Adding Edit and Delete functionality

- In order to edit/update records in our table, we'll need to have access to the id.
- We will use a concept called Path Variables for this.

Path Variables

- We already saw how we can access parameters passed in from an html form into our controller methods:

Example only, not part of current application

```
36 @PostMapping("/register")
37 public String registerUser(
38         @RequestParam String firstName,
39         @RequestParam String lastName,
40         Model model)   {
41
42         User user = new User(firstName, lastName);
43         model.addAttribute("user", user);
44
45         return "welcome_page";
46
47     }
```

Path Variables

- However, we also want to be able to dynamically respond to requests that have extra information that is added to the path itself.

http://localhost:8080/editAvenger/1

http://localhost:8080/deleteAvenger/2

- In the examples above, this would mean
  - Edit the Avenger with id = 1
  - Delete the Avenger with id = 2

Path Variables

- We can easily do this with @PathVariable annotation.

```
@GetMapping("/deleteAvenger/{id}")
public String deleteAvenger(@PathVariable Long id) {
```

- We are specifying the last element in the path is dynamic and we are naming it id.

```
@GetMapping("/deleteAvenger/{id}")
```

- We are asking Spring to supply us this value in our method as a parameter

```
public String deleteAvenger(@PathVariable Long id)
```

Modifying index.html

```
<tr th:each="avenger : ${avengerList}">
    <td th:text="${avenger.name}">Thor</td>
    <td th:text="${avenger.age}">32</td>
    <td><a href="#" th:href="|/editAvenger/${avenger.id}|">edit</a></td>
    <td><a href="#" th:href="|/deleteAvenger/${avenger.id}|">delete</a></td>
</tr>
```

Welcome to the Avengers Database

Name	Age	
Thor	32	edit delete
Nebula	28	edit delete

Add an Avenger

```
<tr>
    <td>Thor</td>
    <td>32</td>
    <td><a href="/editAvenger/1">edit</a></td>
    <td><a href="/deleteAvenger/1">delete</a></td>
</tr>
<tr>
    <td>Nebula</td>
    <td>28</td>
    <td><a href="/editAvenger/2">edit</a></td>
    <td><a href="/deleteAvenger/2">delete</a></td>
</tr>
```

Implementing Delete

- First, we will create a method in our DatabaseAccess class to implement this
- Then we will add a mapping in our controller class that will invoke this method, then return to the index page.

Implementing delete in DatabaseAccess

```
72⊖     /**
73       * Method to delete a single Avenger
74       * @param id: the id of the Avenger to be deleted
75       * @return the number of rows affected; 1 - successful, 0 - not
76       */
77⊖     public int deleteAvenger(Long id) {
78
79         // create a new instance of MapSqlParameterSource for our use
80         MapSqlParameterSource namedParameters =
81                 new MapSqlParameterSource();
82
83         String query = "DELETE FROM avengers WHERE id = :id";
84
85         // add the parameter to our map
86         namedParameters.addValue("id", id);
87
88         // Let our Jdbc template do the work
89         int returnValue = jdbc.update(query, namedParameters);
90
91         // Will return the number of rows affected
92         return returnValue;
93     }
```

Implementing delete in DatabaseAccess

```
MapSqlParameterSource parameters = new MapSqlParameterSource();

// this creates a parameterized query with a parameter called
// myId (notice the : has to precede the name
String query = "DELETE FROM avengers WHERE id = :myId";

// Here we re adding a name-value pair in the map
// with name = myId and the value = id (coming in as a parameter)
parameters.addValue("myId", id);

// We pass both the parameterized query as we as the map to the jdbc template
// at run-time, the jdbc template will see there is a parameter called myId in
// the query and it will look in the map for name-value pair with the same name (myId)
// and will then replace the :myId placeholder with the value corresponding to
// the name myId in the map in the query.

// for example, if the id passed in to the method is 4 (say), then
// the jdbc template will execute "DELETE FROM avengers WHERE id = 4" on the database
int returnValue = jdbc.update(query, parameters);

return returnValue;
```

Implementing delete in our Controller

```
72⊖     /**
73       * @param id will be the last element in the url path
74       * @return "redirect:/" Redirects the request to the '/' resource
75       */
76⊖     @GetMapping("/deleteAvenger/{id}")
77      public String deleteAvenger(@PathVariable Long id) {
78
79          // call the deleteAvenger method of the DatabaseAccess class
80          int returnValue = database.deleteAvenger(id);
81
82          // we're not doing anything with this now, but we could
83          // send a message back through the model, use Elvis ...
84          System.out.println("return value is: " + returnValue);
85
86          // redirect to '/', so we don't have to add to the model...
87          return "redirect:/";
88
89      }
```

Implementing Editing

- Editing an *Avenger* will involve a bit more work.
  - When the user clicks the edit link…
    - We will use the *id* get an instance of *Avenger* from DatabaseAccess (new method).
    - We will forward the instance to an *edit_avenger* page, which is very similar to add_avenger
    - When the form submits, it will invoke an *updateAvenger* method in our controller.

Implementing Editing

- Our controller will then invoke an *updateAvenger*
- method on our DatabaseAccess class.

Implementing getAvenger in DatabaseAccess

```
78      /**
79       * Gets an Avenger from the database
80       *
81       * @param id The id of the Avenger to get
82       * @return The Avenger if found, a otherwise
83       */
84      public Avenger getAvenger(long id) {
85
86          // create a new instance of MapSqlParameterSource for our use
87          MapSqlParameterSource parameters = new MapSqlParameterSource();
88
89          String query = "SELECT * FROM avengers WHERE id = :id";
90
91          parameters.addValue("id", id);
92
93          // will map a row to an instance of Avenger
94          BeanPropertyRowMapper<Avenger> mapper = new BeanPropertyRowMapper<>(Avenger.class);
95
96          Avenger avenger = null; // declare and initialize to null
97
98          try {
99              // Use the queryForObject method to get the one instance
100             avenger = jdbc.queryForObject(query, parameters, mapper);
101         } catch (EmptyResultDataAccessException ex) {
102             // We could get an exception if there is no match
103             // Just print out to the console for now
104             System.out.println("Avenger not found for id=" + id);
105         }
106         return avenger;
107     }
```

Implementing update in DatabaseAccess

```
93      /**
94       * updates an Avenger in the database
95       * @param avenger: the Avenger to add
96       * @return the number of rows affected; 1 - successful, 0 - not.
97       */
98      public int updateAvenger(Avenger avenger) {
99
100         // create a new instance of MapSqlParameterSource for our use
101         MapSqlParameterSource namedParameters =
102                 new MapSqlParameterSource();
103
104         String query =
105                 "UPDATE avengers SET name = :name, age = :age "
106                 + "WHERE id = :id";
107
108         // add the parameters to our map
109         namedParameters
110             .addValue("name", avenger.getName())
111             .addValue("age", avenger.getAge())
112             .addValue("id",  avenger.getId());
113
114         int returnValue = jdbc.update(query, namedParameters);
115
116         return returnValue;
117     }
```

Adding /editAvenger in HomeController

```
75      /**
76       * This method is invoked by the framework as a result of any
77       *  request of the type /editAvenger/#, where # is the id of the
78       *  one the user wants to edit.
79       * @param: id The id coming in as a path variable
80       * @param: model We need the model to send information
81       *         back to Thymeleaf
82       * @return: The Thymeleaf html template to continue the edit
83       *          process. I.e. edit_avenger.html (extension optional)
84       */
85      @GetMapping("/editAvenger/{id}")
86      public String editAvenger(@PathVariable Long id, Model model) {
87
88          // Given the id, get the corresponding Avenger from the database
89          Avenger avenger = database.getAvenger(id);
90
91          if (avenger == null) {
92              // Error condition. Simply log and return to index for now.
93              System.out.println("No result for id=" + id);
94              return "redirect:/";
95          }
96
97          // add the instance of Avenger to the model. This instance
98          // will be used in the form binding of the edit_avenger page
99          model.addAttribute("avenger", avenger);
100
101         // Forward the request to the page where the user will be
102         // able to do the actual editing of values.
103         return "edit_avenger";
104     }
```

Copy over the add_avenger.html

- We will re-use the add_avenger code for our edit_avenger.

1. Right-click the add_avenger.html from the templates and select copy.
2. Right-click the templates folder again and select paste.

edit_avenger.html

```
1  <!DOCTYPE html>
2  <html xmlns:th="http://www.thymeleaf.org">
3  <head>
4  <meta charset="UTF-8">
5  <title>Edit an Avenger</title>
6  </head>
7  <body>
8      <h1>Enter your hero's information</h1>
9
10     <form action="#" th:action="@{/updateAvenger}" method= "post"
11         th:object="${avenger}">
12
13         <!-- We need the id field populated in our HomeController -->
14         <!-- when we get it back via the @ModelAttribute -->
15         <!-- so we will put it here as a hidden field -->
16         <input type="hidden" th:field="*{id}">
17         <p>Name: <input type="text" th:field="*{name}"></p>
18         <p>Age: <input type="number" th:field="*{age}"></p>
19
20         <p><input type="submit" value="Update!"></p>
21     </form>
22 </body>
23 </html>
```

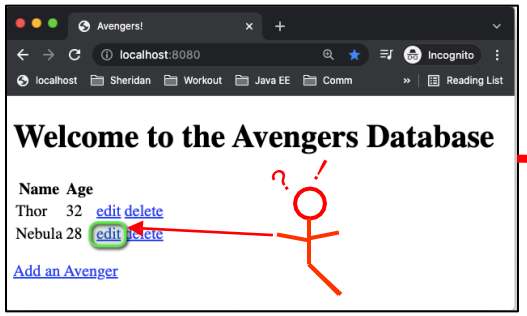
Challenge:
Use a model attribute and Thymeleaf to have only one add_edit_avenger.html file

Adding /updateAvenger in HomeController

```
112     /**
113      * @param avenger the avenger that was bound to the form
114      * @return "redirect:/" Redirects the request to the '/' resource
115      */
116     @PostMapping("/updateAvenger")
117     public String updateAvenger(@ModelAttribute Avenger avenger) {
118
119         // call the updateAvenger method of the DatabaseAccess class
120         int returnValue = database.updateAvenger(avenger);
121
122         // we're not doing anything with this now, but we could
123         // send a message back through the model, use Elvis ...
124         System.out.println("return value is: " + returnValue);
125
126         // redirect to '/', so we don't have to add to the model...
127         return "redirect:/";
128     }
```

What's going on?!!? (see also next slides)

1. User clicks on the edit link



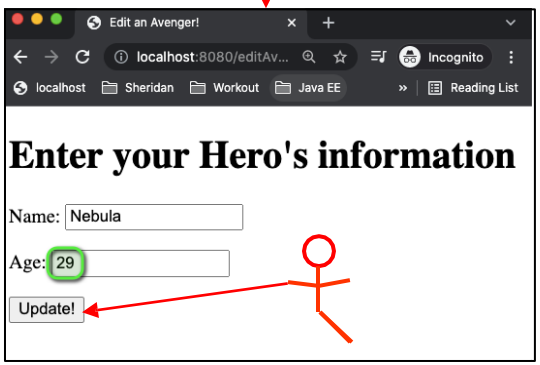
2. Code in controller is invoked (editAvenger)

```
/**
 * This method is invoked by the framework as a result of any
 *  request of the type /editAvenger/#, where # is the id of the
 *  one the user wants to edit.
 * @param: id The id coming in as a path variable
 * @param: model We need the model to send information
 *         back to Thymeleaf
 * @return: The Thymeleaf html template to continue the edit
 *          process. I.e. edit_avenger.html (extension optional)
 */
@GetMapping("/editAvenger/{id}")
public String editAvenger(@PathVariable Long id, Model model) {

    // Given the id, get the corresponding Avenger from the database
    Avenger avenger = database.getAvenger(id);

    // add the instance of Avenger to the model. This instance
    // will be used in the form binding of the edit_avenger page
    model.addAttribute("avenger", avenger);

    // Forward the request to the page where the user will be
    // able to do the actual editing of values.
    return "edit_avenger";
}
```

3. Controller gets the Avenger from the db class and sends it to Thymeleaf edit_avenger

```
<meta charset="UTF-8">
<title>Edit an Avenger!</title>
</head>
<body>
    <h1>Enter your Hero's information</h1>

    <!-- The form is bound to the avenger instance passed in -->
    <!-- via the model. Notice the action is /updateAvenger -->
    <form action="#" th:action="@{/updateAvenger}" method="post"
        th:object="${avenger}">

        <!-- we want to have the id come back on form submit -->
        <!-- but don't want to display it to the user -->
        <!-- so we bind that field to a hidden input -->
        <input type="hidden" th:field="*{id}">

        <!-- We bind the name and age fields to inputs so that the -->
        <!-- user can view the existing values as well as change them -->
        <p>Name: <input type="text" th:field=*{name}></p>
        <p>Age: <input type="number" th:field=*{age}></p>

         <!-- Once the user hits submit, off to /updateAvenger we go! -->
        <p><input type="submit" value="Update!"></p>
    </form>
</body>
</html>
```

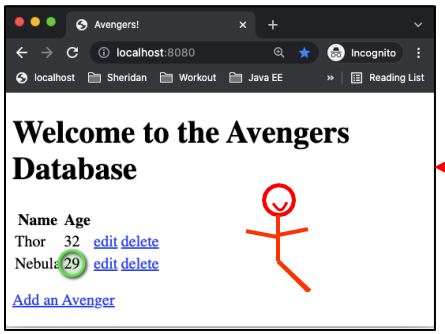
4. Thymeleaf renders the html, and it is sent to the user for their editing



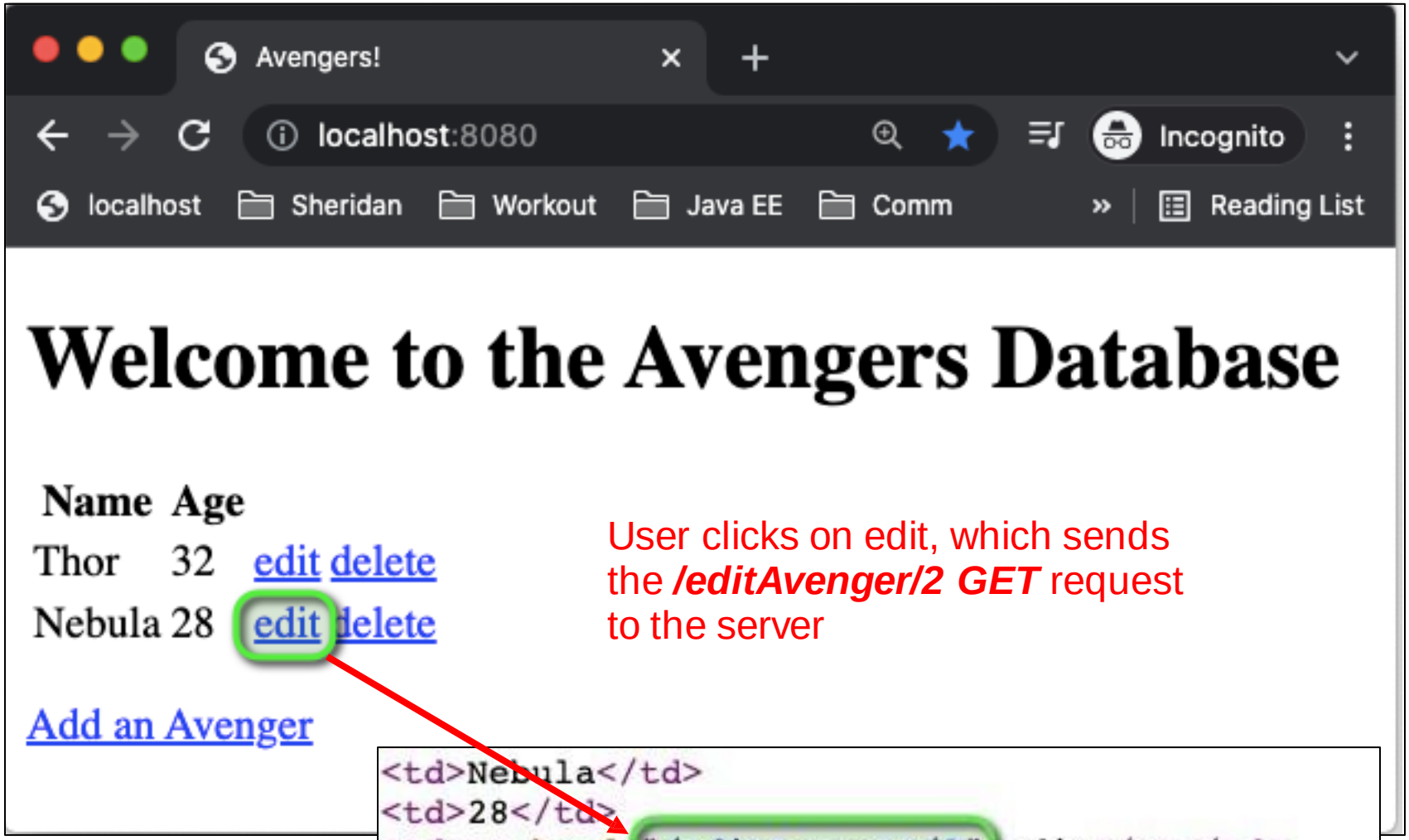
5. User clicked submit and code in controller is invoked (updateAvenger)

```
/**
 * This method is invoked by the framework as a result of the
 *  updateAvenger request from the form in the edit_avenger.html.
 *  I.e. when the clicks the "Update!" button
 * @param avenger A ModelAttribute that Spring will create with the
 *        request params coming in. Recall the hidden 'id', 'name'
 *        and 'age' fields on the form. Spring will create an instance
 *        of avenger with those values and pass it in here.
 * @return Redirects to the root.
 */
@PostMapping("/updateAvenger")
public String updateAvenger(@ModelAttribute Avenger avenger) {

    // Call on the database to perform the update
    int returnValue = database.updateAvenger(avenger);

    // for debugging purposes only
    // System.out.println("return value is: " + returnValue);

    // redirect to the root
    return "redirect:/";
}
}
```

6. User is sent back to the root with the updated information displayed



What's going on 1



What's going on 2

id will be 2 in this case

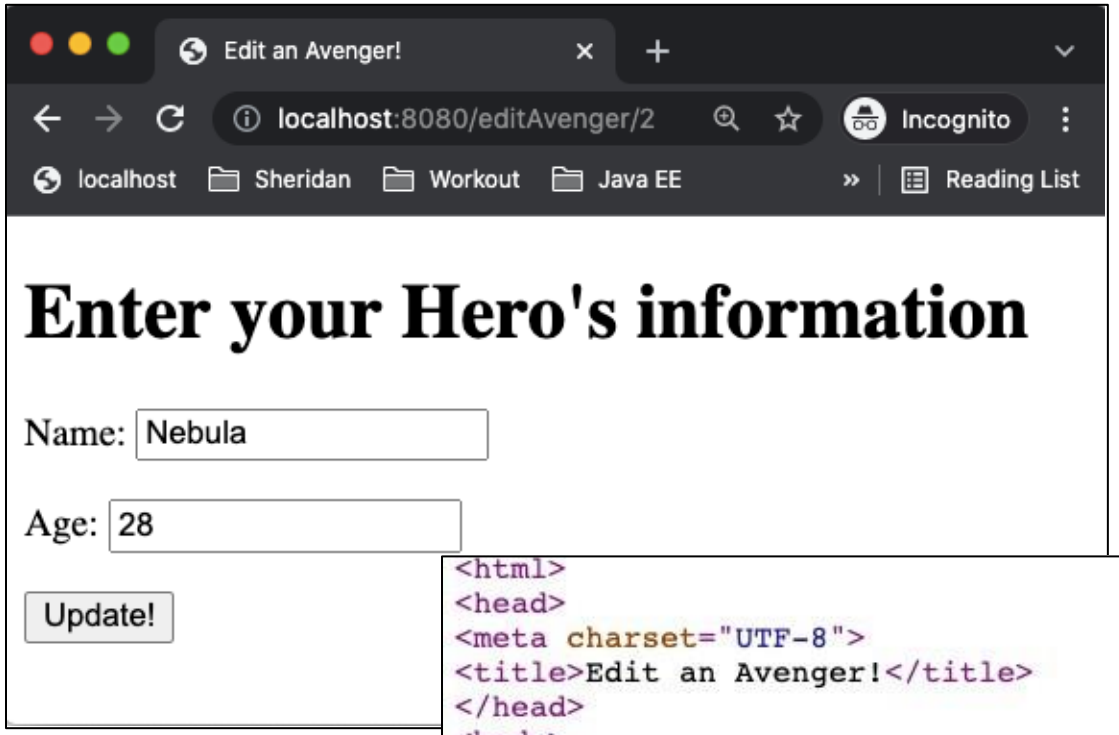
```
85      @GetMapping("/editAvenger/{id}")
86      public String editAvenger(@PathVariable Long id, Model model) {
87
88          // Given the id, get the corresponding Avenger from the database
89          Avenger avenger = database.getAvenger(id);
90
91          if (avenger == null) {
92              // Error condition. Simply log and return to index for now.
93              System.out.println("No result for id=" + id);
94              return "redirect:/";
95          }
96
97          // add the instance of Avenger to the model. This instance
98          // will be used in the form binding of the edit_avenger page
99          model.addAttribute("avenger", avenger);
100
101         // Forward the request to the page where the user will be
102         // able to do the actual editing of values.
103         return "edit_avenger";
104     }
```

What's going on 3

edit_avenger.html

```
4  <meta charset="UTF-8">
5  <title>Edit an Avenger!</title>
6  </head>
7  <body>
8      <h1>Enter your Hero's information</h1>
9
10     <!-- The form is bound to the avenger instance passed in -->
11     <!-- via the model. Notice the action is /updateAvenger -->
12     <form action="#" th:action="@{/updateAvenger}" method="post"
13         th:object="${avenger}">
14
15        <!-- we want to have the id come back on form submit -->
16        <!-- but don't want to display it to the user -->
17        <!-- so we bind that field to a hidden input -->
18        <input type="hidden" th:field="*{id}">
19
20        <!-- We bind the name and age fields to inputs so that the -->
21        <!-- user can view the existing values as well as change them -->
22        <p>Name: <input type="text" th:field=*{name}></p>
23        <p>Age: <input type="number" th:field=*{age}></p>
24
25         <!-- Once the user hits submit, off to /updateAvenger we go! -->
26        <p><input type="submit" value="Update!"></p>
27     </form>
28 </body>
29 </html>
```

What's going on 4



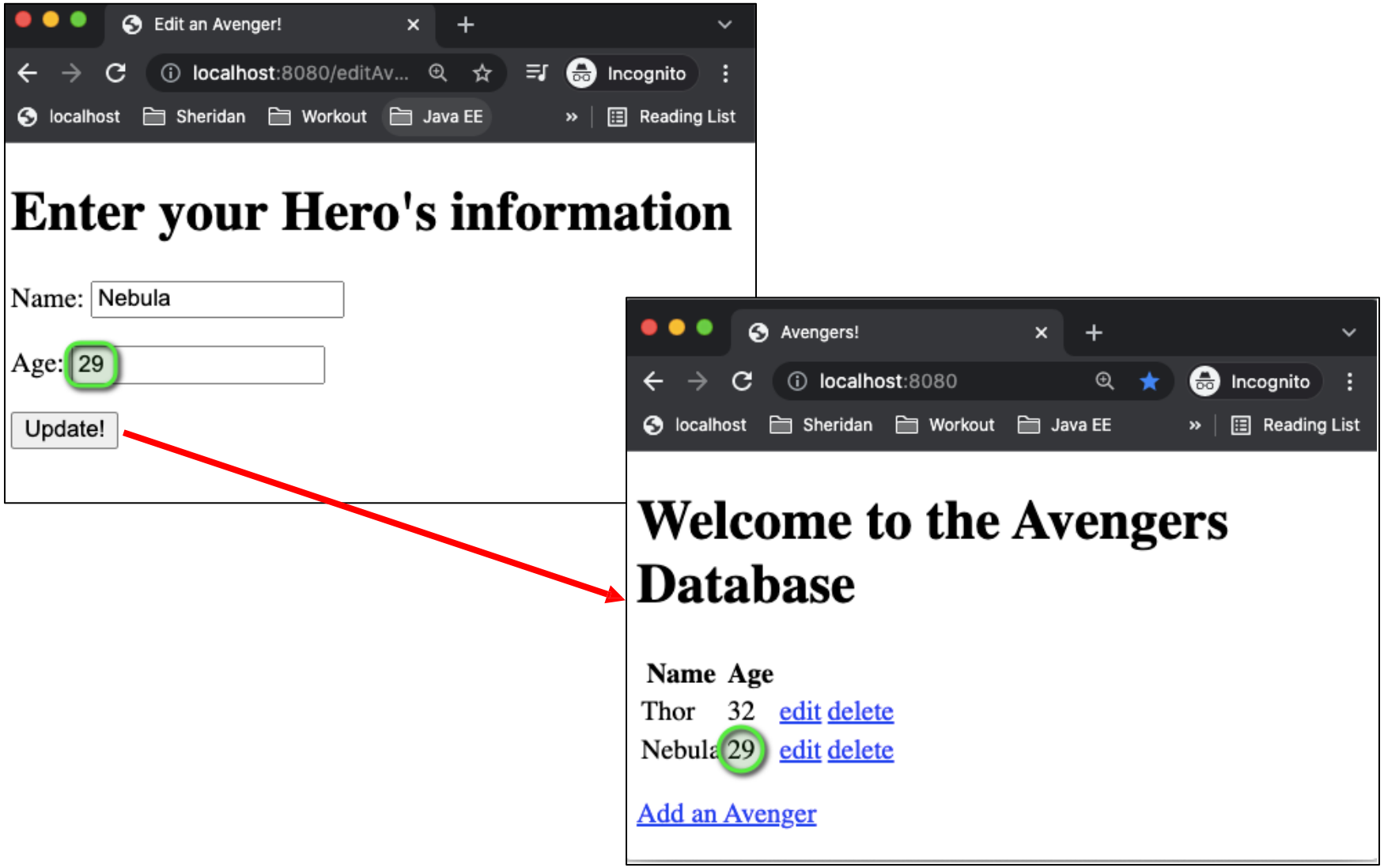
edit_avenger.html rendered by Thymeleaf and sent back to the user. Notice also the input ids, names and values

```
<html>
<head>
<meta charset="UTF-8">
<title>Edit an Avenger!</title>
</head>
<body>
    <h1>Enter your Hero's information</h1>

    <form action="/updateAvenger" method="post">
        <input type="hidden" id="id" name="id" value="2">
        <p>Name: <input type="text" id="name" name="name" value="Nebula"></p>
        <p>Age: <input type="number" id="age" name="age" value="28"></p>

        <p><input type="submit" value="Update!"></p>

    </form>
</body>
</html>
```

What's going on 5

```
99      /**
100      * This method is invoked by the framework as a result of the
101      *  updateAvenger request from the form in the edit_avenger.html.
102      *  I.e. when the clicks the "Update!" button
103      * @param avenger A ModelAttribute that Spring will create with the
104      *        request params coming in. Recall the hidden 'id', 'name'
105      *        and 'age' fields on the form. Spring will create an instance
106      *        of avenger with those values and pass it in here.
107      * @return Redirects to the root.
108      */
109     @PostMapping("/updateAvenger")
110     public String updateAvenger(@ModelAttribute Avenger avenger) {
111
112         // Call on the database to perform the update
113         int returnValue = database.updateAvenger(avenger);
114
115         // for debugging purposes only
116         // System.out.println("return value is: " + returnValue);
117
118         // redirect to the root
119         return "redirect:/";
120     }
121 }
```

What's going on 6



Get it to work!

- Pat yourself on the back!
- You just created a full-fledged, database-driven enterprise application using the Spring Boot framework

General References

1. Notes from Prof. Jonathan Penava, Sheridan College
2. Notes from Prof. Simon Hood, Sheridan College
3. Slides from Prof Paul Bonenfant
4. https://www.thymeleaf.org/
5. https://www.baeldung.com/
6. https://docs.spring.io/
7. https://www.baeldung.com/spring-pathvariable