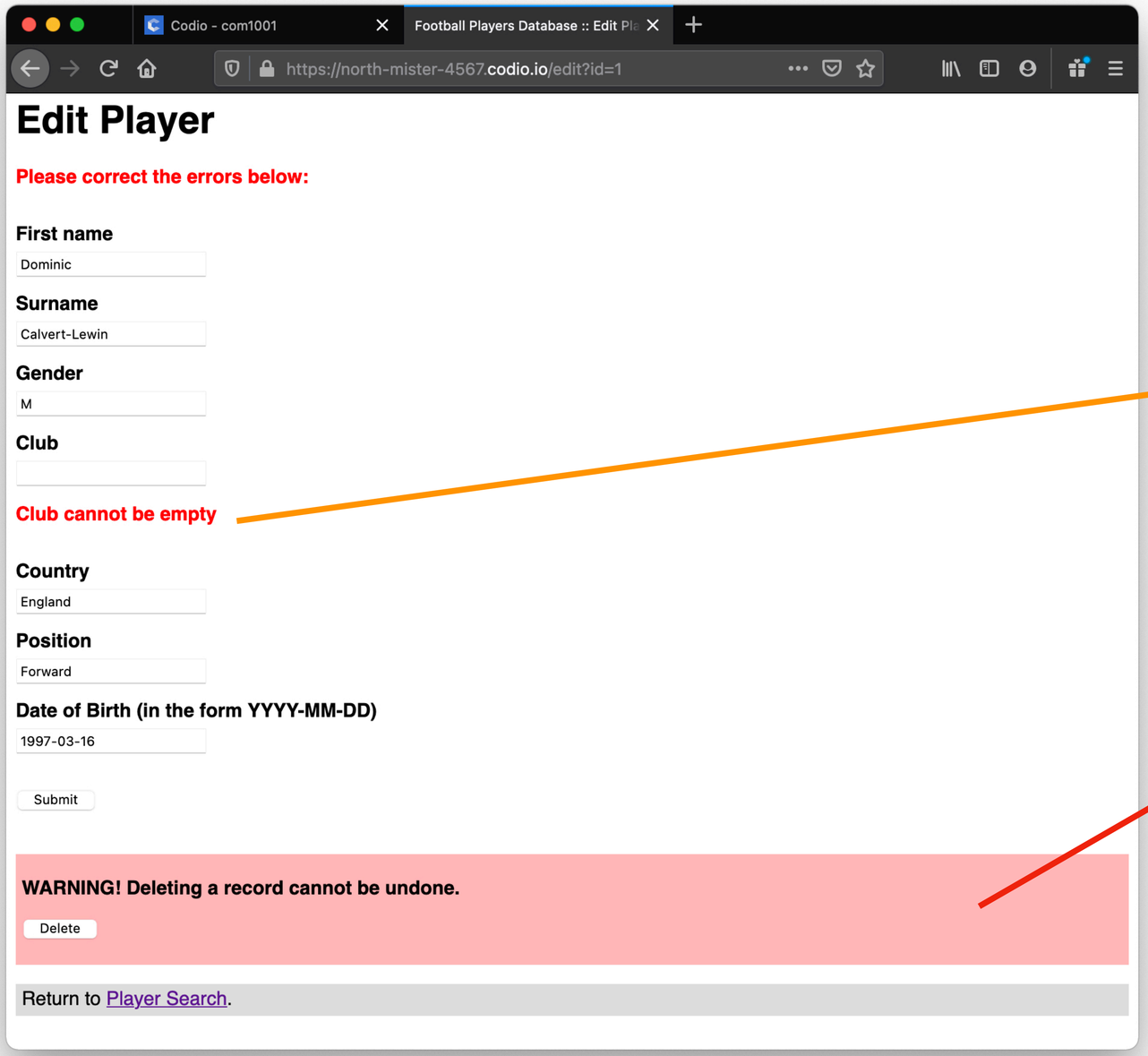


Name	Gender	Club	Country	Position	Age	
Dominic Calvert-Lewin	M	Everton	England	Forward	23	Edit
Sam Kerr	F	Chelsea	Australia	Forward	27	Edit
Harry Kane	M	Tottenham Hotspur	England	Forward	27	Edit
Rose Lavelle	F	Manchester City	USA	Midfielder	25	Edit
Son Heung-min	M	Tottenham Hotspur	South Korea	Forward	28	Edit
Pernille Harder	F	Chelsea	Denmark	Forward	28	Edit
Bruno Fernandes	M	Manchester United	Portugal	Midfielder	26	Edit
Hayley Raso	F	Everton	Australia	Midfielder	26	Edit
Kevin De Bruyne	M	Manchester City	Belgium	Midfielder	29	Edit
Vivianne Miedema	F	Arsenal	Netherlands	Forward	24	Edit
Michael Keane	M	Everton	England	Defender	28	Edit
Ellie Roebuck	F	Manchester City	England	Goalkeeper	21	Edit

The **search** page is the main page of the app, from which we can search for players by club – which limits the list below.

New players can be **added** by clicking this link

Existing players can be **edited** by clicking on the row of the player name

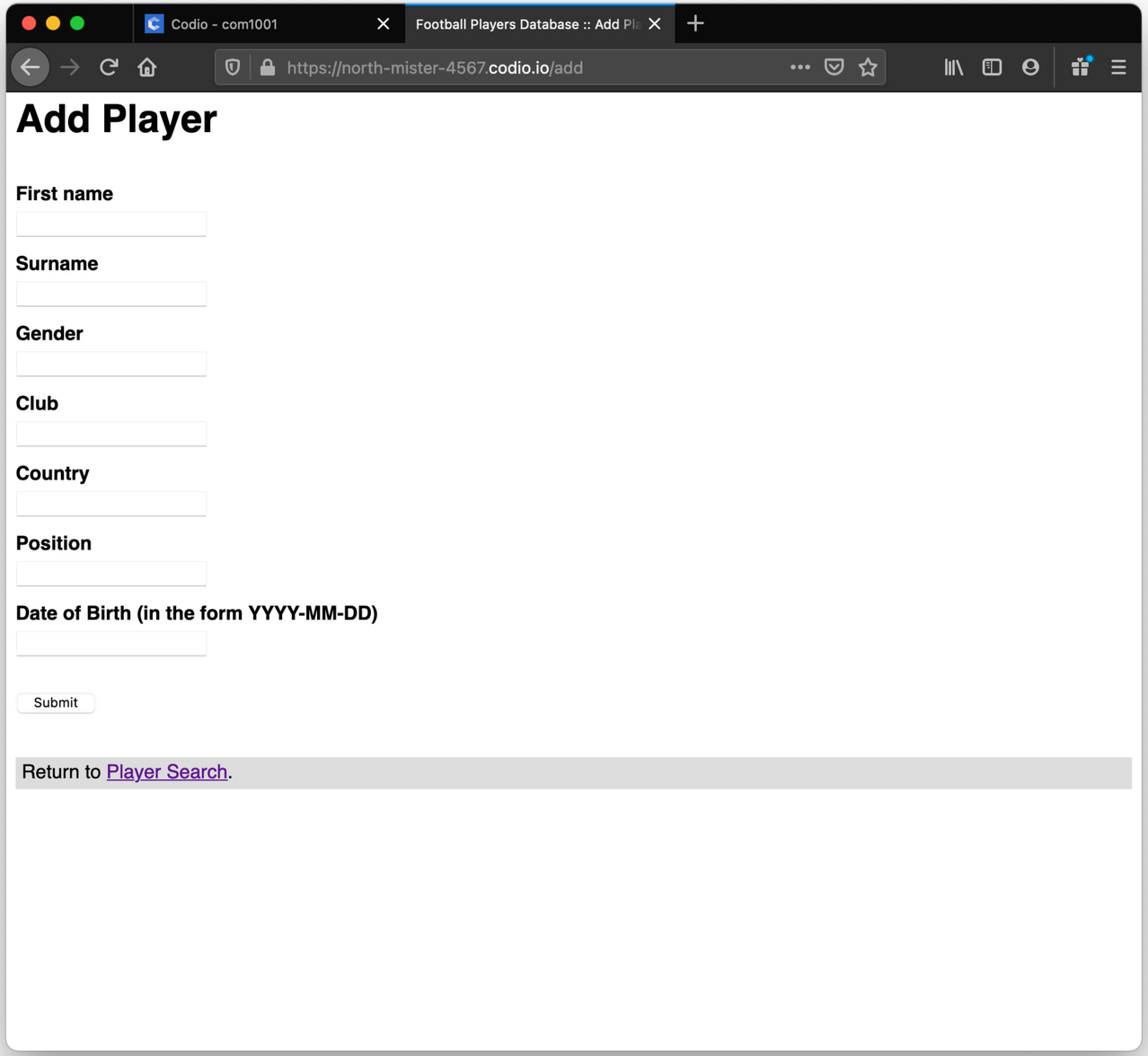


The Edit Page

The **edit** page loads the player record, and allows the user to edit it.

The page **validates** the form fields.

Players can also be **deleted** from this page.

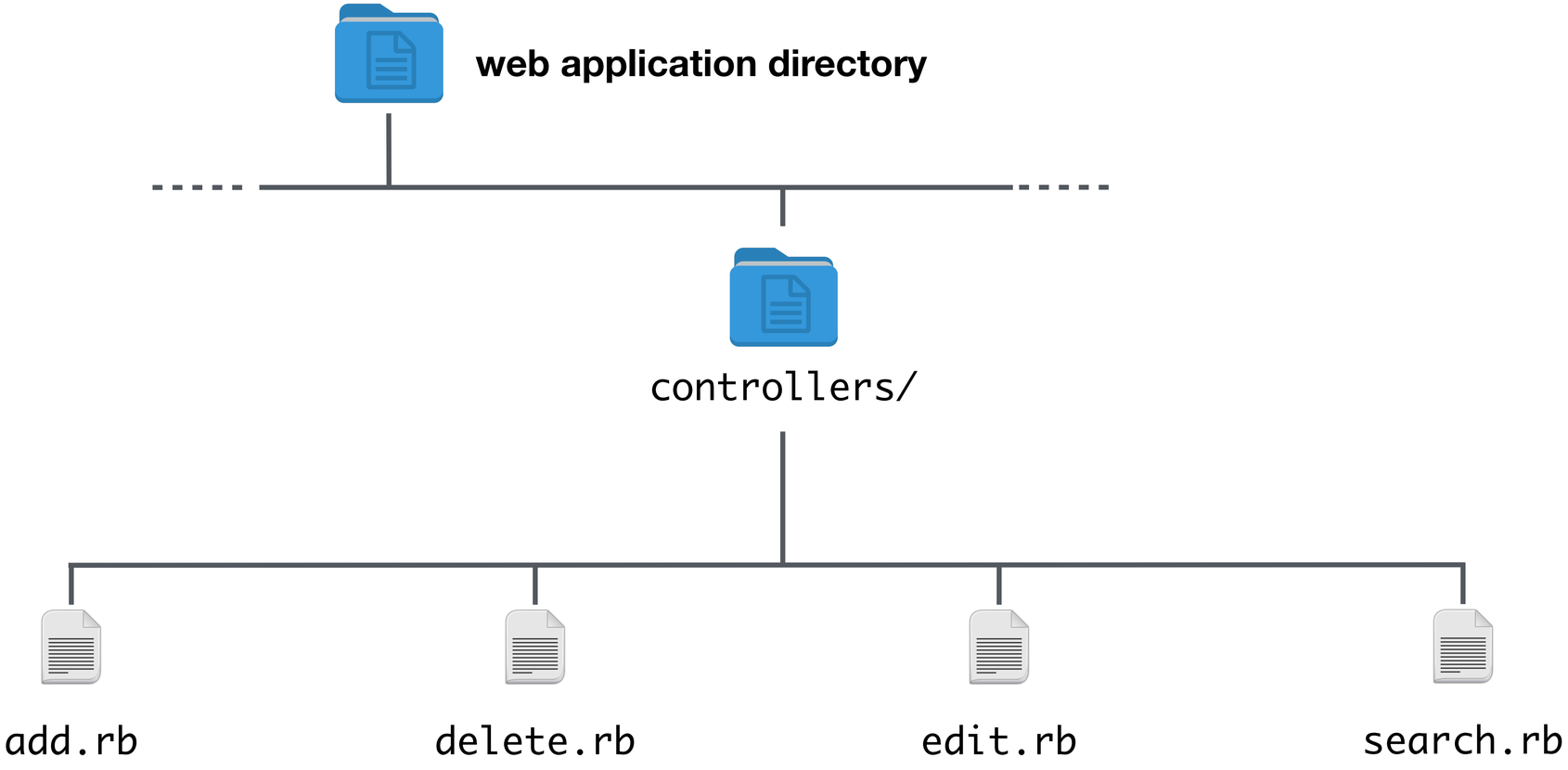


The Add Page

The **add** page is not too dissimilar from the edit page (it re-uses components of the view), except it does not load a record to edit.

The page validates the form fields, like the edit page.

Controllers



The add.rb Controller

```
get "/add" do
  @player = Player.new
  erb :add
end

post "/add" do
  @player = Player.new
  @player.load(params)

  if @player.valid?
    @player.save_changes
    redirect "/search"
  end

  erb :add
end
```

The code for the controllers in this example is relatively simple (as it should be), because the model does the heavy lifting.

The form in the add page uses the `post` method to submit the form. We do not want the data to be present in the URL and adding a new player to the database is a "one-time" action for that player.

forms/football_players/controllers/add.rb

The add.rb Controller

```ruby
get "/add" do
  @player = Player.new
  erb :add
end

post "/add" do
  @player = Player.new
  @player.load(params)

  if @player.valid?
    @player.save_changes
    redirect "/search"
  end

  erb :add
end
```

The `load` method of the `Player` class takes the params hash and sets the field of the object. It also sanitises the data (by removing leading and trailing whitespace through calls to `strip`)

```ruby
def load(params)
  self.first_name = params.fetch("first_name", "").strip
  self.surname = params.fetch("surname", "").strip
  self.gender = params.fetch("gender", "").strip
  self.club = params.fetch("club", "").strip
  self.country = params.fetch("country", "").strip
  self.position = params.fetch("position", "").strip
  self.date_of_birth = params.fetch("date_of_birth", "").strip
end
```

forms/football_players/controllers/add.rb

forms/football_players/models/player.rb

The add.rb Controller

```ruby
get "/add" do
  @player = Player.new
  erb :add
end

post "/add" do
  @player = Player.new
  @player.load(params)

  if @player.valid?
    @player.save_changes
    redirect "/search"
  end

  erb :add
end
```

The `valid?` method of the `Player` class is inherited from `Sequel::Model`. It calls a method, `validate`, that we need to define ourselves. It goes through the fields and adds messages to an instance field of Player, `errors`. These messages can then be used in the view, to report the problems to the end-user.

```ruby
def validate
  super
  errors.add("first_name", "cannot be empty") if first_name.empty?
  errors.add("surname", "cannot be empty") if surname.empty?
  errors.add("gender", "cannot be empty") if gender.empty?

  ...
```

forms/football_players/controllers/add.rb

forms/football_players/models/player.rb

The View

```
<p class="label">First name</p>
<p class="field"><input type="text" name="first_name" value="<%= h @player.first_name %>" /></p>
<% if @player.errors.include?("first_name") %>
  <% @player.errors["first_name"].each do |error| %>
    <p class="error"><strong>First name <%= error %></strong></p>
  <% end %>
<% end %>

...
```

forms/football_players/views/common/player_form.erb

The view queries the `errors` hash of the `player` instance for each field. If there are errors for the field, it iterates through the array of messages, and adds a message into the HTML for each one

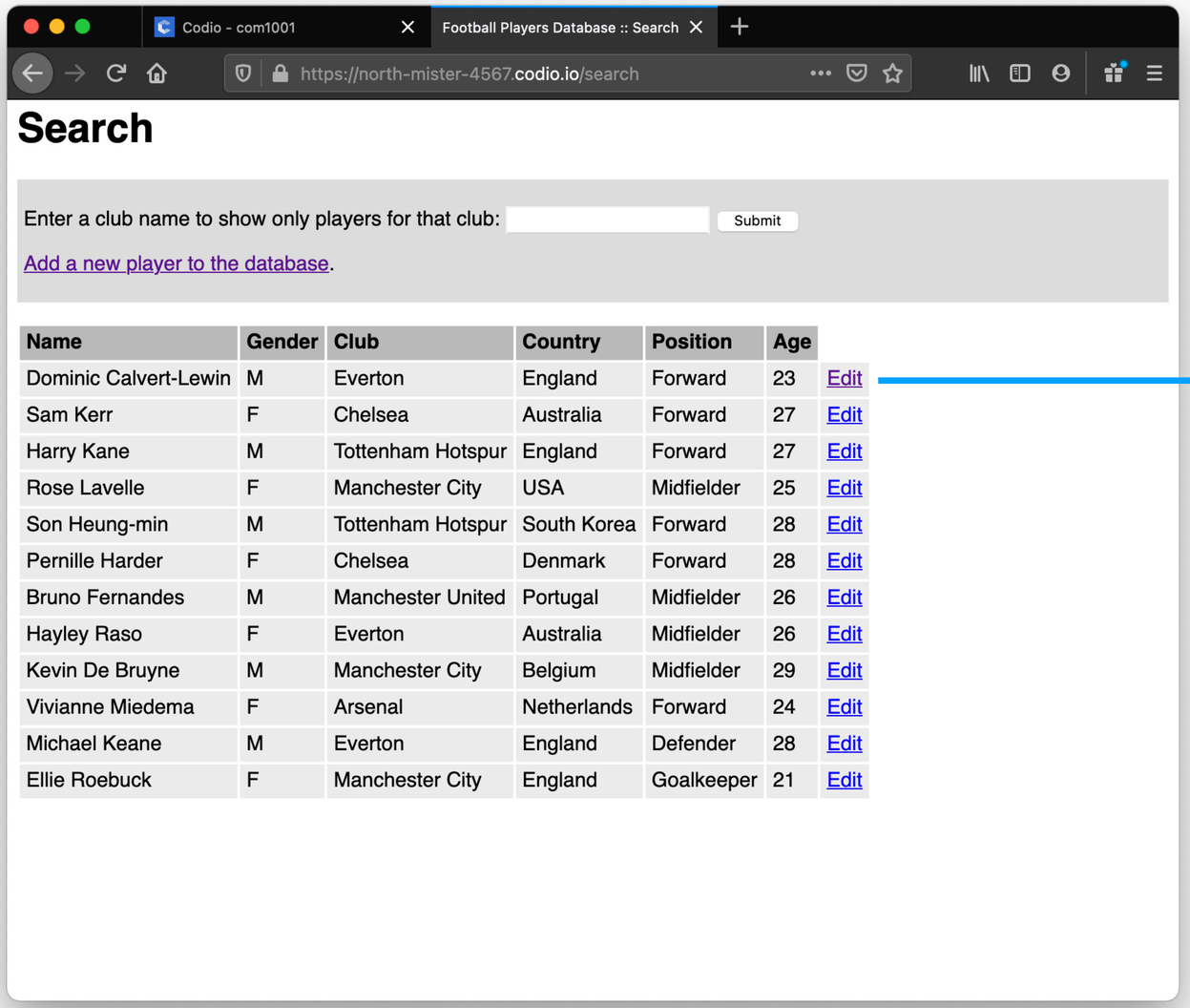
The add.rb Controller

```
get "/add" do
  @player = Player.new
  erb :add
end

post "/add" do
  @player = Player.new
  @player.load(params)

  if @player.valid?
    @player.save_changes
    redirect "/search"
  end

  erb :add
end
```

If the entered data is valid, we save the data in the model to the database, and redirect the browser back to the search page.

forms/football_players/controllers/add.rb

The Search Page



Name	Gender	Club	Country	Position	Age	
Dominic Calvert-Lewin	M	Everton	England	Forward	23	Edit
Sam Kerr	F	Chelsea	Australia	Forward	27	Edit
Harry Kane	M	Tottenham Hotspur	England	Forward	27	Edit
Rose Lavelle	F	Manchester City	USA	Midfielder	25	Edit
Son Heung-min	M	Tottenham Hotspur	South Korea	Forward	28	Edit
Pernille Harder	F	Chelsea	Denmark	Forward	28	Edit
Bruno Fernandes	M	Manchester United	Portugal	Midfielder	26	Edit
Hayley Raso	F	Everton	Australia	Midfielder	26	Edit
Kevin De Bruyne	M	Manchester City	Belgium	Midfielder	29	Edit
Vivianne Miedema	F	Arsenal	Netherlands	Forward	24	Edit
Michael Keane	M	Everton	England	Defender	28	Edit
Ellie Roebuck	F	Manchester City	England	Goalkeeper	21	Edit

This is a link with a querystring

The form is `edit?id=X`, where `X` is the ID of the player. This is so that the edit page knows which player to retrieve to edit.

```ruby
get "/edit" do
  id = params["id"]
  @player = Player[id] if Player.id_exists?(id)
  erb :edit
end

post "/edit" do
  id = params["id"]

  if Player.id_exists?(id)
    @player = Player[id]
    @player.load(params)

    if @player.valid?
      @player.save_changes
      redirect "/search"
    end
  end

  erb :edit
end
```

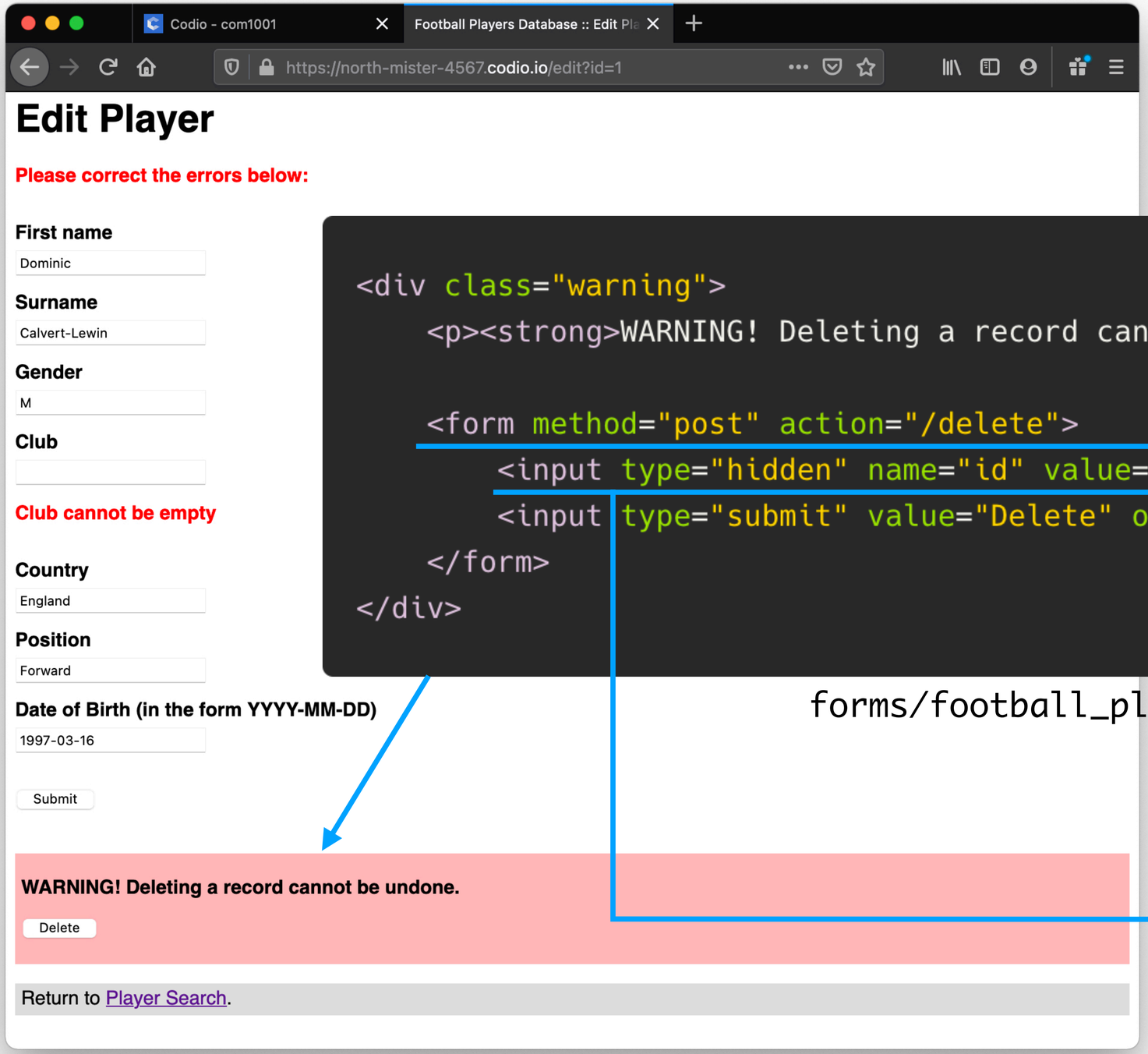
forms/football_players/controllers/edit.rb

The `edit.rb` Controller

The code for the controller of the Edit page is similar to the Add page, except an existing record of the database must be loaded in first.

The search page passes in an `id` through the query string, which the controller loads in, and tries to match against and existing player.

If there is no match, an error message is shown in the view.




```
<div class="warning">
    <p><strong>WARNING! Deleting a record cannot be undone.</strong></p>

    <form method="post" action="/delete">
        <input type="hidden" name="id" value="<%= @player.id %>" />
        <input type="submit" value="Delete" onclick="return confirm('Are you sure?')" />
    </form>
</div>
```

forms/football_players/views/edit.erb

The part of the page that deals with record deletion is itself a form. It stores the `id` of the player as a hidden field, and uses the `post` method to send it to the `/delete` URL.

The `delete.rb` Controller

```
post "/delete" do
  id = params["id"]

  if Player.id_exists?(id)
    player = Player[id]
    player.delete
    redirect "/search"
  end

  erb :delete
end
```

Similar to the `edit.rb` controller, the `delete.rb` controller validates the `id` it is passed.

If the `id` exists, the record is deleted from the database, and the user is sent back to the search page.

`forms/football_players/controllers/delete.rb`