

COM 1001

# INTRODUCTION TO SOFTWARE ENGINEERING

**Professor Phil McMinn**

## Sessions

# Disadvantages of Cookies

Although **HTTP Cookies** provide the ability to track users:

- The information that can be stored in them is limited to strings
- They are cumbersome if you want to store lots of different information
- They are insecure – cookie data is transmitted in plain text (unless HTTPS is being used)

Most web DSLs, frameworks, and dedicated web programming languages provide the concept of a **session** to circumvent these issues.

# Sinatra Sessions

As far as programming in Sinatra is concerned, a **session** is simply a **hash** that can be used to store information about the user that is remembered from one page to another.

Sinatra implements **session** by means of a **session cookie** (but, be careful not confuse the two!)

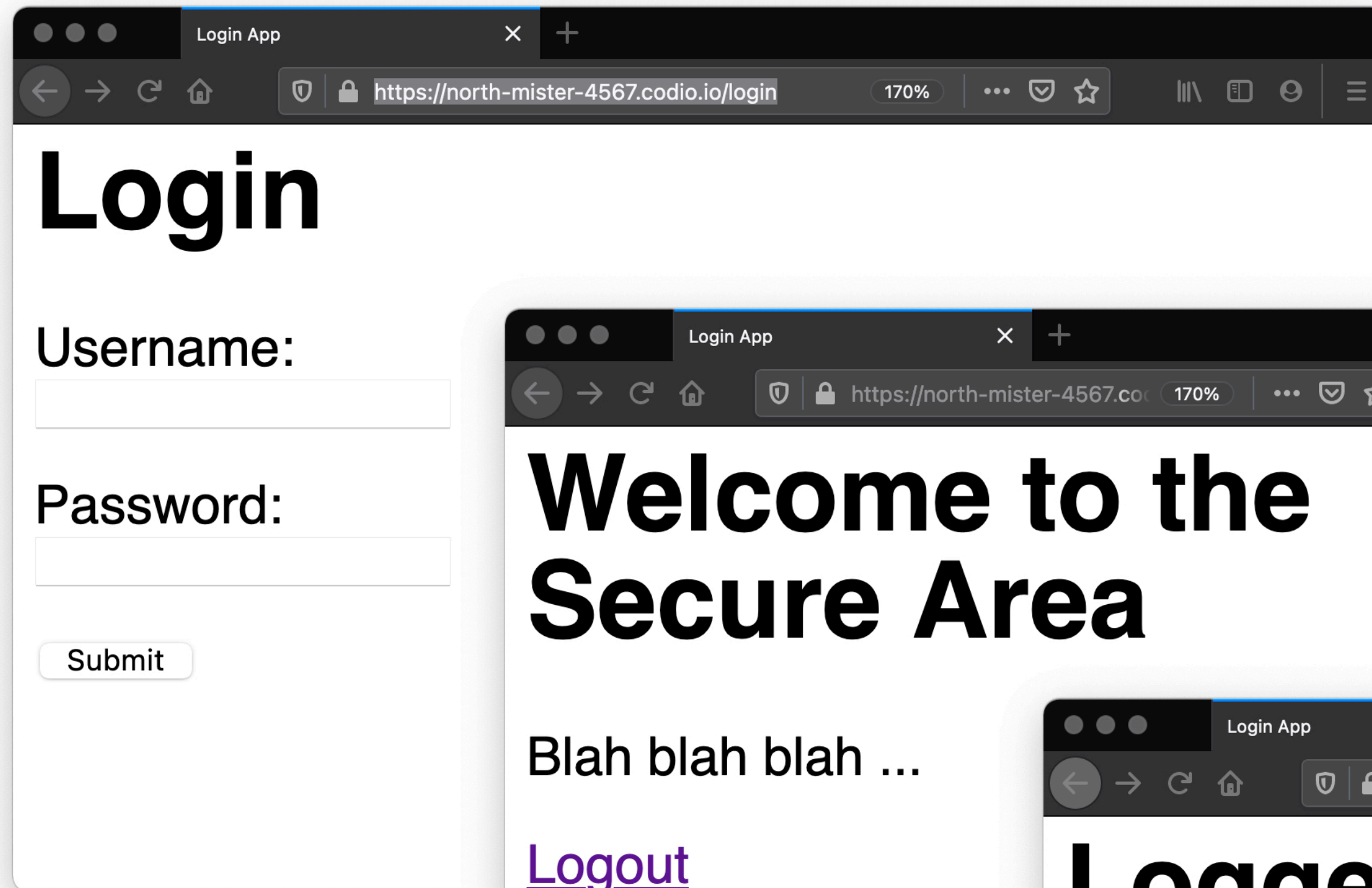
The session hash is converted to string form and encrypted into the cookie to make it secure.

# Sessions: Practical Examples

Storing the fact that a user has logged into an application is one example of a potential use for a session.

Others might include the current basket of items a user has when using an e-commerce site.

Can you think of more?



Login App

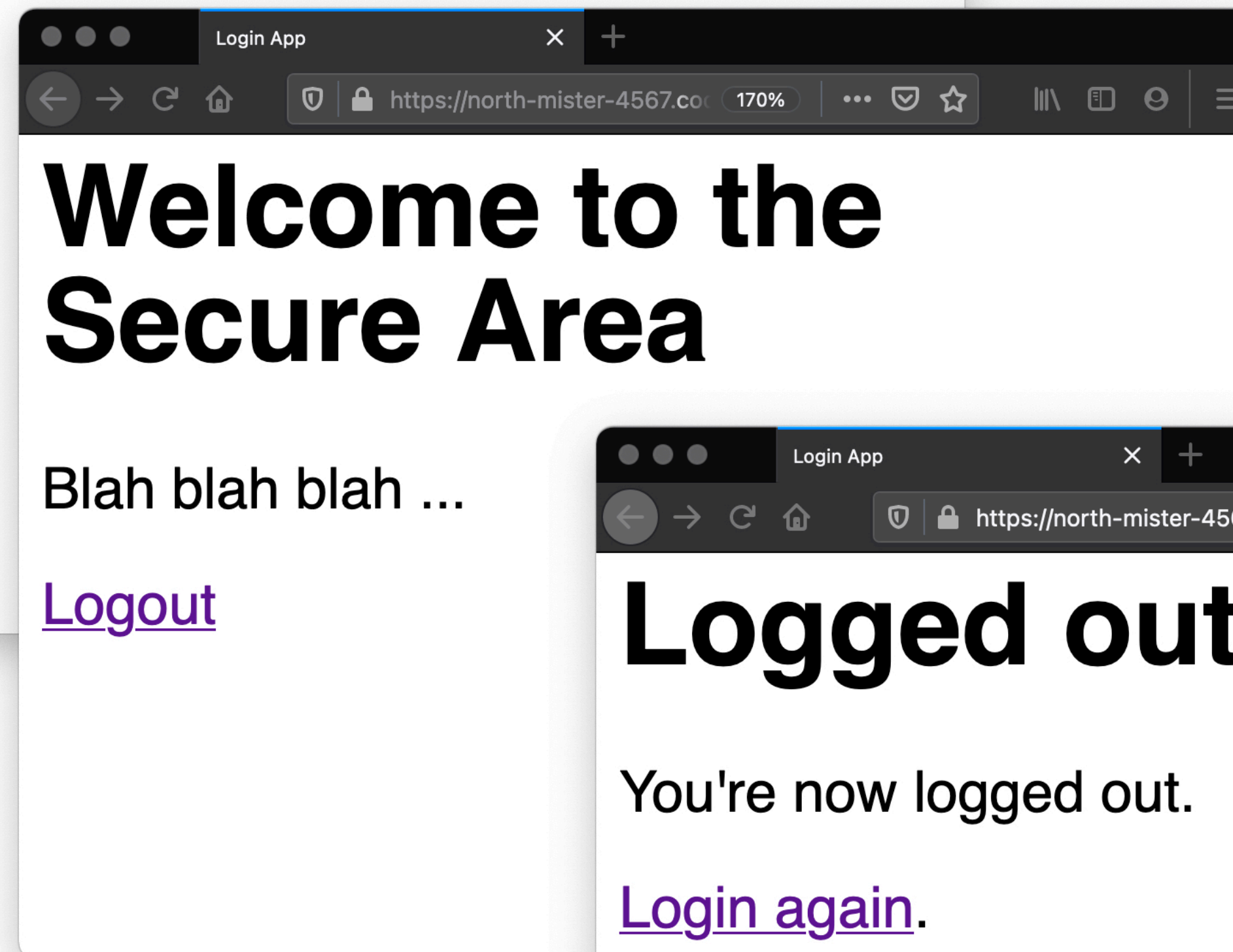
https://north-mister-4567.codio.io/login

## Login

Username:

Password:

Submit



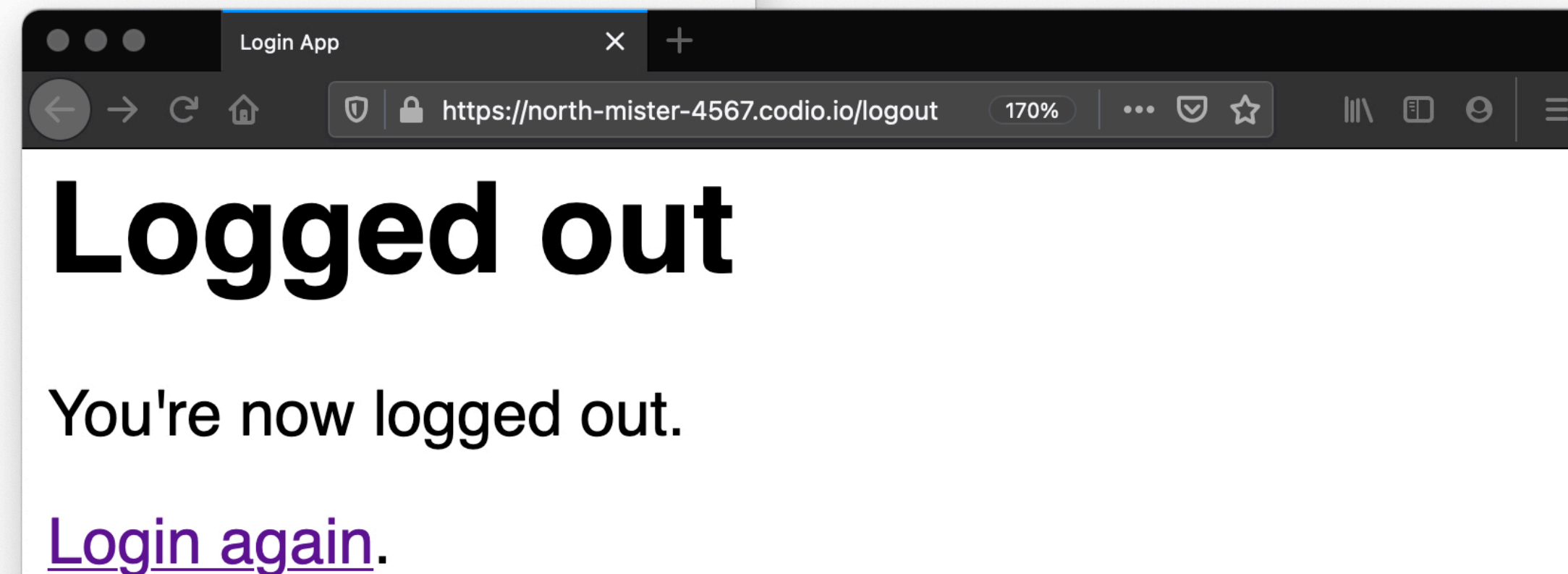
Login App

https://north-mister-4567.co

## Welcome to the Secure Area

Blah blah blah ...

[Logout](#)



Login App

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

## Logged out

You're now logged out.

[Login again.](#)

# Login Example

Check out the [sessions/login](#) example in the repository as a practical example of how to use a session.

```
...  
  
# Sessions  
enable :sessions  
set :session_secret, "$g]Rd2M/WbJ`~~<GZWdH@Fm'ESk2_gckCtLJJkySYG"  
  
...
```

sessions/login/app.rb

This is part of [app.rb](#).

We have to explicitly configure Sinatra to use sessions.

We also need to give it an **encryption key** for encoding data in the session cookie. It can be anything, I randomly generated this one. There are more sophisticated means of doing this...



# The Controller, Part 1

```
get "/login" do
  @user = User.new
  erb :login
end

post '/login' do
  @user = User.new
  @user.load(params)
  @error = nil

  if @user.valid?
    if @user.exist?
      session[:logged_in] = true
      redirect "/"
    else
      @error = "Username/Password combination incorrect"
    end
  else
    @error = "Please correct the information below"
  end

  erb :login
end
```

The main login page is a form. Users and passwords appear in a database table `users`, which is fronted in the application with a model called `User`. The way this works is similar to `Player` in the `forms/football_players` example, except there is an additional method for checking a username and password are valid (i.e., exist in the database).

The key part of this code snippet is what happens when the username and password are valid – we set the `:logged_in` key of the `session` hash to `true`.

# The Controller, Part 2

```
get "/" do
  redirect "/login" unless session[:logged_in]
  erb :index
end

get "/logout" do
  session.clear
  erb :logout
end
```

You're not allowed to see this page unless you're logged in! If you're not logged in, you get redirected to the `/login` route.

We can log a user out by setting `session[:logged_in]` to false, or we can just clear the `session` hash completely.

sessions/login/controllers/login.rb  
(part 2/2)