SUPSI

Bootstrap e Thymeleaf Template Layout

Bootstrap

- È una raccolta di strumenti per la creazione di siti e applicazioni web.
- Contiene modelli di progettazione basati su HTML, CSS e Javascript.
- Per utilizzare Bootstrap è sufficiente aggiungere alle pagine HTML:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
bootstrap/4.1.3/css/bootstrap.min.css" >
```

- Bootstrap è stato sviluppato all'interno di Twitter nel 2011 (version 1) nell'ottica di documentare e condividere modelli di progettazione e attività all'interno della società.

Bootstrap

- Usando Bootstrap possiamo usufruire di un grande lista di componenti (bottoni, forms, liste, menu, barre di navigazione, ecc.) già responsive (che si adattano alle diverse risoluzioni dei device)

- Il file css di Bootstrap continene numerose regole CSS da poter applicare all'interno di applicazioni o siti web

- Altamente personalizzabile https://getbootstrap.com/docs/4.1/getting-started/theming/

- Layout https://getbootstrap.com/docs/4.1/layout

- Tutti i suoi componenti https://getbootstrap.com/docs/4.1/components

Alternative a Bootstrap

- Bulma https://bulma.io/
- Foundation https://foundation.zurb.com/
- Milligram https://milligram.io/
- Pure.css https://purecss.io/
- SemanticUI https://semantic-ui.com/
- Uikit https://getuikit.com/

Bootstrap e Sass

- Il codice sorgente di Bootstrap è scritto **Saas** (Syntactically Awesome StyleSheets, http://sass-lang.com/) nella versione 4

- In precedenza invece veniva usato Less (http://lesscss.org/)

- Sass e Less sono due **preprocessori CSS** che estendono il linguaggio dei fogli di stile con **funzioni**, **operatori** e **variabili**.

- Eesmpio Sass:

```
$background: #fff;

.content-navigation { background-color:$background; }
```

Sass e Scss

- Sass è uno **scripting language** che è interpretato o compilato in CSS.
- Ci sono due possibili sintassi:
  - Sass, quella originale, simile a Yaml. Sintassi indentata. (.sass)
  - Scss, più recente, simile a CSS (.scss)
- Esempio di Scss:

```
$primary-color: #3bbfce;
$margin: 16px;

.content-navigation {
        border-color: $primary-color;
        color: darken($primary-color, 10%);
}
.border {
        padding: $margin / 2;
        margin: $margin / 2;
        border-color: $primary-color;
}
```

Esempio

- Questo file Sass

```
$primary-color: #3bbfce
$margin: 16px

.content-navigation
        border-color: $primary-color
        color: darken($primary-color, 10%)

.border
        padding: $margin / 2
        margin: $margin / 2
        border-color: $primary-color
```

- Compilato diventa

```
.content-navigation {
        border-color: #3bbfce;
        color: darken(#3bbfce, 10%);
}
.border {
        padding: 8px;
        margin: 8px;
        border-color: #3bbfce;
}
```

Funzionalità Sass

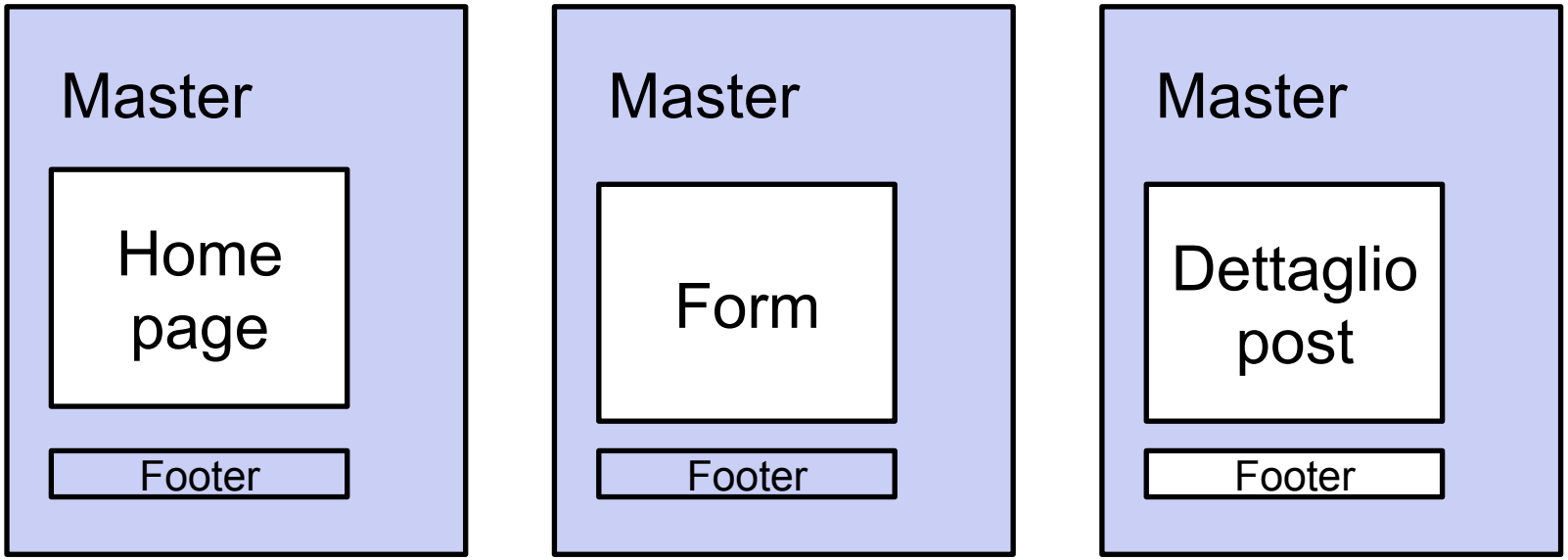
- Sass offre diverse possibilità: varibili, nesting, file parziali, imports, ereditarietà, mixins, operatori

- Per una visione completa: http://sass-lang.com/guide

Ri-utilizzo di codice HTML

Quando scriviamo le pagine HTML di un'applicazione web, nella maggior parte delle volte, tutte le pagine si assomigliano un po': hanno tutte lo stesso footer, header, menu di navigazione, ecc.. Quello che cambia è solo il contenuto nel mezzo della pagina, ma potrebbe anche cambiare il footer, per esempio.

Molto spesso la parte esterna viene chiamata **master page**.



Ri-utilizzo di codice HTML

- Tutto le parti che sono in comune a tutte le pagine HTML, vanno dichiarate all'interno della master page.
- Le pagine HTML dovranno solo dichiarare quale master page utilizzano e sovrascrivere solo le parti che vogliono cambiare.
- In questo modo footer, header, menu di navigazione, ecc. sono scritti solo una volta nella master page e nel caso dovessero essere modificati, verranno modificati in un unico posto.
- Se non ci fosse questo ri-utilizzo di alcune parti, lo sviluppatore sarebbe costretto a modificare, una parte comune a tutte le pagine, in tutte le pagine HTML che hanno quella parte.

Thymeleaf Template Layout

- Per migliorare il ri-utilizzo di codice HTML all'interno delle pagine che utilizzano Thymeleaf possiamo utilizzare i template fragments

- Per poter includere parti da altri template, come footers, headers, menus, ... Thymeleaf permette di definire *frammenti* di codice tramite l'attributo **th:fragment**.

- Ogni pezzo di codice, in qualsiasi pagina, può diventare un template per altre pagine.

Template layout

footer.html

```
<!DOCTYPE html>
<html xmlns:th=http://www.thymeleaf.org>
  <body>
    <div th:fragment="copy">
      &copy; 2018
    </div>
  </body>
</html
```

nome pagina

nome frammento

pagina.html

```
<body>
  <div th:insert="~{footer :: copy}"></div>
</body>
```

pagina.html

```
<body>
  <div th:insert="footer :: copy"></div>
</body>
```

Template layout

- Possiamo includere dei fragments nelle nostre pagine con gli attributi:
  - **th:insert**
  - **th:replace**
  - **th:include**

- Questi attributi vogliono come argomento una ***fragment expression*** **~({.....})**

- Oppure, quando la fragment expression non è complessa, si può usare la stringa

  - NOME_FILE_CHE_CONTIENE_FRAGMENT :: NOME_FRAGMENT

  - Come nell'esempio precedente

Template layout

- Fragment expression:
  - **~{templatename::selector}**
    - Include il frammeto selezionato dal template specificato
      https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html-appendix-c-markup-selector-syntax
  - **~{templatename}**
    - Include tutto il template
  - **~{::selector}**
    - Inserisce un frammento dallo stesso template

Template layout

- Differenze tra th:insert, th:replace e th:include
- **th:insert** inserisce il frammento specificato come corpo del tag
- **th:replace** rimpiazza completamente il tag con il frammento specificato
- **th:include** simile a th:insert, ma inserisce solo il contenuto del frammento come corpo del tag

Template layout

- Esempio:

```
<header th:fragment="head">
   <p>Header</p>
</header>
```

```
<div th:insert="header :: head"></div>
<div th:replace="header :: head"></div>
<div th:include="header :: head"></div>
```

- Il risultato sarà:

```
<div>
   <header>
      <p>Header</p>
   </header>
</div>
<header>
   <p>Header</p>
</header>
<div>
   <p>Header</p>
</div>
```

Template layout

- Frammenti parametrizzabili, simili a una funzione

```
<div th:fragment="frag (var1, var2)">
    <p th:text="${var1} + ' - ' + ${var2}">...</p>
</div>
```

- Possiamo poi invocarla in due modi

```
<div th:replace="::frag (${value1},${value2})">...</div>

<div th:replace="::frag (var1=${value1},var2=${value2})">...</div>
```

Template layout

- Layout flessibili
  - Usando frammenti parametrizzabili e le *fragment expressions* si riescono a creare un meccanismo di layout flessibile

```
<head th:fragment="common_header(title,links)">
  <title th:replace="${title}">Main title</title>
  <link rel="shortcut icon" th:href="@{/images/favicon.ico}”>
  <th:block th:replace="${links}" />
</head>
```

```
<head th:replace="base :: common_header(~{::title},~{::link})">
  <title>Titolo</title>
  <link rel="stylesheet" th:href="@{/css/home.css}">
</head>
```

  - Darà come risultato:

```
<head>
  <title>Titolo</title>
  <link rel="shortcut icon" href="/images/favicon.ico">
  <link rel="stylesheet" href="/css/home.css">
</head>
```

Template layout

- Se vogliamo che una parte del nostro frammento non venga elaborata, possiamo usare un *empty fragment*
  - `<head th:replace="base :: common_header(~{::title},~{})">`
  - Così facendo il blocco di links non sarà visualizzato
- Se vogliamo, invece, che il frammento utilizzi il valore di default possiamo utilizzare il parametro _
  - `<head th:replace="base :: common_header(_,~{::link})">`
  - Così facendo il nostro il title sarà: `<title>Main title</title>`

Template Layout: fragment

```
<html xmlns:th="http://www.thymeleaf.org"
      th:fragment="layout(title, metas, content, footer)">
    <head>
        <meta name="author" content="Marco Bernasconi">
        <th:block th:replace="${metas}" />
        <title th:include="${title}"></title>
    </head>
    <body>
        <main th:include="${content}">
           <section>
              <h2>CONTENT</h2>
           </section>
       </main>
       <footer th:include="${footer}">
           <small>Copyleft</small>
       </footer>
    </body>
</html>
```

Nome file: **master.html**

Template Layout

```
<!DOCTYPE html>
<html xmlns:th=http://www.thymeleaf.org
      th:replace="~{master :: layout(~{::title},~{::meta},~{::main},_)}">
  <head>
    <title>Home</title>
    <meta name="description" content="Pagina Home del sito di Blog">
    <meta name="keywords" content="blog,post">
  </head>
  <body>
    <main>
        <section>
            <h2>I miei blogposts</h2>
            <article th:each="post : ${posts}">
                <p th:text="${post.title}">Titolo</p>
            </article>
        </section>
    </main>
  </body>
</html>
```

Nome file: **home.html**

Template Layout: home.html processato

```
<!DOCTYPE HTML>
<html lang="it">
    <head>
        <meta name="author" content="Marco Bernasconi">
        <meta name="description" content="Pagina Home del sito di Blog">
        <meta name="keywords" content="blog,post”>
        <title>Home</title>
    </head>
    <header>
        <h1>Sito HTML Blogpost</h1>
    </header>
    <main>
        <section>
            <h2>I miei blogposts</h2>
            <article>
                <p>Titolo post</p>
            </article>
        </section>
    </main>
    <footer>
        <small>Copyleft</small>
    </footer>
</html>
```

Quano il file home.html viene processato, il risultato è la pagina master.html con rimpiazzati tutti i fragments che vengono dichiarati nella pagina home.html.

Link utili

- https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#template-layout
- http://getbootstrap.com/docs/4.1/getting-started/introduction/