Конфигуратор промышленного вентиляционного оборудования

В. В. Осокин, Р. Ф. Алимов, А. А. Абдулпотиев

В статье рассматривается задача разработки конфигуратора промышленного вентиляционного оборудования в виде веб-приложения. Мы реализуем веб-интерфейс, позволяющий собрать в браузере любую приточную, вытяжную и приточно-вытяжную установку простым перетаскиванием секций мышкой или пальцем. Данное решение готово для встраивания в веб-приложение по расчету вентиляционных установок.

Ключевые слова: JavaScript, HTML, CSS, конфигуратор, промышленная вентиляция.

Введение

Большинство заводов-производителей вентиляционного оборудования имеет специальное программное обеспечение, позволяющее производить сложные расчеты вентиляционных установок. Главным их недостатком явялется то, что они настолько сложны, что доступны только для инженеров-экспертов в области вентиляции. Также немаловажный недостаток данного программного обеспечания заключается в том, что оно работает только локально на отдельных машинах. В связи с этим возникает задача разработки простого и удобного решения по конфигурированию вентиляционного оборудования, доступного как из браузера, так и с мобильных устройств с тачскринами.

В статье рассматривается задача разработки конфигуратора промышленного вентиляционного оборудования в виде веб-приложения. В начале описывается постановка задачи, затем идет описание архитектуры решения и затем дается обзор непосредсвенно самого решения.

Постановка задачи

Вентиляционные установки бывают приточными, вытяжными и приточно-вытяжными и, соответсвенно, состоят из приточной части, вытяжной части или из них обеих. В приточной части воздух нагнетается в помещение, а в вытяжной нагнетается из помещения. Договоримся устанавливать потоки друг под другом: приток снизу, вытяжка сверху. Также условимся, что приток нагнетает воздух слева направо, а вытяжка справа налево (рис. 1).

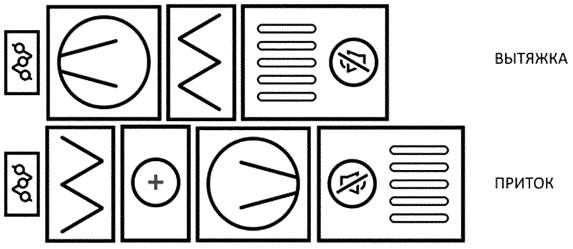

Рис. 1. Приток и вытяжка.

Каждый поток состоит из секций, отвечающих за ту или иную функциональность установки, как нагрев, охлаждение, нагнетание воздуха, рекуперация вытяжного воздуха и прочее. Секции бывают одноэтажные и двухэтажные, которые одновременно располагаются как на притоке, так и на вытяжке. Еще один тип секций — это внешние секции, которые не занимают место непосредственно в корпусе установки, а устанавливаются на других одноэтажных или двухэтажных секциях. Основываясь на приведенных данных, можно выделить следующие типы секций и их свойства.

Одноэтажные (onelevel) секции устанавливаются только на один из потоков. Они могут менять изображение в зависимости от потока. Например, для удобного понимания схемы установки, можно изображать вентилятор как прямоугольник со стрелкой, указывающей на направление нагнетания воздуха.

Двухэтажные (twolevel) секции устанавливаются как на приток, так и на вытяжку. Необходимо выделить класс двухэтажных секций, которые могут перекрещивать потоки: приток и вытяжка меняются местами слева от секции (рис. 2).

Внешние секции (external) устанавливаются либо на одноэтажные, либо на двухэтажные секции и не могут располагаться в установ-

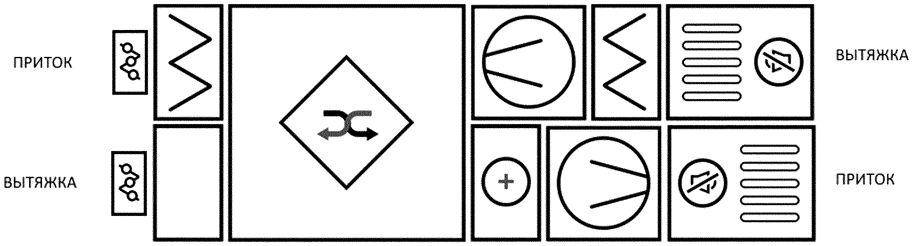

Рис. 2. Двухэтажная секция, меняющая поток.

ке без секции-родителя. Они могут быть установлены либо на один из потоков, либо сверху или снизу одноэтажной или двухэтажной секции (назовем ее секцией-родителем). Рассмотрим их подробнее.

Внешние секции, установленные на один из потоков, при удалении или изменении позиции секции-родителя удаляются, если больше нет других сеций на потоке. Иначе, если есть другие секции, то внешние секции переходят к ним. Всегда располагаются на краю своего потока. То есть внешние секции могут быть либо первыми секциями своего потока, либо последними.

Внешние секции, установленные сверху или снизу секции-родителя, при изменении позиции или при удалении секции-родителя, соответсвенно, меняют свою позицию вместе с секцией-родителем или удаляются. В приточно-вытяжной установке внешние секции не могут быть установлены на верхней части секций нижнего потока и на нижней части верхнего потока. При изменении установки удаляются все внешние секции, которые не могут более существовать.

Рассмотрим требования к конфигуратору.

Конфигуратор должен быть разработан в виде веб-приложения, одинаково хорошо работающего как на десктопах, так и на мобильных устройствах с тачскринами.

Конфигуратор должен быть максимально удобным и простым и напоминать сбор конструктора Lego простым перетаскиванием изображений отдельных секций из списка на поле сбора установки.

При любом изменении конструкции установки (добавление, удаление, изменение позиции секций), конфигуратор должен изменить вид соответствующим образом. Например, при перемещении секции вентилятора из притока на вытяжку необходимо менять изображение данной секции.

Необходимо разработать конфигуратор таким образом, чтобы его можно было встраивать в существующие или новые системы с минимальными доработками. Это означает, что конфигуратор должен быть самодостаточным решением, предоставляющим некоторое API (описывается далее) для системы, в которую он встраивается. Необходимо, чтобы при встраивании в систему не возникала необходимость в изменении кода самого конфигуратора и чтобы можно было в него передавать настройки, применимые непосредственно для данной системы.

Результатом работы конфигуратора являеся код построенной установки. Каждой секции присваивается уникальный идентификатор, называемый GUID. Код установки состоит из кодов потоков, объединенных знаком «+». Код притока начинается на «p^ » и код вытяжки начинается на «v^ » и состоит из кодов секций, объединенных символом «_». Код секции строится по следующему формату: «|обозначение секции|[(для внешней секции — ее позиция: L — слева, R — справа, T — сверху, B — снизу, F — фронт, S — тыл)]–{|GUID секции|[:|GUID секции другого потока, для двухэтажных секций|][<|GUID секции-родителя для внешних секций|]}», где между в символами «|» заключены названия частей, и квадратыми скобками обозначены необязательные части.

Архитектура конфигуратора

В силу того, что конфигуратор должен быть разработан в виде веб-приложения, принято решение реализовать его на языке программирования JavaScript, языке разметки HTML и на каскадных таблицах стилей CSS. В решении использованы библиотеки jQuery [1], jQuery UI [2] (взаимодействия Draggable и Droppable), jQuery Touch Punch [3] (для поддержки мобильных устройств с тачскринами).

Решение разбито на две части: классы Configurator и Drawer. Данное решение принято для того, чтобы при изменении метода отрисовки и взаимодействия с пользователем не было изменений в API конфигуратора и чтобы он продолжал встраиваться в новые или существующие системы тем же способом.

Класс Configurator предоставляет API для встраивания во внешние системы. По умолчанию в данном классе сохранены настройки для секций заслонок (внешняя секция), вентиляторов (одноэтажная, с изменяющейся картинкой в зависимости от потока), пластин-

чатых рекуператоров (двухэтажная, меняющая потоки местами) и прочие. Содержит следующие открытые методы (API): конструктор(externalSettings, container) принимает два параметра. Первый, externalSettings, — это настройки секций, из которых состоит данное встраиваемое решение. Второй, container, — контейнер для встраивания конфигуратора (объект jQuery); getCode() — функция для получения кода построенной установки. Если установка пустая, функция возвращает пустую строку; buildByCode(code) — функция для построения установки по коду, передаваемому через параметр code; empty() — функция для очищения установки.

Класс Drawer, отвечает за непосредственную отрисовку установки и за пользовательское взаимодействие. Класс не содержит открытые методы.

Обзор решения

Конфигуратор состоит из следующих файлов и директорий (за исключением файлов, содержащих библиотеки, перечисленные ранее): файл index.html содержит HTML-разметку для примера встраивания конфигуратора; файл configurator.css содержит каскадные таблицы стилей CSS для примера встраивания конфигуратора; файл configurator.js содержит класс Configurator; файл drawer.js содержит класс Drawer; директория images/sectionImages содержит изображения для примера встраивания конфигуратора.

Далее частично приведем содержимое и описание некоторых файлов.

Файл index.html

Файл index.html содержит HTML разметку для примера внедрения решения.

В начале идет стандартная для всех HTML документов разметка.

```
<!DOCTYPE html>
<html>
<head>
```

Далее кодировка документа устанавливается в значение UTF-8 и подключаются остальные файлы и библиотеки проекта.

```
<meta charset="utf-8">
```

```
  <link href="configurator.css" rel="stylesheet">
  <script src="jquery-2.1.3.min.js"></script>
  <script src="jquery-ui.min.js"></script>
  <script src="jquery.ui.touch-punch.js"></script>
  <script src="drawer.js"></script>
  <script src="configurator.js"></script>
```

После этого располагается разметка видимой части документа, которая после инициализации конфигуратора будет содержать поле списка секций и поле для сборки установки.

```
  <title>Конфигуратор</title>
</head>
<body>
  <div id="content" class="content">
  Здесь будет инициализирован конфигуратор.
  </div>
</body>
<script>
```

Ниже инициализируется объект класса Configurator и в конструктор в качестве первого параметра передается пустой объект, что позволит применить настройки по умолчанию. Вторым параметром передается объект jQuery, соответствующий элементу DIV с аттирбутом ID равным значению content. Здесь и далее договоримся, что запись #elementId будет означать элемент DIV с атрибутом ID равным значению elementID. Внутри данного элемента #content будет размещен конфигуратор.

```
var configurator = new Configurator({}, $('#content'));
</script>
</html>
```

Файл configurator.css

#inner-container содержит все, что относится к примеру встраивания решения. В верхней части #inner-container расположен #droppable-field, а в нижней части расположен #components-field, все перечисленные элементы имеют 100% ширину. Элемент #droppable-field содержит элемент #facility-field, который имеет 50% ширину и 25% отступ слева, что позволяет ему находиться по центру элемента-родителя. Класс .component-image применяется ко всем картинкам секций внутри #components-field, который содержит список секций, из которых может быть построена установка.

```
#inner-container{height: 100%; width: 100%;}
#droppable-field{height: 400px; width: 100%;}
#facility-field{position: absolute; height: 400px; left: 25%; width: 50%;
    background-color: lightgreen;}
#components-field{ height: 100px; width: 100%;}
.component-image{float: left; padding: 10px; background-color: white;}
```

Файл configurator.js

В начале идет объявление класса Configurator и установка переменной self в значение this и атрибута drawer в значение null (далее ему в функции инициализации будет присвоен экземпляр класса Drawer).

```
function Configurator(externalSettings, container){
  var self = this;
  self.drawer = null;
```

Ниже перечисляется список секций, доступных по умолчанию. Настройки каждой секции представляют собой объект, содержащий ключи из некоторого предопределенного набора. Перечислим эти ключи. Ключ title (обязательный) — название секции; ключ type (опциональный, по умолчанию onelevel) — тип секции. Возможные значения: onelevel (одноэтажная), twolevel (двухэтажная) или external (внешняя); ключ img-dir (обязательный) — картинка секции на притоке; ключ img-inv (опциональный) — картинка секции на вытяжке; ключ img-hor (опциональный) — картинка при горизонтальном расположении секции. Ключ задается только для внешних секций; ключ change-direction (опциональный) — флаг-признак, означающий меняет ли секция потоки местами (например, пластинчатый рекуператор).

```
  self.components = {
'ZA':{'title' : 'Заслонка', 'img-dir' : 'za1.png',
      'img-hor' : 'za2.png', 'type' : 'external'},
'FAN':{'title' : 'Вентилятор', 'img-dir' : 'fan.png',
      'img-inv' : 'fanobr.png'},
'PL':{'title' : 'Пластинчатый рекуператор',
      'img-dir' : 'pl.png', 'change-direction' : true, 'type' : 'twolevel'
          },};
```

Далее идут настройки по умолчанию. Настройки содержат путь до директории, в которой хранятся изображения секций (ключ imagePath). По умолчанию используется путь '/images/sectionImages/'.

```
self.options = {
  'imagePath' : '/images/sectionImages/'
};
```

После перечисления значений настроек по умолчанию, идет описание функции инициализации конфигуратора. В самом начале проверяется, был ли передан элемент, в который будет внедрен конфигуратор. В случае, если он не был задан, в консоль разработчика выводится сообщение об ошибке и инициализация конфигуратора прекращается.

```
var initialize = function(){
  if ( typeof container == 'undefined'){
    console.error('Container object is undefined.');
    return;
  }
```

По прохождению предыдущей проверки, необходимо настройки по умолчанию объединить с переданными в конструктор настройками. Приоритет у внешних настроек выше.

```
$.each(externalSettings, function(key, value){
  self.options[key] = value;
});
```

В случае установки флага rewriteComponents в значене true, и при установленном значении ключа components, секции по умолчанию полностью заменяются внешними секциями, переданными в конфигуратор в качестве настроек.

```
if(typeof self.options['rewriteComponents'] != 'undefined' && self.options['
    rewriteComponents'] == true && typeof self.options['components'] != '
    undefined'){
  self.components = self.options['components'];
}
```

В другом случае, если флаг joinComponents установлен в значение true, секции по умолчанию объединяются с внешними секциями, которые были переданы в качестве части настроек конфигуратора.

```
if(typeof self.options['joinComponents'] != 'undefined' && self.options['
    joinComponents'] == true && typeof self.options['components'] != '
    undefined'){
  $.each(self.options['components'],function(key, value){
    self.components[key] = value;
  });
}
```

Далее инициализируется атрибут drawer класса конфигуратора. Его конструктору передается путь директории, содержащей картинки секций, список секций, а также ссылку на функцию, которая будет проверять корректность установки после каждого ее изменения.

```
    self.drawer = new Drawer({'imagePath': self.options['imagePath'], '
        components' : self.components }, container, self.
        checkCorrectOfFacility);
};
```

Функция self.checkCorrectOfFacility проверяет корректность установки. В качестве параметров функция получает массивы секций первого этажа, второго этажа и внешних секций. Задачей данной фукнции является удаление внешних секций, которые не могут более существовать. Правила простые — если больше нет не внешних секций на потоке, то все внешние секции на данном потоке должны быть удалены. По умолчанию считается, что удалять внешние секции не нужно. Для этого переменные deleteFirstLevelExternals (флаг-индиктор небоходимости удаления внешних секций нижнего потока) и deleteSecondLevelExternals (флаг-индиктор небоходимости удаления внешних секций верхнего потока) устанавливаются в значение false.

```
self.checkCorrectOfFacility = function(firstLevel, secondLevel,
    externalSections){
  var deleteFirstLevelExternals = false;
  var deleteSecondLevelExternals = false;
```

Для проверки отсутствия не внешних секций на этаже используется простая проверка на равности длины массива, содержащего секции соответствующего этажа, значению 0.

```
  if(firstLevel.length == 0){
    deleteFirstLevelExternals = true;
  }

  if(secondLevel.length == 0){
    deleteSecondLevelExternals = true;
  }
```

Далее запускается цикл по всем внешним секциям. Если есть необходимость удаления внешних секций потока, на котором располагается внешняя секция, то она удаляется. После прохождения цикла, возвращаются новые значения массивов, содержащих секции.

```
  $.each(externalSections, function(key, value){
    if(value.attr('level-position') == 1 && deleteFirstLevelExternals){
      value.remove();
    }
    if(value.attr('level-position') == 2 && deleteSecondLevelExternals){
      value.remove();
    }
  });
  return {firstLevel: firstLevel , secondLevel: secondLevel, externalSections
      : externalSections};
};
```

Далее вызывается функция инициализации и возвращается переменная self.

```
  initialize();
  return self;
}
```

Файл drawer.js

Для реализации функционала класса Drawer по перемещению секций были использованы взаимодействия Draggable и Droppable. Всем секциям привязывается взаимодействие Draggable, чтобы их можно было перетаскивать мышью или с помощью пальцев на тачскрине. Секции перетаскиваются с панели секций в панель конструктора. У взаимодействия Draggable есть метод helper, в котором создается клон самой секции для перетаскивания.

К панели сборки установки привязано взаимодействие Droppable. При перетаскивании какого-нибудь объекта в эту область, инициализируется событие drop. У данного события можно получить доступ к объекту который перетащили. Далее полученный объект позиционируется в необходимом месте, установка проверятеся на корректность, секции упорядочиваются на этажах и т. д.

В реализации использованы три массива. Массив firstLevelElements используется для хранения секций первого этажа в порядке возрастания; массив secondLevelElements используется для хранения секций второго этажа в порядке возрастания; массив externalElements используется для хранения внешних секций. Порядок в данных массивах определяется по координатам левого края секции в порядке возрастания.

Как было сказано ранее, конструктор класса Drawer в качестве параметров принимает путь директории, содержащей картинки секций, элемент, в который необходимо поместить конфигуратор и фукнцию проверки установки на корректность.

```
function Drawer(options, container, facilityCheckFunction){
```

В классе Drawer опредяются несколько переменных. Переменная scaleDegree — коэффициент масштабирования рисунка. Исходная ширина и высота секции умножается на это число перед позиционированием. Максимальное значение равно 1, и с увеличением длины установки, данный коэффициент будет уменьшаться; переменная hasTwoLevelSection — флаг-признак, означающий является ли установка двухэтажной; переменная hasExternalElement — флаг-признак, означающий наличие или отсутсвие внешних секций; переменная sortedElements — отсортированный массив всех секций.

```
var self = this;
var scaleDegree = 1;
var hasTwoLevelSection = false;
var hasExternalElement = false;
var sortedElements = [];
var firstLevelElements = [];
var secondLevelElements = [];
var externalElements = [];
```

Далее определяются контейнеры для размещения элементов конфигуратора (элемент, содержащий список секций и прочие). Они размещаются в родительский контейнер, передаваемый при инициализации конфигуратора.

```
self.innerContainer=$('<div id="inner-container"></div>')
self.droppableField=$('<div id="droppable-field"></div>')
self. facilityField =$('<div id="facility-field"></div>')
self.componentsField=$('<div id="components-field"></div>')
```

Далее описывается функция инициализации конфигуратора. Она вызывается при создании объекта класса Drawer и инициализирует константы класса, добавляет элементы конфигуратора в родительский контейнер и добавляет секции в панель секций.

```
var initialize = function(){
  self. facilityField .appendTo(self.droppableField);
  self.droppableField.appendTo(container);
  self.componentsField.appendTo(container);
```

Далее к контейнеру droppableField привязывается взаимодействие droppable и в качестве callback функции на событие drop устанавливается функция dropCallback.

```
self.droppableField.droppable({
  drop: dropCallback
});
```

Ниже запускается цикл по всем секциям, в котором для каждой из секций создается элемент на панели секций и к ним привязывается взаимодействие Draggable. Здесь же устанавливаются первоначальные значения атрибутов элементов секций.

```
$.each(options['components'], function(key, value){
  var component = $('<div class="component-image" title="' + value['
      title'] + '"><img class="img-thumbnail" src="' + options['
      imagePath'] + value['img-dir'] + '" /></div>');

    component.draggable({
      helper : function(){
        var img = $('<img src="' + options['imagePath'] + value['img-dir']
            + '" />');
        img.attr('id', 'element-id-' + (elementId++));
        img.attr('title', value['title']);
```

Ниже инициализируется первоначальные параметры в качестве атрибутов. Значения атрибутов получаются из массива компонентов options['components'].

```
        if (typeof value['img-inv'] != 'undefined'){
          img.attr('img-inv', options['imagePath'] + value['img-inv']);
        }
        ...
        return img;
      },
      revert: "invalid"
    });
    component.appendTo(self.componentsField);
  });
};
```

Функция dropCallback вызывается после события drop. То есть после отпускания перетаскиваемого компонента в области Droppable. Данная функция принимает параметры, передаваемые взаимодействием Droppable. Задачей фукции является позиционирование перенесенной секции и установка ее параметров.

```
var dropCallback = function(event, ui){
   ...
}
```

После каждого перемещения секции внутри области Droppable, функция checkingLocation проверяет новое положение секции, и если секция находится далеко от области установки, она удаляется и возвращается значение false. В ином случае просто возвращается значение true. Для краткости опустим реализацию данной фукнции.

```
var checkingLocation = function(element){
   ...
};
```

Функция checkingExternalElementAttributes получает новые значения атрибутов внешней секции и проверяет их на уникальность. Это деляется для того, чтобы две внешних сеции не оказались одна над другой, что недопустимо по определению. Если атрибуты оказались уникальными, возвращается true. Иначе возвращается значение false. Для краткости реализация данной фукнции опущена.

```
var checkingExternalElementAttributes=function(settings){
   ...
};
```

После перестройки установки, внешние секции могут оказаться в вытяжной или в приточной части единственными секциями. Такие секции удаляются функцией removeNotCorrectExternalSections. Для краткости опустим реализацию данной фукнции.

```
var removeNotCorrectExternalSections = function(){
        ...
};
```

Функция calculateLeftForElement вычисляет отступ от левого края области установок для каждой секции, чтобы в дальнейшем позиционировать их правильно. В качестве входных параметров получает позицию секции (elementPosition), определение этажа (isFirstLevel) и массив всех секций (allElements). Для краткости реализация данной фукнции опущена

```
  var calculateLeftForElement = function(elementPosition, isFirstLevel,
      allElements){
      ...
};
```

После каждого изменения в установке, функция calculatingAndPositioning проверяет корректность установки, удаляет некорректные секции, считает новое значение для масштабирования, упорядочивает массивы секций и для каждой секции запускает функцию позиционирования. Для краткости опустим реализацию данной фукнции.

```
var calculatingAndPositioning = function(){
    ...
};
```

Функция positioningElement используется для позиционирования секций с анимацией. В качестве параметров принимает объект jQuery элемента, его позицию (top, left) и размеры (height, width). Для краткости реализация данной фукнции опущена.

```
var positioningElement = function(element, top, left, height, width){
  ...
};
```

Далее идет вызов функции инициализации.

```
  initialize();
  return self;
}
```

Заключение

В статье реализован веб-интерфейс, позволяющий собрать в браузере любую приточную, вытяжную и приточно-вытяжную установку простым перетаскиванием секций мышкой или пальцем. Данное решение готово для встраивания в веб-приложение по расчету вентиляционных установок.

Список литературы

[1] http://jquery.com/

[2] http://jqueryui.com/

[3] http://touchpunch.furf.com/