Лабораторная работа №6. jQuery.

Целью данной работы является знакомство с библиотекой jQuery. Теоретическая часть содержит пример разработки приложения, использующего библиотеку jQuery. В практической части предлагается по аналогии выполнить индивидуальное задание.

Теоретическая часть

Рассмотрим приложение Amazeriffic — приложение для хранения списка задач. Его интерфейс состоит из трёх вкладок: первая будет отображать список задач, начиная с самых новых, рисунок 6.1.

Рис. 6.1 Приложение Amazeriffic

Вторая вкладка будет отображать тот же самый список, но начиная с самых старых задач рисунок 6.2.

Рис. 6.2 Приложение Amazeriffic

А на последней вкладке будет находиться поле ввода, с помощью которого можно добавлять новые задачи рисунок 6.3.

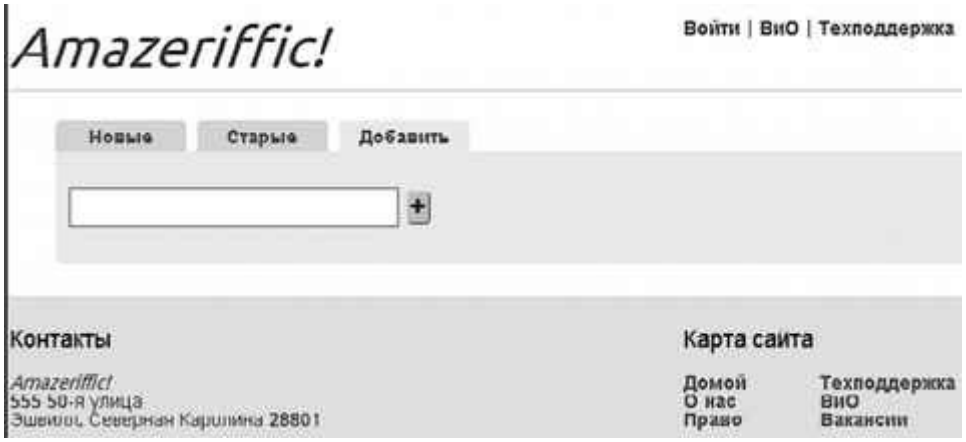

Рис. 6.3 Приложение Amazeriffic

Рассмотрим реализацию клиентской части данного приложения. Следует отметить, что в данной реализации вкладки определены в html, а их заполнение происходит программно с применением библиотеки jQuery.

Пример №1 реализации клиентской части

index.html

```
<!doctype html>
<html>
    <head>
        <title>Простое приложение</title>
        <meta charset="UTF-8">
        <link href="stylesheets/reset.css" rel="stylesheet" type="text/css">
        <link href="stylesheets/style.css" rel="stylesheet" type="text/css">
        <link href='http://fonts.googleapis.com/css?family=Ubuntu'
                  rel='stylesheet' type='text/css'>
        <link href='http://fonts.googleapis.com/css?family=Droid+Sans'
                  rel='stylesheet' type='text/css'>
    </head>
    <body>
        <main>
            <div class="container">
                <div class="tabs">
                    <a href=""><span class="active">Новые</span></a>
                    <a href=""><span>Старые</span></a>
                    <a href=""><span>Добавить</span></a>
                </div>
                <div class="content">
                </div>
            </div>
        </main>
        <footer>
        </footer>
        <script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
        <script src="javascripts/app.js"></script>
    </body>
</html>
```

./stylesheets/reset.css

```
/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
```

```
*/

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
      margin: 0;
      padding: 0;
      border: 0;
      font-size: 100%;
      font: inherit;
      vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
      display: block;
}
body {
      line-height: 1;
}
ol, ul {
      list-style: none;
}
blockquote, q {
      quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
      content: '';
      content: none;
}
table {
      border-collapse: collapse;
      border-spacing: 0;
}
```

./stylesheets/style.css

```
.tabs a span {
    display: inline-block;
    border-radius: 5px 5px 0 0;
    width: 100px;
    margin-right: 10px;
    text-align: center;
    background: #ddd;
    padding: 5px;
}
.tabs a span.active {
    background: #eee;
}
```

```
.content p {
    margin-right: 10px;
    padding: 5px;
    border-radius: 30px;
}
```

./javascripts/app.js

```
var main = function () {
    "use strict";
    var toDos = [
        "Закончить писать эту книгу",
        "Вывести Грейси на прогулку в парк",
        "Ответить на электронные письма",
        "Подготовиться к лекции в понедельник",
        "Обновить несколько новых задач",
        "Купить продукты"
    ];
    $(".tabs a span").toArray().forEach(function (element) {
        $(element).on("click", function () {
            var $element = $(element),
                $content;
            $(".tabs a span").removeClass("active");
            $element.addClass("active");
            $("main .content").empty();

            if ($element.parent().is(":nth-child(1)")) {
                var i;
                $content = $("<ul>");
                for (i = toDos.length; i > -1; i--) {
                    $content.append($("<li>").text(toDos[i]));
                }
            } else if ($element.parent().is(":nth-child(2)")) {
                $content = $("<ul>");
                toDos.forEach(function (todo) {
                    $content.append($("<li>").text(todo));
                });
            } else if ($element.parent().is(":nth-child(3)")) {
                $content = $("<div>");
                $content.append($("<p>").text("Добавьте новую задачу"));
                var $input = $("<input>"),
                    $button = $("<button>");
                $content.append($input);
                $content.append($button.text("+"));
                var addTaskFromInputBox = function () {
                    if ($input.val() !== "") {
                        toDos.push($input.val());
                        $input.val("");
                    }
                };
                $button.on("click", function (event) {
                    addTaskFromInputBox();
                });
                $input.on("keypress", function (event) {
                    if (event.keyCode === 13) {
                        addTaskFromInputBox();
                    }
                });
            }
```

```
            $("main .content").append($content);
            return false;
        });
    });
    $(".tabs a:first-child span").trigger("click");
};

$(document).ready(main);
```

Добавление теговой функциональности в Amazeriffic

К этому моменту мы уже должны научиться работать с объектами JavaScript и JSON, может быть полезно немного попрактиковаться и внедрить некоторые изученные функциональности в приложение Amazeriffic. В этом примере добавим теги для каждого элемента задачи. Мы можем использовать эти теги для сортировки списка задач другим, более удобным способом. В дополнение мы можем инициализировать список задач из файла JSON вместо массива, жёстко помещённого в код.
Добавьте файл JSON, в котором будет находиться список задач. Можно сохранить файл под названием todos.json в основной папке проекта (в той, где находится файл index.html):

todos.json

```
[
    {
        "description": "Купить продукты",
        "tags": ["шопинг", "рутина"]
    },
    {
        "description": "Сделать несколько новых задач",
        "tags": ["писательство", "работа"]
    },
    {
        "description": "Подготовиться к лекции в понедельник",
        "tags": ["работа", "преподавание"]
    },
    {
        "description": "Ответить на электронные письма",
        "tags": ["работа"]
    },
    {
        "description": "Вывести Грейси на прогулку в парк",
        "tags": ["рутина", "питомцы"]
    },
    {
        "description": "Закончить писать книгу",
        "tags": ["писательство", "работа"]
    }
]
```

Вы видите, что файл JSON содержит массив с задачами и каждая задача имеет свой массив с текстовыми элементами, которые и являются тегами. Наша цель — добиться того, чтобы теги работали как вспомогательный способ организации списка задач.
Чтобы достичь этого, нужно добавить немного кода jQuery, который будет читать файл JSON. Но, поскольку функция main , рассмотренная в предыдущей теме, зависит от списка задач, нужно модифицировать её так, чтобы вызов getJSON происходил до вызова main . Для этого

добавим анонимную функцию в вызов document.ready , который будет вызывать getJSON , а затем вызывать main с результатом общей работы:

```
var main = function (toDoObjects) {
"use strict";
// как main имеет доступ к списку задач!
};
$(document).ready(function () {
    $.getJSON("todos.json", function (toDoObjects) {
        // вызов функции main с аргументом в виде объекта toDoObjects
        main(toDoObjects);
    });
});
```

Появилась небольшая проблема: код не будет работать, так как мы изменили структуру объекта со списком задач. Ранее это был массив со строками, содержащими описание задачи, а сейчас это массив объектов. Если нам нужно, чтобы код работал как раньше, можно создать массив старого типа из нового с помощью функции map .

Функция map

Функция map берет массив и делает из него новый, применяя функцию к каждому элементу. Запустите консоль в Chrome и попробуйте следующее:

```
// создаем массив из чисел
var nums = [1, 2, 3, 4, 5];
// применим функцию map
// для создания нового массива
var squares = nums.map(function (num) {
return num*num;
});
console.log(squares);
//=> [1, 4, 9, 16, 25]
```

В этом примере функция, которая возвращает num*num , применяется к каждому элементу для создания нового массива. Этот пример может показаться экзотическим, но посмотрите на другой, более интересный:

```
// создаем массив имен
var names = [ "эмили", "марк", "брюс", "андреа", "пабло" ];
// а сейчас мы создадим массив,
// где первая буква каждого имени прописная
var capitalizedNames = names.map(function (name) {
// получаем первую букву
var firstLetter = name[0];
// возвращаем ее в виде прописной
// в строку, начинающуюся с индекса 1
return firstLetter.toUpperCase() + name.substring(1);
});
console.log(capitalizedNames);
//=> ["Эмили", "Марк", "Брюс", "Андреа", "Пабло"]
```

Сейчас, если вы поняли, как работает функция map, то очень легко создать новый массив из старого:

```
var main = function (toDoObjects) {
      "use strict";
      var toDos = toDoObjects.map(function (toDo) {
            // просто возвращаем описание
```

```
            // этой задачи
            return toDo.description;
      });
      // сейчас весь старый код должен работать в точности как раньше!
      // ...
};
$(document).ready(function () {
      $.getJSON("todos.json", function (toDoObjects) {
      // вызываем функцию main с задачами в качестве аргумента
      main(toDoObjects);
      });
});
```

А сейчас, когда весь старый код работает, как раньше, мы можем создать вкладку Теги .

Добавление вкладки Теги

Начнём с добавления вкладки Теги в пользовательский интерфейс. Поскольку мы сделали код (относительно) гибким, это будет не очень сложно. Начнём с открытия файла index.html и добавления кода для вкладки под названием Теги между вкладками Старые и Добавить :

```
<div class="tabs">
      <a href=""><span class="active">Новые</span></a>
      <a href=""><span>Старые</span></a>
      <a href=""><span>Теги</span></a>
      <a href=""><span>Добавить</span></a>
</div>
```

С помощью этой дополнительной строки мы добавляем вкладку в пользовательский интерфейс. Теперь (почти) все будет работать так, как нужно. Единственная проблема состоит в том, что создание вкладок основывается на их расположении в списке, поэтому, щёлкнув на вкладке Теги , мы увидим интерфейс для вкладки Добавить. Вот это точно не то, что нам нужно, но требует лишь небольших изменений.
Все, что необходимо сделать, — добавить дополнительный блок else-if в середину кода, обеспечивающего работу вкладок, и слегка переставить цифры:

```
} else if ($element.parent().is(":nth-child(3)")) {
      // ЭТО КОД ДЛЯ ВКЛАДКИ ТЕГИ
      console.log("Щелчок на вкладке Теги");
} else if ($element.parent().is(":nth-child(4)")) {
      $input = $("<input>"),
      $button = $("<button>").text("+");
      $button.on("click", function () {
            toDos.push($input.val());
            $input.val("");
      });
      $content = $("<div>").append($input).append($button);
}
```

Создание пользовательского интерфейса

Сейчас, когда мы знаем, как все устроено, подумаем, к чему нужно стремиться на этой вкладке. Посмотрите на рисунок 6.4.
На этой вкладке все имеющиеся теги должны быть перечислены как заголовки, а после них добавлены все описания задач, подходящие под эту категорию. Это значит, что некоторые задачи могут появляться в нескольких местах.

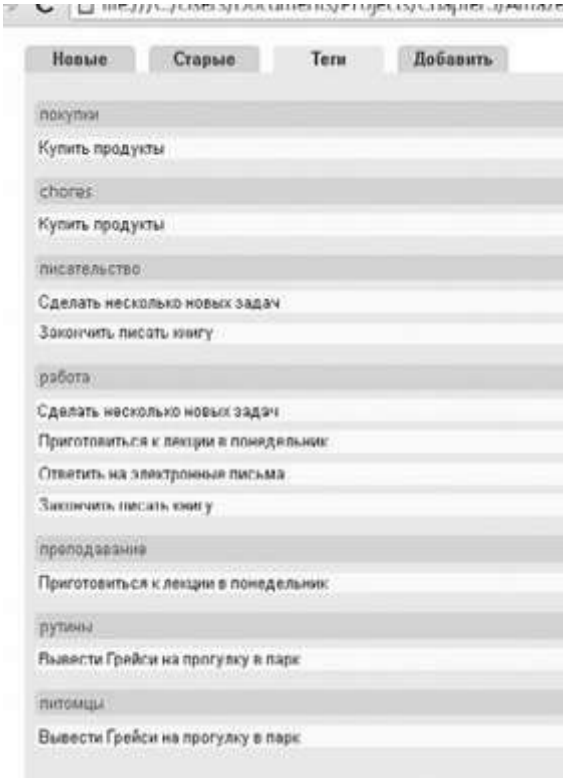

Рис. 6.4 Вкладка с тегами

```
[
    {
        "name": "покупки",
        "toDos": ["Купить продукты"]
    },
    {
        "name": "рутина",
        "toDos": ["Купить продукты", "Вывести Грейси на прогулку в парк"]
    },
    {
        "name": "писательство",
        "toDos": ["Сделать несколько новых задач", "Закончить писать книгу"]
    },
    {
        "name": "работа",
        "toDos": [
                    "Сделать несколько новых задач",
                    "Подготовиться к лекции в понедельник",
                    "Ответить на электронные письма",
                    "Закончить писать книгу"
                ]
    },
    {
        "name": " преподавание",
        "toDos": ["Подготовиться к лекции в понедельник"]
    },
    {
        "name": "питомцы",
        "toDos": ["Вывести Грейси на прогулку в парк "]
    }
]
```

Мы можем изменить изначальный объект toDoObjects и привести его к такому виду с помощью серий вызовов функций map и forEach ! Но пока оставим эту трансформацию для

следующего раздела и сконцентрируемся на создании пользовательского интерфейса. Пока пропишем этот формат (или его упрощённую версию) жёстко в коде для вкладки Теги как переменную под названием organizedByTag :

```
} else if ($element.parent().is(":nth-child(3)")) {
      // ЭТО КОД ДЛЯ ВКЛАДКИ ТЕГИ
      console.log("щелчок на вкладке Теги");
      var organizedByTag = [
      {
            "name": "покупки",
            "toDos": ["Купить продукты "]
      },
      {
            "name": "рутина",
            "toDos": ["Купить продукты", "Вывести Грейси на прогулку в парк "]
      },
      /* и т. д. */
      ];
}
```

Сейчас наша цель — выполнить итерации с этим объектом, добавив новую секцию в элемент страницы .content , как мы уже делали. Для этого просто добавим элемент h3 вместе с ul и li для каждого тега. Элементы ul и li организованы так же, как и ранее, поэтому стиль может остаться тем же самым. Стили для элементов h3 в файле style.css можно оставить прежними, как и в предыдущем примере, или указать собственные:

```
} else if ($element.parent().is(":nth-child(3)")) {
      // ЭТО КОД ДЛЯ ВКЛАДКИ ТЕГИ
      console.log("щелчок на вкладке Теги");
      var organizedByTag = [
      /* и т. д.*/
      ]
      organizedByTag.forEach(function (tag) {
            var $tagName = $("<h3>").text(tag.name),
            $content = $("<ul>");
            tag.toDos.forEach(function (description) {
                  var $li = $("<li>").text(description);
                  $content.append($li);
            });
            $("main .content").append($tagName);
            $("main .content").append($content);
      });
}
```

Осталось проделать всего две вещи. Нужно модифицировать вкладку Добавить так, чтобы вместе с новой задачей обновлялся и список тегов, а также решить, как преобразовать текущий список задач в список, организованный с помощью тегов. Начнём с последнего. Таким образом наше приложение будет динамически обновляться при добавлении новых элементов.

Создание промежуточной структуры данных о тегах

В данном случае нужно создать функцию с названием organizeByTags , которая принимает объект, хранящийся в нашей программе, в таком виде:

```
[
      {
      "description" : "Сделать покупки",
      "tags" : [ "шопинг", "рутина" ]
```

```
        },
        {
        "description" : "Сделать несколько новых задач",
        "tags" : [ "писательство", "работа" ]
        },
        /* и т. д. */
]
```

и преобразует его в объект, выглядящий примерно так:

```
[
        {
        "name": "шоппинг",
        "toDos": ["Сделать покупки"]
        },
        {
        "name": "рутина",
        "toDos": ["Сделать покупки", "Вывести Грейси на прогулку в парк"]
        },
        /* и т. д. */
]
```

После того как у нас появилась функция, мы можем легко модифицировать её и поместить на место жёстко прописанного в коде объекта:

```
} else if ($element.parent().is(":nth-child(3)")) {
        // ЭТО КОД ДЛЯ ВКЛАДКИ ТЕГИ
        console.log("щелчок на вкладке Теги");
        var organizedByTag = organizeByTag(toDoObjects);
}
```

Настройка тестовой платформы

Мы можем проверить результаты нашей работы в консоли Chrome, но постепенно, по мере разрастания и усложнения функций сделать это будет становиться труднее. Предпочтительнее создать внешнюю программу JavaScript, которая содержит функцию и тестирует её, выводя в консоль результат работы. Чтобы сделать это, лучше всего иметь какой-нибудь неиспользуемый файл HTML, находящийся где-то вне вашей файловой иерархии. Этот файл понадобится только для того, чтобы запустить сценарий, в котором мы и будем ставить эксперименты, поэтому HTML не нуждается в нашей стандартной «рыбе», а может содержать лишь следующее:

```
<script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
<script src="test.js"></script>
```

Сейчас, сохранив этот файл как index.html и создав файл test.js следующего содержания:

```
var toDoObjects = [
        {
        "description" : "Сделать покупки",
        "tags" : [ "шопинг", "рутина" ]
        },
        {
        "description" : "Сделать несколько новых задач",
        "tags" : [ "писательство", "работа" ]
        },
        /* etc */
];
```

```
var organizeByTags = function (toDoObjects) {
      console.log("organizeByTags вызвана");
};
var main = function () {
      "use strict";
      var organizeByTags = function () {
            console.log("organizeByTags вызвана");
      };
      organizeByTags(toDoObjects);
};
$(document).ready(main);
```

мы должны увидеть в консоли сообщение «organizeByTag вызвана», а также пример объекта, который настроили. После того как вы справитесь с этим, уделить немного времени самостоятельному решению задачи. Есть несколько способов решения, как часто бывает в компьютерном программировании. Далее показано решение, но лишь в качестве ориентира, чтобы вы поняли, насколько оно отличается от вашего.

Вариант решения

Решение относительно просто объяснить в два этапа. Сначала создаётся массив, который содержит все возможные теги, с помощью перебора изначальной структуры с помощью forEach . После того как будет получен необходимый результат, используется функция map для связи своего массива с тегами с желаемым объектом с помощью перебора списка задач и выделения тех, у которых есть такой тег.
Начнём с первого этапа. Единственная новая для вас вещь здесь — функция indexOf , которая работает с любыми массивами. Мы можем видеть, как это работает, из консоли Chrome:

```
var nums = [1, 2, 3, 4, 5];
nums.indexOf(3);
//=> 2
nums.indexOf(1);
//=> 0
nums.indexOf(10);
//=> -1
```

Упрощенно это можно изложить так: если объект содержится в массиве, функция возвращает индекс этого объекта (все индексы начинаются с 0). Если же в массиве его нет, функция возвращает –1 . Это работает в том числе для строк:

```
var msgs = ["hello", "goodbye", "world"];
msgs.indexOf("goodbye");
//=> 1
msgs.indexOf("hello");
//=> 0
msgs.indexOf("HEY!");
//=> -1
```

Эта функция понадобится нам, чтобы избежать добавления в массив дубликатов:

```
var organizeByTags = function (toDoObjects) {
      // создание пустого массива для тегов
      var tags = [];
      // перебираем все задачи toDos
      toDoObjects.forEach(function (toDo) {
            // перебираем все теги для каждой задачи
            toDo.tags.forEach(function (tag) {
                  // убеждаемся, что этого тега
                  // еще нет в массиве
```

```
                if (tags.indexOf(tag) === -1) {
                    tags.push(tag);
                }
            });
        });
        console.log(tags);
};
```

Запустив этот код, увидим выведенные наименования тегов без дубликатов. Во второй части решения используем функцию map:

```
var organizeByTags = function (toDoObjects) {
        /* первая часть, приведенная выше */
        var tagObjects = tags.map(function (tag) {
            // здесь мы находим все задачи,
            // содержащие этот тег
            var toDosWithTag = [];
            toDoObjects.forEach(function (toDo) {
                // проверка, что результат
                // indexOf is *не* равен -1
                if (toDo.tags.indexOf(tag) !== -1) {
                    toDosWithTag.push(toDo.description);
                }
            });
            // мы связываем каждый тег с объектом, который
            // содержит название тега и массив
            return { "name": tag, "toDos": toDosWithTag };
        });
        console.log(tagObjects);
};
```

Сейчас, когда у нас есть функция и она корректно работает, можем вставить её в функцию main кода приложения, и он будет работать! Эта задача может быть решена различными способами, так что будет полезно испробовать какие-нибудь ещё.

Теги как часть входных данных

Итак, элементы списка задач организовывались и отображаются с помощью тегов, но как добавлять теги к новым элементам, которые вводятся с помощью вкладки Добавить ? Это требует модификации кода, который отвечает за отображение интерфейса вкладки. Его следует преобразовать так, чтобы интерфейс выглядел как на рисунке 6.5.
Там находятся два поля ввода. Во втором из них пользователь может ввести список тегов, разделённых запятыми, которые затем ассоциируются с добавляемым объектом.
Если вы до конца разобрались с предыдущем примером, то ваш код, вероятно, выглядит примерно так:

```
} else if ($element.parent().is(":nth-child(4)")) {
        var $input = $("<input>"),
        $button = $("<span>").text("+");
        $button.on("click", function () {
            toDos.push($input.val());
            $input.val("");
        });
        $content = $("<div>").append($input).append($button);
}
```

Рис. 6.5 Вкладка Добавить с полем Теги

Чтобы пользовательский интерфейс выглядел так, как показано на рис. 6.5, нужно добавить дополнительное поле ввода для тега. Затем мы модифицируем обработчик $button для создания массива с тегами и вставки его в объекты.

Единственное, что нам нужно для этого знать, — как разделить строковый объект. У всех строк есть встроенная функция, называемая split (разделить), которая именно это и делает — разделяет единичную строку на массив строк. Как и в случае с большинством изученных нами функций, её работу можно наблюдать внутри консоли Chrome:

```
var words = "hello,world,goodbye,world";
// это разделяет массив в соответствии с запятыми
var arrayOfWords = words.split(",");
console.log(arrayOfWords);
//=> ["hello","world","goodbye","world"]
arrayOfWords[1];
//=> "world"
arrayOfWords[0];
//=> "hello"
```

Таким образом, пользователь может ввести строку слов, разделённых запятыми, которые являются тегами и могут быть добавлены в объект в виде массива.

И наконец, нужно переработать создание массива toDos из нового массива toDoObjects . Для этого был скопирован и вставлен код, который мы использовали в верхней части функции (нарушив таким образом принцип DRY — подумайте, как можно избежать этого), в результате чего код стал похож на следующий:

```
} else if ($element.parent().is(":nth-child(4)")) {
      var $input = $("<input>").addClass("description"),
            $inputLabel = $("<p>").text("Новая задача: "),
            $tagInput = $("<input>").addClass("tags"),
            $tagLabel = $("<p>").text("Тэги: "),
            $button = $("<span>").text("+");
      $button.on("click", function () {
            var description = $input.val(),
                  // разделение в соответствии с запятыми
                  tags = $tagInput.val().split(",");
            toDoObjects.push({"description":description, "tags":tags});
            // обновление toDos
            toDos = toDoObjects.map(function (toDo) {
                  return toDo.description;
            });
            $input.val("");
```

```
            $tagInput.val("");
      });
}
```

Пример №2 реализации клиентской части с теговой функциональностью

index.html

```
<!doctype html>
<html>
    <head>
        <title>Простое приложение</title>
        <meta charset="UTF-8">
        <link href="stylesheets/reset.css" rel="stylesheet" type="text/css">
        <link href="stylesheets/style.css" rel="stylesheet" type="text/css">
        <link href='http://fonts.googleapis.com/css?family=Ubuntu'
              rel='stylesheet' type='text/css'>
        <link href='http://fonts.googleapis.com/css?family=Droid+Sans'
              rel='stylesheet' type='text/css'>

    </head>
    <body>
        <main>
            <div class="container">
                <div class="tabs">
                    <a href=""><span class="active">Новые</span></a>
                    <a href=""><span>Старые</span></a>
                    <a href=""><span>Теги</span></a>
                    <a href=""><span>Добавить</span></a>
                </div>
                <div class="content">
                </div>
            </div>
        </main>
        <footer>
        </footer>
        <script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
        <script src="javascripts/app.js"></script>
    </body>
</html>
```

todos.json

```
[
    {
        "description": "Купить продукты",
        "tags": ["шопинг", "рутина"]
    },
    {
        "description": "Сделать несколько новых задач",
        "tags": ["писательство", "работа"]
    },
    {
        "description": "Подготовиться к лекции в понедельник",
        "tags": ["работа", "преподавание"]
    },
    {
```

```
        "description": "Ответить на электронные письма",
        "tags": ["работа"]
    },
    {
        "description": "Вывести Грейси на прогулку в парк",
        "tags": ["рутина", "питомцы"]
    },
    {
        "description": "Закончить писать книгу",
        "tags": ["писательство", "работа"]
    }
]
```

./stylesheets/reset.css

```
/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
*/

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
      margin: 0;
      padding: 0;
      border: 0;
      font-size: 100%;
      font: inherit;
      vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
      display: block;
}
body {
      line-height: 1;
}
ol, ul {
      list-style: none;
}
blockquote, q {
      quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
      content: '';
      content: none;
}
table {
```

```
      border-collapse: collapse;
      border-spacing: 0;
}
```

./stylesheets/style.css

```
.tabs a span {
    display: inline-block;
    border-radius: 5px 5px 0 0;
    width: 100px;
    margin-right: 10px;
    text-align: center;
    background: #ddd;
    padding: 5px;
}
.tabs a span.active {
    background: #eee;
}

.content p {
    margin-right: 10px;
    padding: 5px;
    border-radius: 30px;
}

.content li:nth-child(even) {
    background: lavender;
}
.content li:nth-child(odd) {
    background: gainsboro;
}

.content h3 {
    color: blueviolet;
}
```

./javascripts/app.js

```
var main = function (toDoObjects) {
    "use strict";
    var toDos = toDoObjects.map(function (toDo) {
        return toDo.description;
    });
    var organizeByTags = function (toDoObjects) {
        //console.log("organizeByTags вызвана - 0");
        var toArr = [];
        var toTags = [];
        toDoObjects.forEach(function (e1) {
            //console.log(e);
            e1.tags.forEach(function (tag) {
                if (toArr.indexOf(tag) === -1) {
                    //console.log(tag);
                    toArr.push(tag);
                    toTags.push({
                        "name": tag,
                        "toDos": [e1.description]
                    });
                } else {
                    toTags.forEach(function (e2) {
```

```
                        if (e2.name === tag) {
                            e2.toDos.push(e1.description);
                        }
                    });
                }
            });
        });
        //console.log(toTags);
        return toTags;
    };
/*    var toDos = [
        "Закончить писать эту книгу",
        "Вывести Грейси на прогулку в парк",
        "Ответить на электронные письма",
        "Подготовиться к лекции в понедельник",
        "Обновить несколько новых задач",
        "Купить продукты"
    ];*/
    $(".tabs a span").toArray().forEach(function (element) {
        $(element).on("click", function () {
            var $element = $(element),
                $content;
            $(".tabs a span").removeClass("active");
            $element.addClass("active");
            $("main .content").empty();

            if ($element.parent().is(":nth-child(1)")) {
                var i;
                $content = $("<ul>");
                for (i = toDos.length; i > -1; i--) {
                    $content.append($("<li>").text(toDos[i]));
                }
            } else if ($element.parent().is(":nth-child(2)")) {
                $content = $("<ul>");
                toDos.forEach(function (todo) {
                    $content.append($("<li>").text(todo));
                });
            } else if ($element.parent().is(":nth-child(3)")) {
                console.log("Щелчок на вкладке Теги");
                /*                 var organizedByTag = [
                 {
                 "name": "покупки",
                 "toDos": ["Купить продукты"]
                 },
                 {
                 "name": "рутина",
                 "toDos": ["Купить продукты", "Вывести Грейси на прогулку в
парк"]
                 },
                 {
                 "name": "писательство",
                 "toDos": ["Сделать несколько новых задач", "Закончить писать
книгу"]
                 },
                 {
                 "name": "работа",
                 "toDos":
                 [
                 "Сделать несколько новых задач",
                 "Подготовиться к лекции в понедельник",
                 "Ответить на электронные письма",
                 "Закончить писать книгу"
```

```
                ]
                },
                {
                "name": "преподавание",
                "toDos": ["Подготовиться к лекции в понедельник"]
                },
                {
                "name": "питомцы",
                "toDos": ["Вывести Грейси на прогулку в парк"]
                }
                ];*/
               //console.log(organizeByTags(toDoObjects));
               var organizedByTag = organizeByTags(toDoObjects);
               organizedByTag.forEach(function (tag) {
                   var $tagName = $("<h3>").text(tag.name),
                           $content = $("<ul>");
                   tag.toDos.forEach(function (description) {
                       var $li = $("<li>").text(description);
                       $content.append($li);
                   });
                   $("main .content").append($tagName);
                   $("main .content").append($content);
               });
           } else if ($element.parent().is(":nth-child(4)")) {
               $content = $("<div>");
               $content.append($("<p>").text("Добавьте новую задачу"));
               var $input = $("<input>");
               var $button = $("<button>");
               $content.append($input);
               $content.append("<p>Теги</p>");
               var $input_tags = $("<input>");
               $content.append($input_tags);
               $content.append($button.text("+"));
               var description = $input.val();
               var tags = $input_tags.val().split(",");
               var addTaskFromInputBox = function () {
                   if ($input.val() !== "") {
                       toDos.push($input.val());
                       toDoObjects.push({
                            "description": $input.val(),
                            "tags": $input_tags.val().split(",")
                       });
                       $input.val("");
                       $input_tags.val("");
                   }
               };
               $button.on("click", function (event) {
                   addTaskFromInputBox();
               });
               $input.on("keypress", function (event) {
                   if (event.keyCode === 13) {
                       addTaskFromInputBox();
                   }
               });
               console.log(toDoObjects);
           }
           $("main .content").append($content);
           return false;
       });
   });
   $(".tabs a:first-child span").trigger("click");
};
```

```
$(document).ready(function () {
    $.getJSON("todos.json", function (toDoObjects) {
// вызываем функцию main с задачами в качестве аргумента
        main(toDoObjects);
    });
});
```

Практическая часть

1. Ознакомиться с теоретической частью.
2. Продемонстрировать работу приложений из примеров №1 и №2.
3. Пояснить преподавателю работу приложений из примеров №1 и №2.
4. Получить у преподавателя индивидуальное задание.
5. В соответствии с индивидуальным заданием разработать приложение аналогичное описанному в теоретической части.
6. Приложение должно позволять добавлять некоторые сущности. Например, сотрудников организации.
7. Обеспечить добавление указанных сущностей на вкладке Добавить.
8. Обеспечить отображение добавляемых сущностей на вкладке Старые.
9. Обеспечить отображение добавляемых сущностей на вкладке Новые.
10. Сущность должна иметь указанные свойства. Среди этих свойств одно должно быть многозначным, остальные однозначными.
11. Обеспечить отображение добавляемых сущностей на отдельной вкладке, которая выступает в роли средства их отображения с группировкой по значениям заданного многозначного свойства, то есть по аналогии с отображением по тегам в рассмотренных в теоретической части примерах.
12. Обеспечить отображение добавляемых сущностей на отдельной вкладке, которая выступает в роли средства их отображения с группировкой по значениям задаваемым однозначными свойствами, то есть нужно сделать группировку по двум указанным однозначным свойствам.
13. Рекомендуется отображать сущности на вкладках в виде отдельных стилизованных блоков, которые легко идентифицируются при просмотре.
14. Рекомендуется отображать группы сущностей на вкладках в виде отдельных стилизованных блоков, которые легко идентифицируются при просмотре.
15. Пример.
    Задание № 1 «Справочник сотрудников».
    Сущность — Сотрудник (ФИО, пол[9], *отдел*[10], возраст[11]).
16. Для конструирования содержимого html-документов использовать jQuery.
17. Исходный набор сущностей загружать из JSON-файла с помощью функции getJSON.
    Для предоставления браузеру chrome доступа к файловой системе введите в терминале следующую команду:
    chrome.exe –allow-file-access-from-files
    В Firefox нужно сбросить security.fileuri.strict_origin_policy в false. Для этого нужно зайти в about:config.
    Для Safari выберите в меню `Develop`. В выпадающем меню выберите `Disable Local File Restrictions`.

9 Первое свойство сущности для группировки, выделено подчёркиванием.
10 Выделение курсивов свойства сущности, которое может иметь несколько значений, т. е. это аналог тега.
11 Второе свойство сущности для группировки, выделено подчёркиванием.

Задания на лабораторную работу

1. Справочник сотрудников. Сущность — Сотрудник (ФИО, пол, *отдел*, возраст).
2. Учёт инструментов. Сущность — Инструмент (*производитель*, ФИО ответственного, название инструмента, дата выдачи)
3. Библиотека. Сущность — Книга (дата возврата, ФИО держателя, название, издательство, год издания, *авторы*).
4. Автостоянка. Сущность — Автомобиль (*госномер*, номер парковочного места, дата стоянки, ФИО владельца).
5. Доска объявлений. Сущность — Объявление (дата подачи, автор, цена, *текст объявления*).
6. Регистрация входящих писем. Сущность — Входящее письмо (дата получения, организация отправитель, тип, краткое содержание, *адресаты внутри организации*).
7. Менеджер игр. Сущность — Игра (жанр, производитель, рейтинг, *отзывы*).
8. Регистрация исходящих писем. Сущность — Исходящее письмо (дата отправки, организация получатель, тип, краткое содержание, *авторы внутри организации*[12]).
9. Электронная библиотека документов. Сущность — Документ (дата создания, автор, тип, *название*).
10. Учёт студентов в институте. Сущность — Студент (институт, группа, форма обучения, ФИО, *номер телефона*).
11. Туристические путёвки. Сущность — Путёвка (страна, дата вылета, длительность, количество звёзд, *название отеля*).
12. Салон — парикмахерская. Сущность — Услуга (категория[13], ФИО мастера, название, цена, *дата и время*).
13. Маникюрный салон. Сущность — Услуга (название, ФИО мастера, цена, *дата и время*).
14. Расписание пар №1. Сущность — Занятие (день недели, время, предмет, *лектор*).
15. Расписание пар №2. Сущность — Занятие (день недели, время, *предмет*, вид занятия).
16. Бронирование авиабилетов №1. Сущность — Билет (дата, время, *авиакомпания*, откуда, куда).
17. Бронирование авиабилетов №2. Сущность — Билет (дата, время, *цена*, откуда, куда).
18. Летнее кафе. Сущность — Заказ (дата, столик, *блюдо*, количество гостей).
19. Личные дела. Сущность — Задача (дата, время, *пометка*, название, место).
20. Аптека. Сущность — Рецепт (дата, врач, *лекарство*, название аптеки).
21. Журнал преподавателя. Сущность — Журнал (дата, группа, *ФИО присутствующих*, предмет).
22. Медиа архив. Сущность — Документ (дата архивирования, архиватор, *автор документа*, название).
23. Электронная библиотека. Сущность — Книга (дата возврата, ФИО держателя, название, *издательство*, авторы).
24. Учёт личных расходов. Сущность — Чек (дата, статья, название магазина, *товар*).
25. Садовый инвентарь. Сущность — Инвентарь (производитель, адрес склада, название инвентаря, *описание*).
26. Ветеринарная клиника. Сущность — Медкарта (ФИО владельца животного, питомец, кличка, *причина обращения*).
27. Автомобильные запчасти. Сущность — Запчасть (производитель, страна, код, название, *марка автомобиля*).

12 Из которой письмо отправляют.

13 Стрижка, укладка, покраска и т.п.

28. Художественная галерея. Сущность — Картина (год написания, художник, название, *характеристика*).
29. Прокат видео - кассет, CD, DVD — дисков. Сущность — Носитель (филиал, количество, название альбома, *песня*).
30. Расписание телепередач. Сущность — Телепередача (вид, день недели, время, *описание*).
31. Бюро находок. Сущность — Находка (филиал, ФИО нашедшего, название находки, *описание*).
32. Ювелирный салон. Сущность — Украшение (производитель, адрес магазина, название изделия, *описание*).
33. Салон продажи сотовых телефонов. Сущность — Телефон (производитель, страна, модель, цена, *описание*).