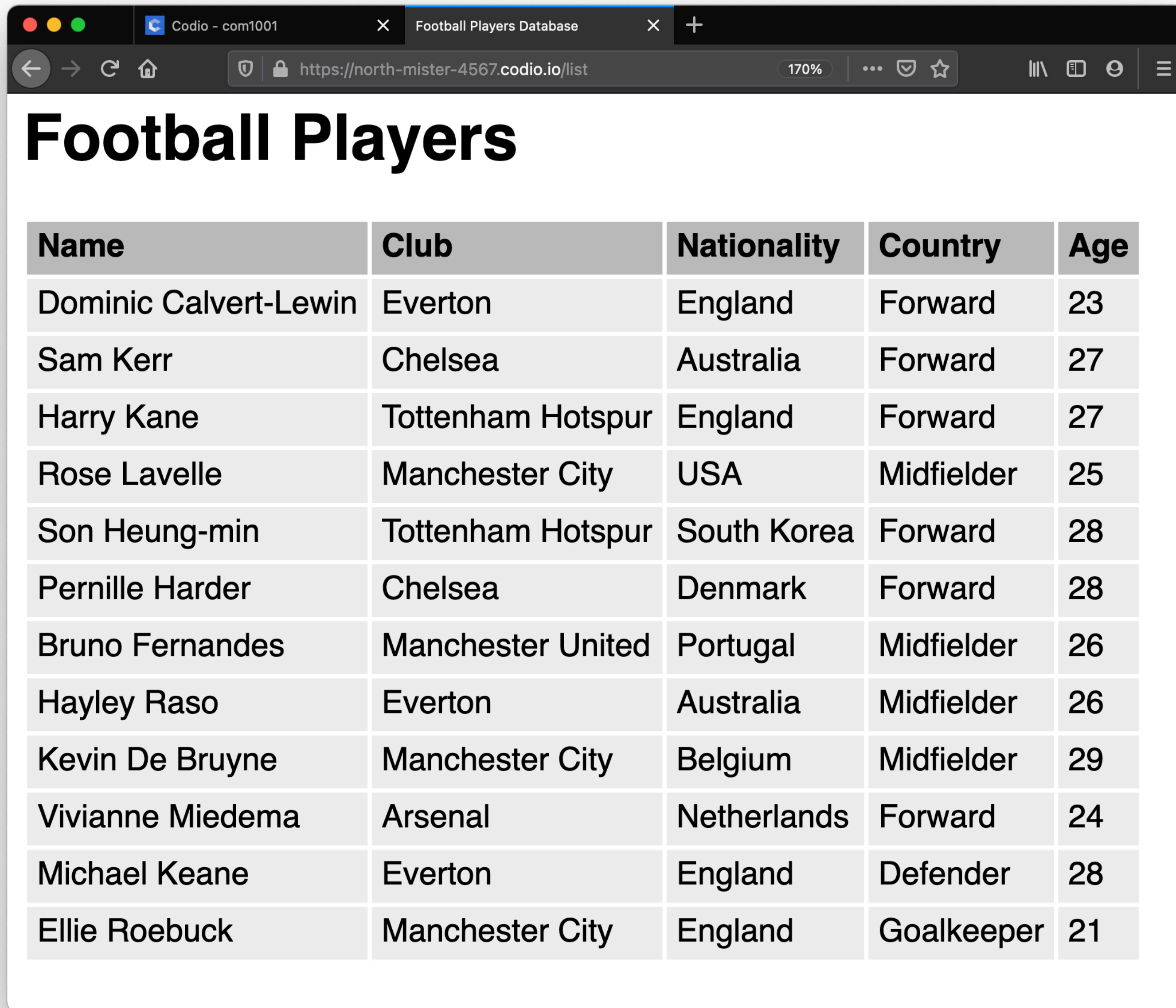


COM 1001

# INTRODUCTION TO SOFTWARE ENGINEERING

Professor Phil McMinn

## Using Models in Sinatra



The screenshot shows a web browser window with the title 'Football Players Database'. The address bar displays 'https://north-mister-4567.codio.io/list'. The main content area features a table titled 'Football Players' with the following data:

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

In this lecture we're going to build a simple application to display the contents of a database table in a web browser, with the help of a model.

Codio - com1001Football Players Database

←→↺🏠

https://north-mister-4567.codio.io/list

170%⋮🔖🌟

📄🔍🔄☰

# Football Players

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



# The Main Application File

```
# Gems
require "require_all"
require "sinatra"

# So we can escape HTML special characters in the view
include ERB::Util

# App
require_rel "db/db", "models", "controllers"
```

models/football\_players/app.rb

Database setup is  
in its own file in  
the `db/` directory

This includes and let us use the “`h`”  
method in the view. More on that to  
shortly...

We now include all Ruby code in the  
`models/` directory, as well as  
`controllers/` (next line of code)

# Setting up the Database

The database `sqlite3` file will always be in the same directory as this file. This line of code obtains that directory. “`__FILE__`” is the path to the current file, “`File.dirname`” gets its directory.

We’re also setting up the database to log SQL commands (for debugging) to the `logs` directory (which created if not present)

```
require "logger"
require "sequel"

# what mode are we in?
type = ENV.fetch("APP_ENV", "production")

# find the path to the database file
db_path = File.dirname(__FILE__)
db = "#{db_path}/#{type}.sqlite3"

# find the path to the log
log_path = "#{File.dirname(__FILE__)}/../log/"
log = "#{log_path}/#{type}.log"

# create log directory if it does not exist
Dir.mkdir(log_path) unless File.exist?(log_path)

# set up the Sequel database instance
DB = Sequel.sqlite(db, logger: Logger.new(log))
```

The purpose of this code is so that we can use a different database for testing and production –

we don’t want to be testing the app with real data, potentially adding, changing and removing records!

This code works by trying to look up the value of a variable in a global environment hash, `ENV`. If we have not set `APP_ENV`, the code sets “`production`” as the default for the variable `type` and this causes the file `db/production.sqlite3` to be used as the database file.

For testing, we can set `ENV[“APP_ENV”]` to “`test`”, so that a different database is used.

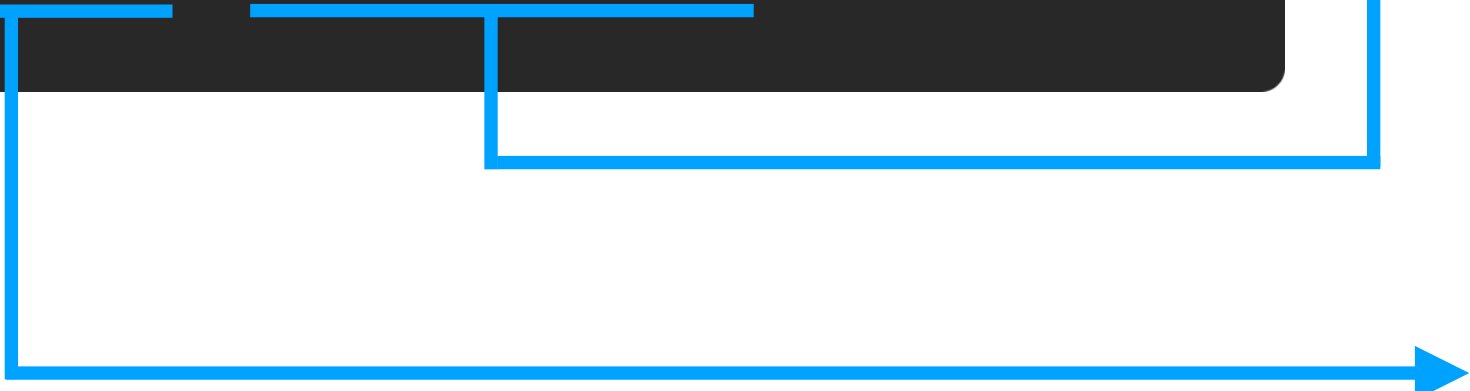
`models/football_players/db/db.rb`

# A Simple Controller and a Model We've Seen Before...

```
# Gems
require "require_all"
require "sinatra"

# So we can escape HTML special characters in the view
include ERB::Util

# App
require_rel "db/db", "models", "controllers"
```



```
get "/list" do
  @players = Player.all
  erb :list
end
```

models/football\_players/controllers/list.rb

```
require "time_difference"

# A player record from the database
class Player < Sequel::Model
  def name
    "#{first_name} #{surname}"
  end

  def age(at_date = Date.today)
    dob = Date.strptime(date_of_birth, "%Y-%m-%d")
    TimeDifference.between(dob, at_date).in_years.floor
  end
end
```

models/football\_players/models/player.rb



# The View

If there are no records  
in the database, it  
informs the user

Note the inclusion of the “**h**”s.  
This is a special method that  
“escapes” characters our  
strings so that they will be  
formatted correctly in a web  
page. If we don’t do this, any  
special characters like “<”  
and “>” in database field  
values will be interpreted as  
HTML by the browser. This  
method ensures they are not.

```
<html>
<head>
  <title>Football Players Database</title>
  <link rel="stylesheet" href="style/style.css">
</head>
<body>
  <h1>Football Players</h1>
  <% if @players.count > 0 %>
    <table>
      <tr>
        <th>Name</th>
        <th>Club</th>
        <th>Nationality</th>
        <th>Country</th>
        <th>Age</th>
      </tr>
      <% @players.each do |player| %>
        <tr>
          <td><%=h player.name %></td>
          <td><%=h player.club %></td>
          <td><%=h player.country %></td>
          <td><%=h player.position %></td>
          <td><%=h player.age %></td>
        </tr>
      <% end %>
    </table>
  <% else %>
    <p>The database is empty!</p>
  <% end %>
</body>
</html>
```

The view iterates through the  
**@players** array that was  
initialised in the controller.

It formats the details of each  
player in a HTML table, by  
calling methods of each  
**player** object (which are  
instances of the **Player**  
model class)

models/football\_players/views/list.erb

# Unit Tests for the Route

```
# Ensure we use the test database
ENV["APP_ENV"] = "test"
...
```

models/football\_players/helpers/spec\_helper.rb

Since we're going to have lots of test files going forward, instead of setting up the test environment each time, we're going to use a helper file to set everything up for us, including the `ENV["APP_ENV"]` variable that ensures the test database (`db/test.sqlite`) is used by `db.rb`

```
require_relative "../spec_helper"
...
```

models/football\_players/spec/unit/list\_spec.rb (part 1/3)



# Unit Test #1

The first test checks what happens with the application when the database is empty – the page should report this (refer back to the [view](#) earlier in the lecture)

```
describe "GET /list" do
  context "with an empty database" do
    it "says the database is empty" do
      get "/list"
      expect(last_response.body).to include("The database is empty!")
    end
  end
end

...
```

models/football\_players/spec/unit/list\_spec.rb (part 2/3)

# Unit Test #2

The second test checks the list of players itself.

It enters a record into the database ... and then checks that the page displays it.

...

```
context "with one record in the database" do
```

```
  it "lists the player" do
```

```
    # set up the test
```

```
    player = Player.new(first_name: "Test", surname: "Player", date_of_birth: "2020-1-1")
```

```
    player.save_changes
```

```
    # perform the test by going to the page and examining the content
```

```
    get "/list"
```

```
    expect(last_response.body).to include("Test Player")
```

```
    # reset the state of the database
```

```
    DB.from("players").delete
```

```
  end
```

```
end
```

```
context "with one record in the database" do
  it "lists the player" do
    # set up the test
    player = Player.new(first_name: "Test", surname: "Player", date_of_birth: "2020-1-1")
    player.save_changes

    # perform the test by going to the page and examining the content
    get "/list"
    expect(last_response.body).to include("Test Player")

    # reset the state of the database
    DB.from("players").delete
  end
end
end
```

models/football\_players/spec/unit/list\_spec.rb (part 3/3)

It's important to leave the database in the same state that we found it, ready for the next test, if there is one. If we did not ensure the same database state before each test, the state left after one test could interfere with another, causing it to **pass or fail when it shouldn't**. This makes the test unreliable. Unreliable tests are known as **flaky tests**.



# Unit Tests for the Model

```
require_relative "../spec_helper"

RSpec.describe Player do
  describe "#name" do
    it "returns the player's full name" do
      player = described_class.new(first_name: "A", surname: "B")
      expect(player.name).to eq("A B")
    end
  end

  describe "#age" do
    it "returns the age of the player" do
      player = described_class.new(date_of_birth: "2000-1-1")
      expect(player.age("2020-1-1")).to eq(20)
    end
  end
end
```

models/football\_players/spec/unit/player.rb

This code should look familiar!

The same helper we used in the feature test sets everything up for the unit test too.

We organise tests in the `spec/` subdirectory of the main application. Unit tests go in `spec/unit/`. (Other types of test we will encounter later in the module will go in a different subdirectory...)

# Understanding the File Structure

