ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

UNIVERSIDAD DE CÓRDOBA

Programación web

Prácticas

Semana 6: HTML (II)

Aurora Ramírez Quesada (aramirez@uco.es)

Departamento de Ciencia de la Computación e Inteligencia Artificial

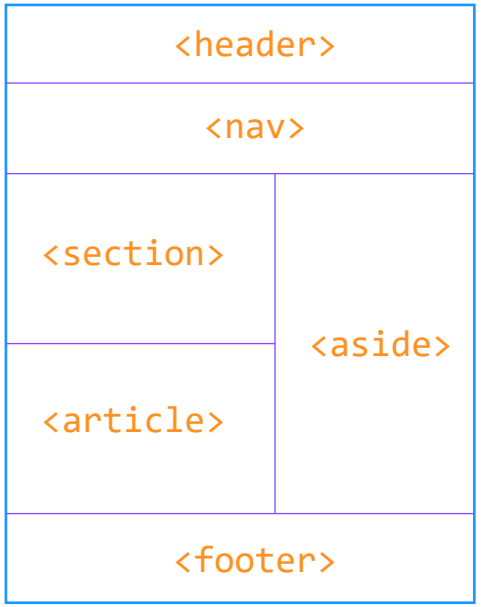
Universidad de Córdoba

Índice de contenido

Etiquetas básicas
Marcado semántico

https://www.w3schools.com/html/html5_semantic_elements.asp

- **Marcado semántico**: elementos que no necesariamente están destinados a afectar la estructura de las páginas, pero dan información adicional sobre ellas
- Su propósito es describir con mayor precisión el contenido de una página para otros programas como lectores de pantalla o motores de búsqueda
- Elementos para definir partes de un documento:
  - **<header>**: Cabecera del documento (puede haber varias)
  - **<nav>**: Sección de links de navegación
  - **<section>**: Sección de un documento, como una información temática, capítulo de contenido, etc.
  - **<article>**: Elemento de contenido independiente y auto-suficiente (noticia, blog, comentarios, etc.)
  - **<footer>**: Pie de página que suele contener información sobre autoría, contacto, mapa del sitio, etc.



Etiquetas básicas
Marcado semántico

https://www.w3schools.com/html/html_formatting.asp

- El elemento **** indica que su contenido tiene una gran importancia
  - Los navegadores muestran el contenido del elemento en **negrita**
- El elemento **** indica un énfasis que cambia sutilmente el significado de la oración
  - Los navegadores muestran el contenido del elemento en *cursiva*

```
<p>
    <strong>Beware:</strong> Pickpockets operate
    in this area.
</p>
<p>This toy has many small pieces and is
    <strong>not suitable for children under five
     years old.</strong>
</p>
```

```
<p>I <em>think</em> Ivy was the first.</p>
<p>I think <em>Ivy</em> was the first.</p>
<p>I think Ivy was the <em>first</em>.</p>
```

Etiquetas básicas
Marcado semántico

http://www.w3schools.com/html/html_quotation_elements.asp

- Etiquetas para citar contenido:
  - **<blockquote>** se usa para citas largas, y dentro de ella se incluye el párrafo <p> con el texto a citar
  - **<q>** se utiliza para citas más cortas dentro de un elemento <p>
  - En ambos casos, se puede usar el atributo *cite="URL"* para indicar la procedencia de la cita
  - **<cite>** hace referencia a una obra (libro, película, etc.)
- Etiquetas para definiciones y otro tipo de contenido específico:
  - **<dfn>** se utiliza para indicar la instancia de definición de un nuevo término
  - **<abbr>** indica una abreviación (subrayado discontinuo) sobre la que se puede ver el nombre completo si se pasa el ratón por encima (se mostrará el contenido de su atributo *title=""*)
  - **<address>** contiene detalles de autoría, normalmente se muestra en cursiva
  - **<code>** muestra el contenido con una fuente diferente (tipo consola)

Etiquetas básicas
Bloques y contenedores

https://www.w3schools.com/html/html_blocks.asp

- Los elementos de bloque siempre comienzan en una nueva línea: <h2>, <p>, <ul>,
- Los elementos en línea continúan en la misma línea que sus vecinos: <a>, ,
- Aunque se aplican a cualquier elemento, para los bloques es especialmente relevante utilizar dos atributos globales: *id* y *class*
  - El atributo ***id***
    - Identifica de forma exclusiva el elemento contenedor de otros en la página
    - El valor comienza con una letra o guion bajo
    - La singularidad le permite darle un estilo diferente usando CSS
  - El atributo ***class***
    - Proporciona una forma de identificar y diseñar varios elementos
    - Puede compartir el mismo valor que el atributo de clase en otro elemento
    - Puede separar los nombres de clase por espacio para un elemento

Etiquetas básicas
Bloques y contenedores

https://www.w3schools.com/tags/tag_div.asp

- Existen dos elementos de bloque adicionales que permiten agrupar contenido, ajustando el espacio que ocuparán los elementos que contiene y dándoles una apariencia común:
- Bloque ** **: Agrupa elementos en un cuadro de nivel de bloque. Usado con *class* o *id*:
  - Se puede crear reglas de estilo CSS para el elemento
  - Se puede determinar cuánto espacio ocupa el elemento
  - Se puede cambiar la apariencia de todos los elementos que contiene
- Bloque ****: Actúa como un equivalente en línea del elemento , usualmente para controlar la apariencia del contenido de los elementos que encierra mediante CSS
  - Para contener una sección de texto donde no hay otro elemento adecuado para diferenciar del texto circundante
  - Para contener una serie de elementos en línea
  - Usado con *class* o el atributo *id* para estilizar con CSS o acceder a un grupo de elementos con Javascript

Etiquetas básicas
Imágenes

https://www.w3schools.com/html/html_images.asp
https://www.w3schools.com/tags/tag_figure.asp

- Las imágenes deben ser relevantes y transmitir información complementaria a la página
- Deben ajustarse a la paleta de colores, cumplir licencias y ofrecer texto alternativo
- Se recomienda utilizar un directorio específico para almacenarlas (p. ej. */img*)

- Etiqueta **<img>** para cargar una imagen indicando la ruta (*src*) y el texto alternativo (*alt*)

```
<img src="pic_trulli.jpg" alt="Italian
Trulli“ />
```

Fig.1 - Trulli, Puglia, Italy.

- Etiquetas **<figure>** y **<figurecaption>** (HMTL5) para asociar imágenes con sus títulos

```
<figure>
  <img src="pic_trulli.jpg" alt="Trulli"
style="width:100%">
  <figcaption>Fig.1 - Trulli, Puglia,
Italy.</figcaption>
</figure>
```

Tablas
Estructura

- Elemento para crear una tabla: **<table>**
- El contenido se escribe por filas
- Para crear cada fila, se usa la etiqueta: **<tr>**
- Cada elemento <tr> contiene uno o más elementos de datos de tabla (celdas): **<td>**
- Los datos se insertan en elementos **<td>**
- Para encabezados (fila o columna): **<th>**
  - Ayuda a personas que usan lectores de pantalla (elemento semántico)
  - Los navegadores los muestran en **negrita** y en el centro de la celda

```
<table>
    <tr>
        <th></th>
        <th scope="col">Saturday</th>
        <th scope="col">Sunday</th>
    </tr>
    <tr>
        <th scope="row">Tickets sold:</th>
        <td>120</td>
        <td>135</td>
    </tr>
    <tr>
        <th scope="row">Total sales:</th>
        <td>$600</td>
        <td>$675</td>
    </tr>
</table>
```

https://www.w3schools.com/html/html_tables.asp

Tablas
Formato

- Atributos para combinar celdas:
  - Expansión en más de una columna: *colspan="num"*
  - Expansión en más de una fila: *rowspan="num"*
  - Al expandir, el número de elementos <td> o <th> será menor en esa fila o en las posteriores
- Otras etiquetas para trabajar con tablas grandes:
  - **<thead>**, **<tbody>** y **<tfoot>** ayudan a distinguir entre el contenido principal, la primera fila y la última fila, que pueden tener contenido diferente
  - El encabezado de la tabla debe estar dentro del elemento <thead>
  - El contenido principal de la tabla debe estar dentro del elemento <tbody>
  - El pie de página pertenece al elemento <tfoot>

https://www.w3schools.com/html/html_tables.asp

Thymeleaf
Controles de iteración

- Será frecuente que el controlador envíe a la vista un objeto iterable (array, lista, etc.) con los resultados de una consulta
- En la vista, necesitaremos recorrer dicho objeto para mostrarlo en formato texto, lista o tabla
- Thymeleaf nos proporciona un atributo para iterar sobre un objeto: ***th:each="x : $iterableObject"***
  - **`iterableObject`** será el objeto iterable que queremos recorrer
  - **`x`** será el nombre de la variable que tomará cada elemento del objeto iterable en cada iteración
- El atributo *th:each* debe asociarse al contenedor o bloque que se va a construir dinámicamente (, <ul>, <table>, etc.)
- En cada elemento interno del bloque o contenedor, se accederá al objeto **x** y sus propiedades

https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#iteration

Thymeleaf
Controles de iteración

- Existen variables de estado ("*status variable*") para conocer detalles del proceso iterativo:
  - La propiedad *index* devuelve la posición de la iteración actual (comienza en 0)
  - La propiedad *size* devuelve el total de elementos del objeto iterable
  - La propiedad *current* devuelve el objeto accedido en la iteración actual
  - La propiedad *first* apunta al primer elemento del objeto iterable
  - La propiedad *last* apunta al último elemento del objeto iterable
- Thymeleaf también proporciona métodos auxiliares para conocer propiedades de los objetos:
  - Para arrays: #arrays.*length*(array), #arrays.*isEmpty*(array), #arrays.*contains*(array, element)
  - Para listas: #lists.*size*(list), #lists.*isEmpty*(list), #lists.*contains*(list, element)

https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#iteration
https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#arrays
https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#lists

Thymeleaf
Ejemplo MVC

- Ante la petición **GET**, el controlador `ShowDepartmentsController`: /showDepartments
  a. Busca el número de profesores asociado a cada departamento a través del repositorio
  b. Guarda la información en un nuevo objeto de dominio (`Department`)
  c. Redirecciona a la vista *showDepartmentsView.html*

```java
@GetMapping("/showDepartments")
public ModelAndView showDepartments() {
    this.modelAndView.setViewName(viewName:"showDepartmentsView.html");
    String deptNames [] = new String []{"Maths and Statistics", "Computer Science and AI", "Physics"};
    List<Department> listOfDepartments = new ArrayList<Department>();
    for(String name: deptNames){
        int numProfs = this.professorRepository.getNumberProfessorsInDepartment(name);
        listOfDepartments.add(new Department(name, numProfs));
    }
    this.modelAndView.addObject(attributeName:"listOfDepartments", listOfDepartments);
    return modelAndView;
}
```

```java
3   public class Department {
4       private String name;
5       private int numProfessors;
6
7       public Department(){
8
9       }
10
11      public Department(String name, int numProfessors) {
12          this.name = name;
13          this.numProfessors = numProfessors;
14      }
15
16      public String getName() {
17          return name;
18      }
19
```

Por simplicidad, los nombres de los departamentos no se están leyendo de la base de datos

Thymeleaf
Ejemplo MVC

- Mostrando la información en la vista: *showDepartmentsView.html*
  a. Acceso a propiedades del objeto lista: `.size`
  b. Creación dinámica de la lista no ordenada (<**ul**>) con **th:each**
  c. Cada elemento de la lista accede a las dos propiedades de `Department`

src > main > resources > templates > showDepartmentsView.html > ...

```
1  <!DOCTYPE html>
2  <html   xmlns="http://www.w3.org/1999/xhtml"
3          xmlns:th="http://www.thymeleaf.org">
4      <head>
5          <title>My first application</title>
6          <body>
7              <h1>Departments:</h1>
8              <p th:text="'Number of departments: ' + ${listOfDepartments.size}"></p>
9              <ul th:each="dept : ${listOfDepartments}">
10                 <li th:text="'Name: ' + ${dept.name} +
11                 '  - Num. Professors: ' + ${dept.numProfessors}"></li>
12             </ul>
13         </body>
14     </head>
15 </html>
```


localhost:8080/showDepartments

Departments:

Number of departments: 3

- Name: Maths and Statistics - Num. Professors: 1
- Name: Computer Science and AI - Num. Professors: 2
- Name: Physics - Num. Professors: 1

Thymeleaf
Ejemplo MVC

- Un uso frecuente será mostrar el contenido completo de una entidad de la base de datos utilizando una tabla en la vista
- Ante la petición **GET**, el controlador **`ShowStudentsControllers`**: /showStudents

a) Utiliza el método "*findAllStudents*" ya disponible en `StudentRepository` para obtener la lista de estudiantes

b) Almacena la lista en el objeto `ModelAndView` con el nombre "*listOfStudents*"

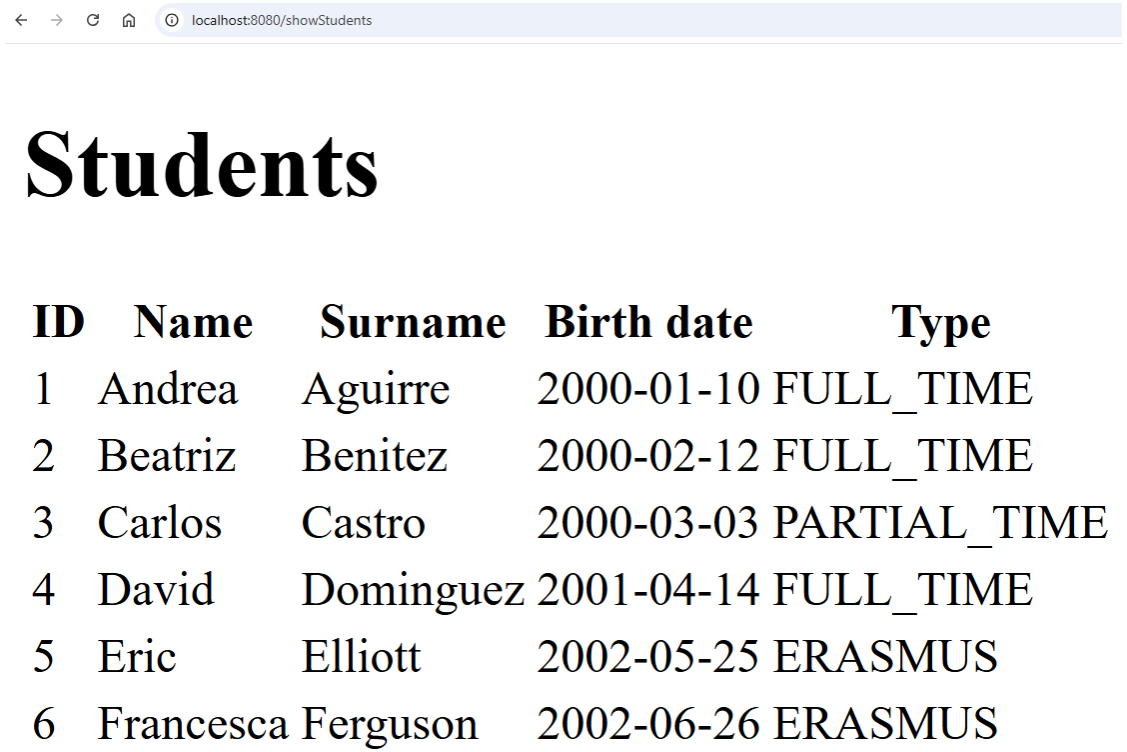
c) Configura el objeto para redirigir a la vista: /showStudentsView.html

```java
@Controller
public class ShowStudentsController {

    StudentRepository studentRepository;
    private ModelAndView modelAndView = new ModelAndView();

    public ShowStudentsController(StudentRepository studentRepository){
        this.studentRepository = studentRepository;
        String sqlQueriesFileName = "./src/main/resources/db/sql.properties";
        this.studentRepository.setSQLQueriesFileName(sqlQueriesFileName);
    }

    @GetMapping("/showStudents")
    public ModelAndView getShowStudentView() {
        this.modelAndView.setViewName(viewName:"showStudentsView.html");
        List<Student> listOfStudents = this.studentRepository.findAllStudents();
        this.modelAndView.addObject(attributeName:"listOfStudents", listOfStudents);
        return modelAndView;
    }
}
```

Thymeleaf
Ejemplo MVC

- En la vista, se replica la estructura de la entidad de la base de datos con encabezados (<th>) para las columnas y rellenando las filas con el contenido de la lista devuelta desde el controlador

```
<body>
    <h1>Students</h1>
    <table>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Surname</th>
            <th>Birth date</th>
            <th>Type</th>
        </tr>
        <tr th:each="student : ${listOfStudents}">
            <td th:text="${student.id}"></td>
            <td th:text="${student.name}"></td>
            <td th:text="${student.surname}"></td>
            <td th:text="${student.birthDate}"></td>
            <td th:text="${student.type}"></td>
        </tr>
    </table>
    <ul >
    </ul>
</body>
```

Thymeleaf
Ejemplo MVC

- Algunos controladores necesitarán que el usuario seleccione un criterio de búsqueda desde la vista
- Ante la petición **GET**, el controlador **`SearchStudentsByTypeController`**: /searchStudentsByType

a) El nombre de la vista se asigna en el constructor para que esté disponible aún cuando no se haya seleccionado nada

b) Si el criterio de búsqueda ha sido asignado (`selectedType`), se invoca al repositorio para realizar la consulta parametrizada

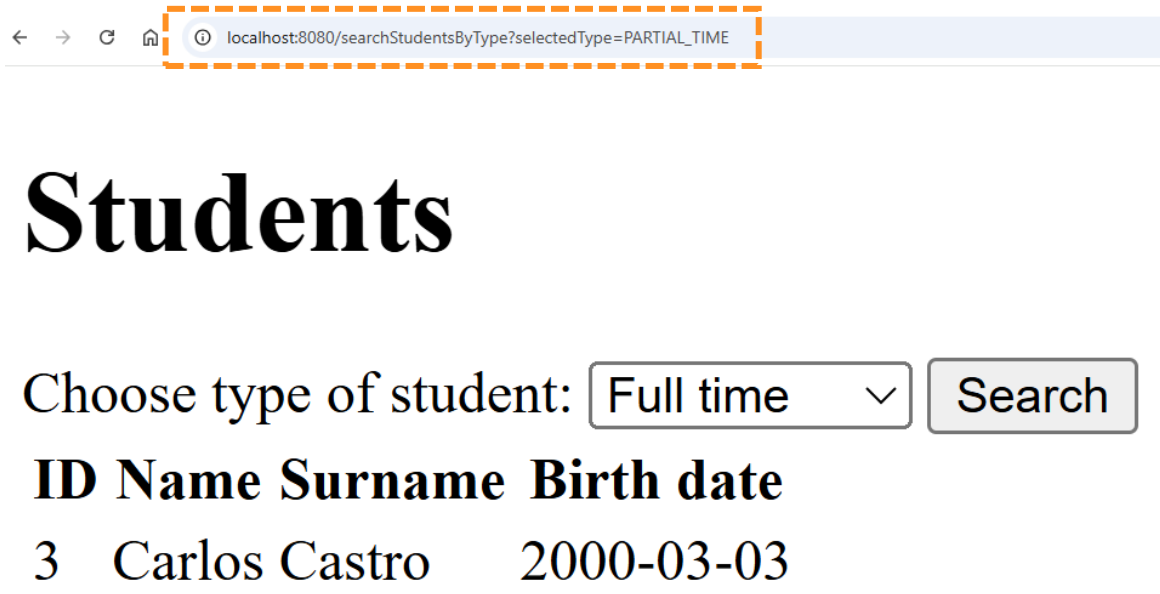
c) Se asigna el objeto a la vista y se redirige al usuario a la misma vista: /searchStudentsByTypeView.html

```java
@Controller
public class ShowStudentsByTypeController {

    StudentRepository studentRepository;
    private ModelAndView modelAndView = new ModelAndView();

    public ShowStudentsByTypeController(StudentRepository studentRepository){
        this.studentRepository = studentRepository;
        String sqlQueriesFileName = "./src/main/resources/db/sql.properties";
        this.studentRepository.setSQLQueriesFileName(sqlQueriesFileName);
        this.modelAndView.setViewName(viewName:"searchStudentsByTypeView.html");
    }

    @GetMapping("/searchStudentsByType")
    public ModelAndView searchStudentsByType(@RequestParam(value = "selectedType", defaultValue = "none") String selectedType){
        if(!selectedType.equals(anObject:"none")){
            List<Student> listOfStudents = this.studentRepository.findStudentsByType(StudentType.valueOf(selectedType));
            this.modelAndView.addObject(attributeName:"listOfStudents", listOfStudents);
        }
        return this.modelAndView;
    }
}
```

Thymeleaf
Ejemplo MVC

- En la vista se combina la selección del tipo de estudiante (<select>) con la tabla de resultados que se actualiza con cada nueva búsqueda
- Puesto que el formulario realiza una petición **GET**, el parámetro de búsqueda es visible en la **URL**

```
<form method="GET" th:action="@{/searchStudentsByType}">
    <label for="types">Choose type of student:</label>
    <select name="selectedType">
        <option value="FULL_TIME">Full time</option>
        <option value="PARTIAL_TIME">Partial time</option>
        <option value="ERASMUS">Erasmus</option>
    </select>
    <input type="submit" value="Search">
</form>
<table>
    <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Surname</th>
        <th>Birth date</th>
    </tr>
    <tr th:each="student : ${listOfStudents}">
        <td th:text="${student.id}"></td>
        <td th:text="${student.name}"></td>
        <td th:text="${student.surname}"></td>
        <td th:text="${student.birthDate}"></td>
    </tr>
</table>
```



Objetivos de la semana

1. Replica el ejemplo explicado en clase, ejecutándolo sobre tu base de datos
   - Versión completa disponible en el repositorio Github: https://github.com/aramirez-uco/pw-examples
2. Incorpora las vistas necesarias para las funciones de **consulta** del enunciado de prácticas:
   - Define las vistas apropiadas según la visualización de datos que indique el enunciado
   - En aquellas vistas donde se muestren listados, utiliza tablas para darle mejor estructura
   - Conecta adecuadamente controladores y vistas, enviando y recibiendo la información necesaria
   - Enlaza las nuevas vistas en los menús o crea otros nuevos si es necesario
3. Consulta la bibliografía recomendada:
   - *HTML tables:* https://www.w3schools.com/html/html_tables.asp
   - *Validador HTML:* https://validator.w3.org/
   - *Thymeleaf:* https://www.thymeleaf.org/
   - *Tutorial Thymeleaf + Spring:* https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html