COM 1001

Introduction to Software Engineering

Professor Phil McMinn

Test Coverage

Metrics for Testing

Test Coverage is a metric that describes **how much** of a system was tested by its tests (for example, unit and end to end tests). There are various ways to measures this.

Code Coverage is a type of **Test Coverage metric** that measure how much of our code was **executed** by the tests.

Line Coverage is a specific **Code Coverage metric** that measures how many code lines were executed.

Code Coverage tells us which parts of a system were **missed by our tests**, and what tests we may still need to write.

SimpleCov

SimpleCov is a gem that can track the lines of code executed by our RSpec tests.

We can set it up to work with our tests in `spec_helper.rb` as follows:

```ruby
# Configure coverage logging
require "simplecov"
SimpleCov.start do
  add_filter "/spec/"
end
SimpleCov.coverage_dir "coverage"
```

`forms/football_players/spec/spec_helper.rb`

This starts monitoring lines executed, ensuring we don't track the `spec/` directory – i.e., coverage of the tests themselves!

SimpleCov produces HTML reports in the `coverage/` directory of our app

SimpleCov Monitors Our Tests

We can supply `rspec` command with a directory name and it will run all the tests in it – anything ending in `_spec.rb`

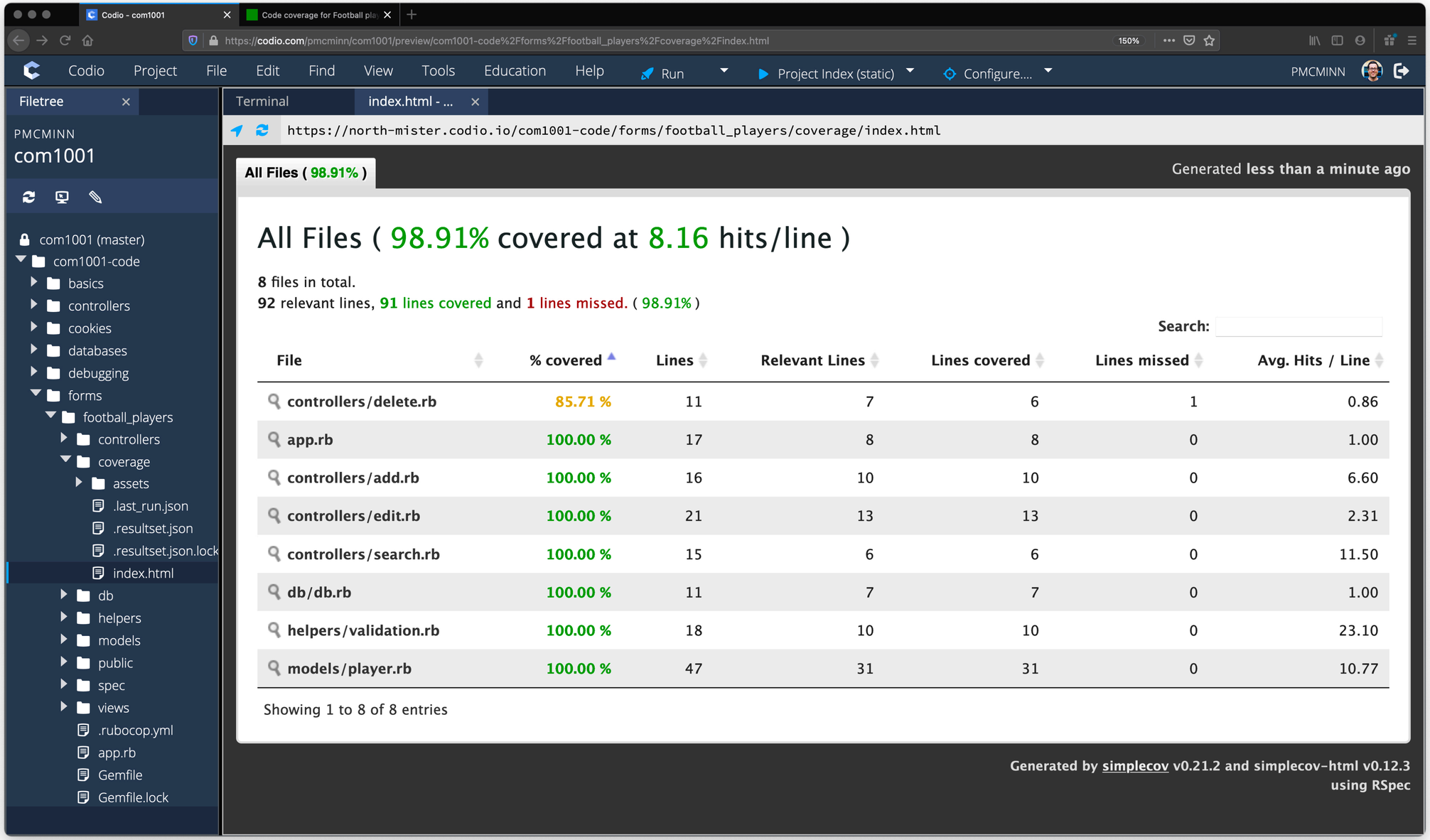
```
codio@north-mister:~/workspace/com1001-code/forms/football_players$ rspec spec/features/
............

Finished in 2.4 seconds (files took 2.38 seconds to load)
12 examples, 0 failures

Coverage report generated for RSpec to /home/codio/workspace/com1001-code/forms/football_players/coverage. 91 / 92 LOC (98.91%) covered.
codio@north-mister:~/workspace/com1001-code/forms/football_players$
```

This line is output by SimpleCov.

SimpleCov only counts code in `.rb` files, i.e., our controllers and models etc. It does not monitor coverage in `.erb` files. Hence only 92 **lines of code (LOC)** in total, and not more.

It says we executed (covered) 91/92 LOC.

We can find out which line the feature tests missed by checking out the coverage report.

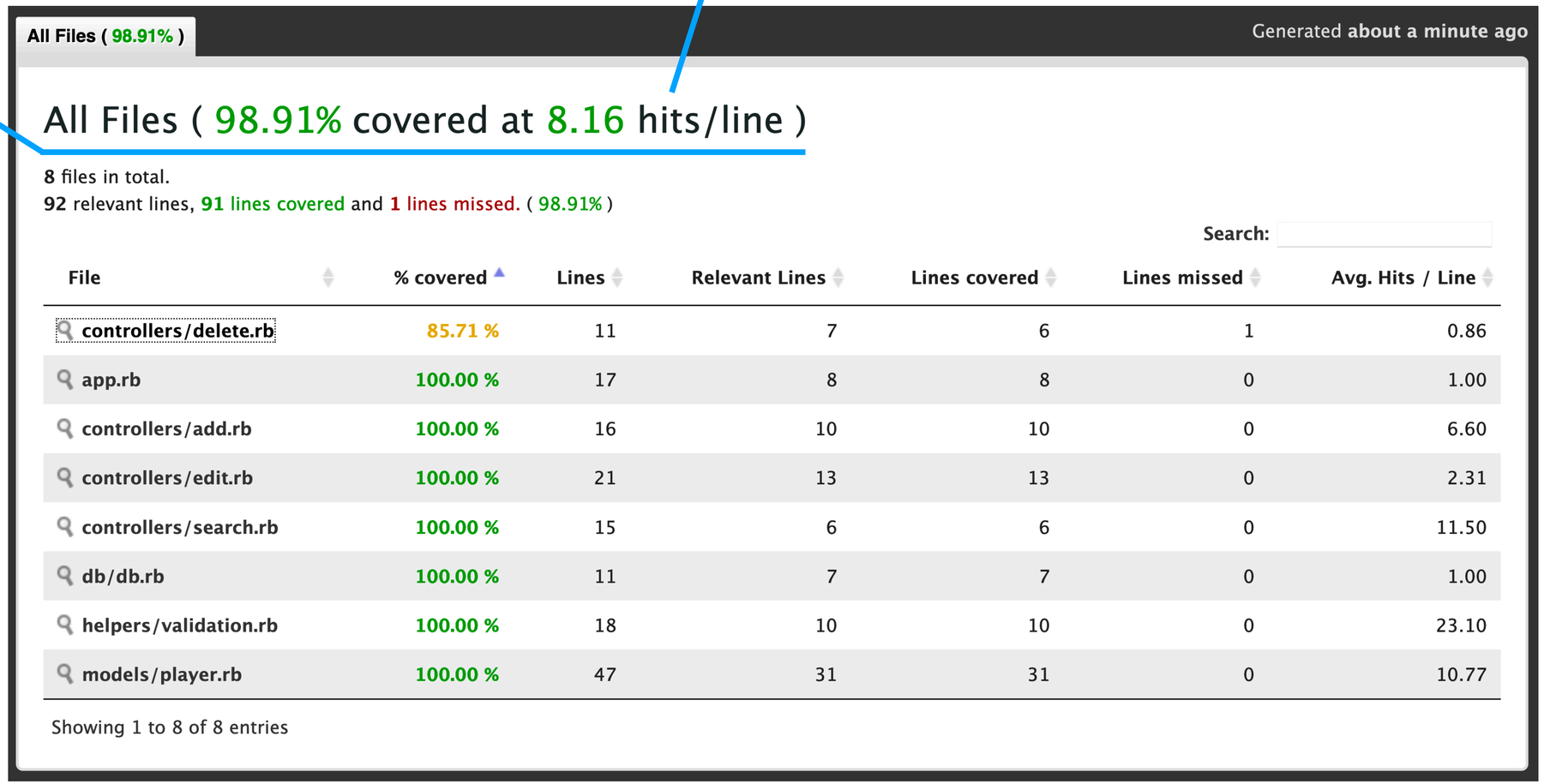


To open the report in Codio, go to the `coverage` directory in the file tree. Right click on `index.html` and click **"Preview static"**

Ideally, we want the **coverage percentage** to be as high as possible, and **hits/line** to be equal or greater to 1.

A “**hit**” is an execution of a line. “**hits/line**” is the number of times, on average, each line in the system was executed by the tests.

The more hits a line has, the more it has been executed. So we can see which files have been executed the most.

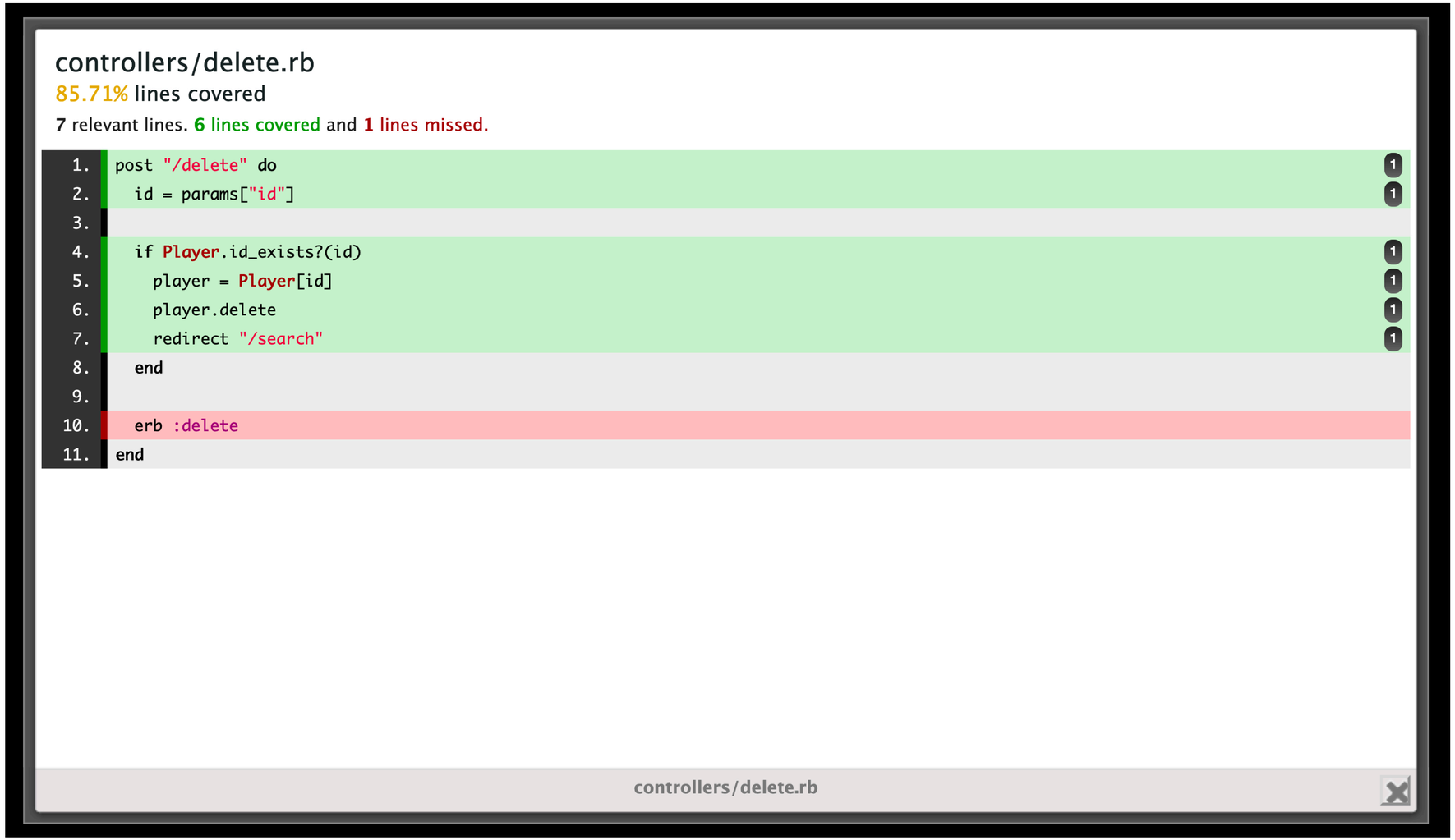


Coverage Tracking

We can see which lines were covered and not covered in each individual file.
Covered lines are in green, uncovered lines are in red. Whitespace/negligible code is in grey.

None of our feature tests executed the condition on line 4 as false, hence they never reached line 10, which displays an error message (player does not exist).

Can you think why this is not possible?



A user of our system can only delete players via the edit page, and they can only edit players already added to the database.

So there is no way for them to delete a non-existent player.

Remember we cannot visit POST routes in our browser, so nor can Capybara.

So we can never cover this line via an acceptance test. However, we need to test what happens if we did try to delete a player that did not exist (e.g., if attempted by a robot)

We can still do this via a unit test…

Additional Unit Test

```ruby
describe "POST/delete" do
  it "does not allow deletion of a invalid player" do
    post "/delete", "id" => "-100"
    expect(last_response.body).to include("Unknown player")
  end
end
```

forms/football_players/spec/unit/delete_spec.rb

This is why we need to have both **end-to-end tests** **and** **unit tests**:

Unit tests ensures the app works beyond what the user can see, by testing edge cases…

… but unit tests cannot, on their own, guarantee all the units will work together.

If we don't supply any parameters to the `rspec` command it will run all of our tests, feature and unit – anything ending in `_spec.rb`

```
codio@north-mister:~/workspace/com1001-code/forms/football_players$ rspec
...............

Finished in 2.38 seconds (files took 2.69 seconds to load)
15 examples, 0 failures

Coverage report generated for RSpec to /home/codio/workspace/com1001-code/forms/football_players/coverage. 92 / 92 LOC (100.0%) covered.
codio@north-mister:~/workspace/com1001-code/forms/football_players$
```

All lines of code now covered

Code Coverage: Caveats

Code Coverage is a primarily a measure of **test effort**, *not* **test effectiveness**

Merely executing a line of code in a test does not guarantee that it is bug free.

High coverage increases our confidence in correctly working software, but writing tests *just* to achieve coverage is not always sensible or a good use of time.