**Лабораторная работа №2**

**Javascript**

Используя стандартные средства Javascript, реализовать один из следующих вариантов (поддержка старых браузеров не обязательна):

|  |  |
| --- | --- |
| **Вариант** | **Задание** |
| 1 | Автоматический конструктор форм |
| 2 | Фоторедактор |
| 3 | Древовидный справочник |
| 4 | Логическая игра «Три в ряд» |
| 5 | Канбан |
| 6 | Развивающая игра «Вспомнить все» |
| 7 | Шаблонизатор документов |
| 8 | Редактор посадочных мест в ресторане |

1. **Автоматический конструктор форм**

Входным параметром является объект описания формы вида:

{

method: “POST”, // атрибут method у выходного <form>

action: “post.php”, // атрибут action у выходного <form>

submit: “Отправить”, // текст элемента <input type=”submit”>

fields: [...] // поля формы

}

где каждый элемент fields имеет вид:

{

name: “phone”, // - имя параметра

label: “Телефон”, // - подпись элемента

type: “text”, // - тип элемента

value: “...” // - значение для checkbox или radio

// Условие отображения. Элемент отображается только, если поля

// формы, указанные в объекте, содержат перечисленные значения

if: {

field1: "1",

field2: "2",

....

}

// Параметры валидации

necessary: true, // - если true, то поле является обязательным

test: /\+?\d+/ // - регулярное выражение проверки значения

minLen: 11, // - мин. длина значения

maxLen: 11, // - макс. длина значения,

}

Перечень типов элементов:

|  |  |  |
| --- | --- | --- |
| Тип элемента | Результирующий элемент | Проверка значения |
| text | <input type=”text”> | Да |
| password | <input type=”password”> | Да |
| checkbox | <input type=”checkbox”> | Нет |
| radio | <input type=”radio”> | Нет |
| textarea | <textarea></textarea> | Да |
| info | <div> с текстом label | Нет |

Необходимо построить форму, содержащую соответствующие поля и кнопку отправки.

При нажатии на кнопку отправки необходимо производить проверку полей форм согласно параметрам валидации. В случае ввода неверных значений выводить сообщение об ошибке, ставить фокус на первом неверном элементе и отменять отправку формы.

**!ВАЖНО:** форма должна генерироваться исключительно с помощью javascript. В HTML-коде изначально не должно быть формы

Пример:

|  |  |
| --- | --- |
| Входной объект | Форма |
| {  method: “POST”,  action: “register.php”,  submit: “Зарегистрироваться”,  fields: [  {  label: "Тип пользователя"  type: "info"  }  {  name: "type",  label: "Юридическое лицо"  type: "radio",  value: "1",  necessary: true  },  {  name: "type",  label: "Физическое лицо"  type: "radio",  value: "2",  necessary: true  },  {  name: "fio"  label: "ФИО"  type: "text",  necessary: true  },  {  name: "ogrn"  label: "ОГРН"  type: "text",  if: {  type: "1"  }  test: /^\d{13}$/  necessary: true  },  {  name: "about"  label: "О себе"  type: "textarea"  }  ]  } | Поле «ОГРН» отображается только, если значение поля “type” равно 1, т.е. выбран вариант «Юридическое лицо»  При отправке формы необходимо проверить, что поля “type”, “fio” не пусты. Если выбрано значение “type” == 1, “ogrn” так же должно быть не пусто и попадать под шаблон /^\d{13}$/, т.е. содержать 13 цифр |

1. **Фоторедактор**

Страница содержит форму со следующими полями:

|  |  |  |
| --- | --- | --- |
| **Тип <input>** | **Название** | **Возможные значения/комментарий** |
| file | Входное изображение. | Допустимые форматы файла: jpeg, png. |
| number | Требуемая ширина | Целое число, больше нуля |
| number | Требуемая высота | Целое число, больше нуля |
| select | Фильтры | При выборе фильтра должна появляться кнопка удаления фильтра и новый выпадающий список для добавления следующего фильтра  При выборе фильтра с параметром (бинаризация) должно появляться дополнительное поле с вводом значения параметра |

**Перечень фильтров:**

1. Перевод в черно-белое (градации серого)
2. Отразить по вертикали
3. Отразить по горизонтали
4. Повернуть на 90 градусов влево
5. Повернуть на 90 градусов вправо
6. Бинаризация по порогу (ввод дополнительного параметра – порога бинаризации. Все пиксели ярче заданного порога должны стать белыми, темнее - черными)

При нажатии на кнопку отправки (либо сразу после выбора файла) необходимо:

1. загрузить выбранную картинку в память (см. FileReader)
2. изменить размер картинки с помощью элемента <canvas>
3. В цикле последовательно применить все выбранные фильтры
4. отобразить полученное изображение на странице и добавить кнопку сохранения полученного файла на компьютер (см. получение URL у <canvas> и атрибут download у ссылки)

Сайт так же должен поддерживать Drag’n’drop файлов либо в спец. отведенную зону, либо на все окно

1. **Древовидный справочник**

Некоторый справочник представлен в виде дерева, каждый узел которого может быть, либо узлом-разделом (тип А), либо узлом с информацией (тип Б).

Узлы типа А имеют вид:

{

title: “Категория”, // заголовок узла

children: [] // список дочерних узлов

}

Узлы типа B имеют вид:

{

title: “Понятие”, // заголовок узла

image: “...”, // (необяз)URL-изображения

text: “...” // текст информации, который не должен отображаться как

// html (перед выводом нужно экранировать), строки в нем

// разделяются символами переноса строки “\n”

}

HTML-код справочника должен генерироваться автоматически. Входным параметром является корневой узел типа А. Необходимо разработать страницу, содержащую древовидный раскрывающийся список разделов справочника (изначально все подразделы должны быть скрыты) и область просмотра. При выборе элемента в списке (по нажатию) в области просмотра должно отобразиться его содержимое, согласно формату:

|  |  |
| --- | --- |
| **Тип А** | **Тип Б** |
| Заголовок H1   * заголовок дочернего узла 1 (активная ссылка, переключающая текущий отображаемый узел) * заголовок дочернего узла 2 * …   Кнопка «Добавить элемент» | Заголовок H1  [Изображение, если есть]  Экранированный текст узла с сохранением переносов строк |

При нажатии на кнопку «Добавить элемент» отобразить всплывающее окно с полями:

1. Тип элемента: «Раздел» или «Информация»
2. Заголовок
3. Если выбран тип «Информация», то должны появиться дополнительные поля:
   1. Ссылка на картинку
   2. textarea для ввода текста
4. Кнопка «Сохранить», при нажатии на которую новый элемент помещается в раздел из которого была нажата кнопка «Добавить элемент»

**!ВАЖНО:** Справочник должен генерироваться исключительно с помощью javascript. В HTML-коде изначально не должно быть справочника



Пример интерфейса

1. **Логическая игра «Три в ряд»**

У игры должно быть два экрана:

1. Игровое поле:

- поле игры (квадратное поле 10x10)

- таймер

- кол-во набранных очков

1. Таблица рекордов

Описание логики игры:

На игровом поле случайным образом раскиданы цветные камни. Игрок может обменять местами два соседних камня. Если оказывается, что 3 и более камней выстроились в ряд, игрок получает очки, камни исчезают и на их место опускаются камни, находившиеся выше. Сверху на место опустившихся камней появляются новые случайно раскрашенные камни.

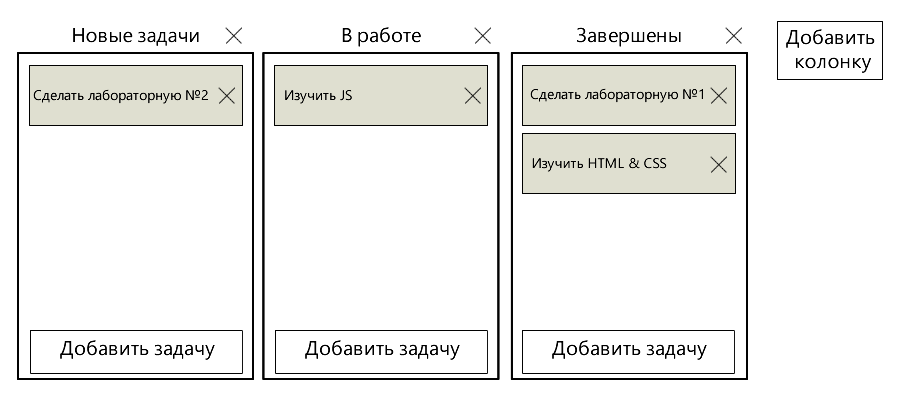
Игра заканчивается, когда таймер достигает нуля.

После завершения нужно спросить имя игрока (можно использовать функцию prompt) и записать результат в таблицу рекордов. Таблица рекордов должна храниться в localStorage

1. **Канбан**

Ваша задача разработать менеджер задач, которые распределяются по отдельным колонкам/стекам.

* Изначально на доске 3 колонки: «Новые задачи», «В работе», «Завершены».
* Пользователь может добавить задачу в любую колонку. При добавлении необходимо спросить текст задачи (используйте ф-цию prompt)
* Пользователь может удалить любую из существующих задач (перед удалением, спросите, действительно ли он это хочет с помощью функции confirm)
* Пользователь может перетаскивать задачи с помощью технологии drag&drop: <https://developer.mozilla.org/ru/docs/Web/Guide/HTML/Drag_and_drop>
* Пользователь может создавать новые колонки (спросите название колонки) и удалять существующие (спросите подтверждение). При удалении колонки должны удалить все ее задачи
* Сохраняйте данные в localStorage, чтобы при перезагрузке задачи оставались на месте



1. **Развивающая игра «Вспомнить все»**

У игры должно быть два экрана:

1. Игровое поле:

- поле игры (квадратное поле 8x8)

- кол-во набранных очков

- число оставшихся попыток

1. Таблица рекордов

Описание логики игры:

Ячейки игрового поля заполнены изображениями случайным образом. Каждое изображение повторяется дважды. Задача игрока на закрытом поле (когда изображения не видны) по памяти отыскивать пары одинаковых изображений.

Перед началом игры игровое поле скрыто. Как только игрок готов, он нажимает кнопку «Начать игру». Ячейки раскрываются на 5 секунд и затем скрываются вновь.

Игрок нажимает на любую ячейку, та показывается и ему нужно по памяти найти такую же картинку в другой ячейке. Для этого он нажимает на другую ячейку, и та тоже вскрывается. Если игрок нашел пару верно, ячейки остаются открытыми, и он зарабатывает одно очко. Если игрок ошибся, то ячейки скрываются и число оставшихся попыток уменьшается на 1. Всего игроку дается 3 попытки.

Игра завершается если число оставшихся попыток становится 0 или он успешно вскрывает все поле (в этом случае он получает дополнительно по 2 очка за каждую сохраненную попытку).

После завершения нужно спросить имя игрока (можно использовать функцию prompt) и записать результат в таблицу рекордов. Таблица рекордов должна храниться в localStorage

1. **Шаблонизатор документов**

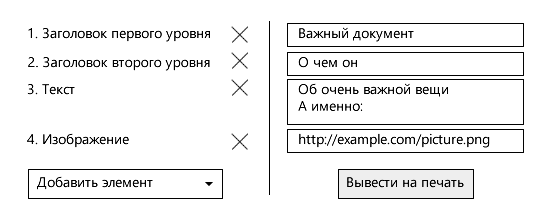
У пользователя на экране есть 2 секции:

* редактор шаблона
* форма заполнения документа

В редакторе шаблона пользователь добавляет элементы, из которых должен состоять документ, а затем в форме заполнения пользователь указывает содержимое этих элементов.

После заполнения документа пользователь может вывести документ на печать. Для этого ниже формы строится документ и вызывается функция document.print(). Чтобы форма не попала в печать скройте ее с помощью медиа-запроса в CSS.

Пример:



Выводимый документ:

**Важный документ**

**О чем он**

Об очень важной вещи.   
А именно:



В таблице указаны элементы, из которых может состоять документ:

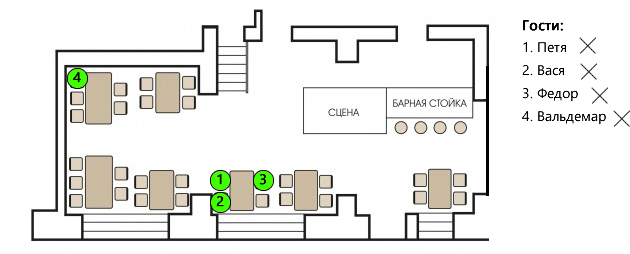
|  |  |  |
| --- | --- | --- |
| **Название элемента** | **Поле для заполнения** | **Генерируемый тег в документе** |
| Заголовок первого уровня | <input type=’text’> | <h1> |
| Заголовок второго уровня | <input type=’text’> | <h2> |
| Текст | <textarea> | текст оборачивается в <p>. Переносы в <textarea> заменяются на <br> |
| Изображение | <input type=’text’> | <img> |

У пользователя должна быть возможность удалить ранее добавленный элемент

1. **Редактор посадочных мест в ресторане**

Необходимо разработать веб-страницу, в которую пользователь загружает схему зала ресторана, а затем отмечает на ней, какой гость где должен сидеть на предстоящем торжестве.

* После загрузки схема зала отображается на экране пользователя
* Чтобы добавить гостя, пользователь нажимает на какую-либо точку схемы и вводит его имя (используйте функцию prompt). На месте нажатия появляется метка с номером гостя. Имя гостя должно отобразиться в списке гостей справа
* Чтобы удалить гостя пользователь или нажимает «крестик» в списке или дважды кликает на метку на схеме. Перед удалением необходимо спросить подтверждение (используйте функцию confirm)
* Должна быть возможность распечатать полученную схему со списком гостей (используйте document.print())



**В отчете:**

1. Постановка задачи
2. Код HTML + JS
3. Скриншоты