

ЛАБОРАТОРНО УПРАЖНЕНИЕ №4

Създаване на динамични Web приложения

Цел: Създаване на динамично Web съдържание чрез използване на JavaScript и библиотека jQuery.

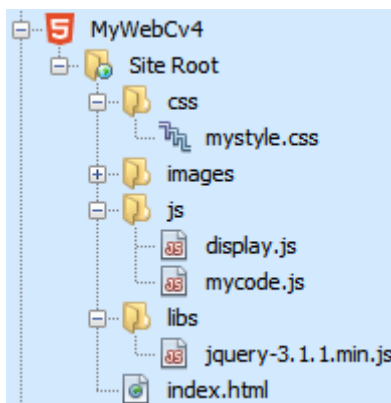
Задача 1: Създайте Web приложение, което ще бъде ваше лично CV. Разширете възможностите на приложението от Лабораторно упражнение №3 със следната функционалност:

- Въведете нова секция с идентификатор "login-box". В тази секция да има HTML5 форма за проверка на автентичността на клиента чрез име и парола. Валидирането на полетата от формата (име и парола) да се реализира чрез използване на възможностите на HTML5 за работа с *регулярни изрази*.
- Създайте папка **libs** в която ще запишете използваните от вас библиотеки. На този етап ще използваме само jQuery.
- Създайте папка **js** в която ще бъде JavaScript кода на вашия проект. Създайте следните файлове в тази папка: 1) **mycode.js** – съдържа функция, инициализира проекта. Тази функция трябва да се активира след зареждане на библиотека jQuery; 2) **display.js** – съдържа функции чрез които може да се скрива и показва съдържанието на различни секции от тялото на етикет **main**.
- След стартиране на приложението да се визуализира само секцията с идентификатор „login-box“. При натискане на бутон „Изпрати“ от login формата да се активира JavaScript функция **login()**, която на този етап да показва в alert box въведените име и паролата на клиента - **alert("низ")**. След това да се визуализира секция с идентификатор „info-box“.



Потърсете в Интернет компресирания код на библиотека jQuery и го запишете в папка **libs**. На този етап ще използваме тази библиотека, за да реализираме достъп до DOM обектите от изгледа на приложението. За целта се създава референция към желанния обект и така програмно се достъпват всички негови свойства и методи. Чрез тази библиотека можем да достопим и CSS свойствата на всеки обект.

Структурата на обновения проект ще бъде следната:



Как се прехваща събитието „Библиотека jQuery е заредена“?

Библиотека jQuery позволява обработка на събития, генерирани от DOM обектите. Тя позволява и създаване на потребителски тип събития. В конкретния случай обаче се интересуваме кога самата библиотека е вече заредена в паметта. Това трябва **ЗАДЪЛЖИТЕЛНО** да се провери, за да сме сигурни, че няма да адресираме jQuery методи, преди библиотеката да е заредена в паметта. **Програмният код JavaScript се изпълнява асинхронно и е възможно вашият код да започне да се изпълнява преди желана от вас библиотека да е заредена в паметта!** Когато библиотека jQuery се зареди в паметта, тя създава специален обект с име \$. Следва пример как може чрез анонимна функция да прехванете кога jQuery е заредена в паметта:

```
$(function () {  
    // код чрез който се инициализира приложението  
});
```

Въведете тази функция във файл mycode.js. В тялото на функцията трябва да напишете програмния код за инициализация на приложението. При конкретната задача това е програмния код чрез който се визуализира секцията с login формата. Нека това да се реализира чрез функция `showLoginForm()`. Тъй като тя е свързана с екрана, кодът ѝ ще бъде част от файл `display.js`. В този файл ще напишем по една подобна `show` функция за всяка една секция от тялото на етикет `main`, например:

```
function view(viewName) {  
    $('main > section').hide();  
    $('# ' + viewName).show();  
}  
  
function showInfo() {  
    view('info-box');  
}  
  
function showLoginForm() {  
    view('login-box');  
    $('#loginForm').trigger('reset');
```

Функция `showInfo()` има за задача да покаже секцията с информацията за студента, а функция `showLoginForm()` – формата за проверка на автентичността. Когато се показва една секция, останалите трябва да се скрият. За тази цел се използва функция `view()`. Чрез jQuery метод `hide()` можем да скрием DOM обект, а чрез метод `show()` – да го покажем. За целта, тези методи се прилагат към желания обект. Той се създава като се указва селектора, който описва обекта, например:

```
$('main > section') // връща референция към всички секции в тялото на main  
$('#login-box') // връща референция към DOM обект с id= login-box
```

След като се покаже формата за авторизация, програмно се изчиства съдържанието на полетата за въвеждане на име и парола:

```
$('#loginForm').trigger('reset');
```

където `loginForm` е идентификатора на формата.

Как да реализираме програмно формата за проверка на автентичността?

Самата форма се създава чрез етикет **form**. За да форматираме стилово формата ще зададем атрибут **class** със стойност **login**. За да получим достъп до DOM обекта-форма ще зададем атрибут **id** със стойност **loginForm**. Всяка форма може да съдържа обекти, чрез които се реализира интерфейса с потребителя: полета за въвеждане на текст или парола; радио бутони, чек боксове, бутони за изчистване на формата и за изпращане на съдържанието към ресурс (локален или глобален програмен код) и много други. Всички тези обекти се създават с един и същ HTML етикет – **input**. Какъв точно обект ще се получи, зависи от стойността на атрибут **type**. Ще използваме следните обекти от формата:

- Поле за въвеждане на името на студента, **type=text**.
- Поле за въвеждане на паролата на студента, **type=password**.
- Бутон за изпращане на въведената информация към ресурс, **type=submit**.

Следва примерно съдържание на секцията с формата е следното:

```
<section id="login-box">
  <div class="titleForm">Форма за авторизация</div>
  <form id="loginForm" class="form" onsubmit="return login();">
    <div>
      <input type="text" name="username" />
    </div>
    <div>
      <input type="password" name="password" />
    </div>
    <div>
      <input type="submit" value="Изпрати"/>
    </div>
  </form>
</section>
```

При конкретният пример, събитието, генерирано при натискане на бутон “Изпрати”, се прехваща чрез атрибут **onsubmit** (етикет **form**). Стойността на този атрибут е стойността, която функцията **login()** ще върне (**true** или **false**).

Как се валидират HTML5 форми?

При HTML5 е възможно валидирането на полета от HTML форми да се реализира чрез *регулярни изрази* (regular expressions). Те се използват в програмирането още от 1960 г. Програмният език JavaScript дава възможност за работа с регулярни изрази по два начина: 1) Използване на конструктора на клас *RegExp* или 2) Използване на израз-литерал. При първият случай на конструктора на клас *RegExp* се предава низ, който се преобразува до регулярен израз, докато при използване на литерали не е необходимо това преобразуване. Поради тази причина е препоръчително да се използват литерали при работа с регулярни изрази.

При формиране на регулярен израз се използват специални символи и последователност от символи. Следва списък (виж Табл. 1) на символите, които се използват с цел указване какво е действието на регулярния израз.

Табл. 1. Символи, управляващи действието на регулярните изрази

Символ	Предназначение
[exp]	Квадратните скоби [] указват да се търси съвпадение за който да е от символите описани чрез exp, например: [AУг] – търси кой да е от символите А, У или г.
[exp1-exp2]	Знакът – между exp1 и exp2 указва да се търси кой да е символ между тези, които се задават чрез exp1 и exp2, например: [3-5] – търсят се цифрите, които са между 3 и 5, включително 3 и 5.
[^exp]	Знакът ^ преди exp указва да се търсят всички символи без тези, които са описани чрез exp, например: При [^абв] или [^а-в] се търсят всички символи без а, б или в.
(exp)	Нормалните скоби () указват, че се търси точно съвпадение на последователност на символи, която се описва чрез exp, например: (куче) – търси се наличието на „куче“
(exp1 exp2)	Знакът между exp1 и exp2 указва, че се търси съвпадение на последователността от символи, която се описва чрез exp1 ИЛИ exp2, например: (куче котка) – търси се или „куче“, или „котка“
{ n }	Търсят се точно n съвпадения за символа (символите) пред скобите. Числото n трябва да е цяло и положително. Например, за низ „крава“: шаблон а{1,} връща „а“, „а“; шаблон а{2,} връща null.
{ n, m }	Числата n и m са цели положителни и $n \leq m$. Търси се съвпадение за символа (символите) пред скобите не по-малко n пъти и най-много m пъти. Ако липсва m се приема, че $m = \infty$. Например, при низ „вардааа“: шаблон а{1,} връща „а“ и „ааа“; а{1,2} връща „а“, „аа“ и „а“; а{1,3} връща „а“, „ааа“; а{2,3} връща „ааа“.
exp+	Знакът + след exp указва да се търси наличието на символ, описан чрез exp, който се среща <u>един или няколко</u> пъти в низа. Като функционалност съвпада с {1,}.
exp*	Знакът * след exp указва, че символът, описан чрез exp трябва да не се среща в низа или да се среща 1 или повече пъти. Като функционалност съвпада с {0,}.
exp?	Знакът ? след exp указва, че символът, описан чрез exp трябва да не се среща в низа или да се среща, но само 1 път. Като функционалност съвпада с {0,1}. Приложен след *,+ или { води до минимизиране на съвпаденията. Например, за низ „проба123“ шаблон \d+ връща „123“, а шаблон \d+? връща „1“, „2“ и „3“.
.exp	Знакът . търси кой да е символ без кода за нов ред. Например, за низ „вардаа“ при шаблон „.а“ се връща „ва“ и „да“; при шаблон „...а“ се връща „рда“.
^exp	Търси в началото на низа наличието на съответния символ (символи), които са зададени чрез exp. Например, шаблон „^а“ за низ „вардар“ връща null, а за низ „арда“ връща „а“.
exp\$	Търси в края на низа наличието на съответния символ (символи), които са зададени чрез exp. Например, шаблон „а\$“ за низ „вардар“ връща null, а за низ „арда“ връща „а“.
x(?=y)	Търси низ x след който следва низ y.
x(?!y)	Търси низ x след който не следва низ y.
\s	Търсят се символи, които се интерпретират като един или няколко интервала, например: интервал (space), табулатор (tab), CR, LF. Например, шаблон „А\sБ“ търси символ А след който има интервален символ последван от Б.
\S	Търсят се всички символи, без тези които се визуализират като един или няколко интервала.
\d	Търсят се символите на всички числа от 0 до 9. Функционално съвпада с шаблон [0-9].
\D	Търсят се всички символи, без тези на числата от 0 до 9. Функционално съвпада с шаблон [^0-9].
\w	Търсят се всички букви, числа и долно подчертаване. Функционално съвпада с шаблон [A-Za-z0-9_]. Например, за низ „\$1.34“ този шаблон връща „1“, „3“ и „4“.
\W	Търсят се символи различни от букви, числа и долно подчертаване. Функционално съвпада с шаблон [^A-Za-z0-9_]. Например, за низ „\$1.34“ този шаблон връща „\$“, и „.“.

При използването на регулярни изрази можете да зададете и така наречения **модификатор**. Основните модификатори, които се използват при HTML5 са следните:

g	Задава търсене на <i>всички</i> съпадения по шаблона (глобално търсене).
i	Задава търсене, като не прави разлика между малки и главни букви (case-insensitive search)

При HTML5 можете да зададете регулярен израз за всяко **input** поле чрез атрибут **pattern**. Нека чрез използване на регулярни изрази да зададем следните ограничения за името и паролата на студента:

- Име на студента – минимум 6 символа, без ограничение какви са символите.
- Парола на студента – минимум 8 символа, задължително да има поне едно число, поне една малка буква и поне една главна буква.

Шаблоните за името и паролата на студентите са следните:

".{6,}" – шаблон за името

"(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}" – шаблон за паролата

Добре е за полетата име и парола на студента да се даде *подсказка* каква информация се очаква да се въведе от клиента. Това се реализира чрез стойността на атрибут **placeholder**. Когато не е изпълнено изискването, зададено чрез регулярния израз може да се покаже някаква помощно съобщение. Това се реализира чрез стойността на атрибут **title**. Следва пример за input полето за въвеждане на име на студента:

```
<input type="text" name="username"
       pattern=".{6,}"
       title="Минимум 6 знака"
       placeholder="Име на потребител"
       required />
```

След стартиране на приложението трябва да получите следния изглед:

ИНФОРМАЦИЯ ЗА МЕН

ФОРМА ЗА АВТОРИЗАЦИЯ

Име на потребител

Парола на потребител

Изпрати

© Име и фамилия, 2021


При некоректно въведени данни трябва да получите:

ИНФОРМАЦИЯ ЗА МЕН

ФОРМА ЗА АВТОРИЗАЦИЯ

xscfbgbfdscccc

....|

 Моля, спазвайте изисквания формат.
Минимум 8 знака, числа и букви (малки и главни)!

© Име и фамилия, 2021