**Brakhmen Emergency Map Documentation**

1. **Introduction**
2. **Main part**
   1. **Used technologies**
   2. **How to use**
   3. **How it works**
3. **Conclusion**

**Introduction**

Developers of Emergency Map were tasked to implement a combined *visualization* of the data on the example Ministry’s of Emergency Situations of fire accidents statistics.

The product should easily adapt to any sort of data and be a universal tool for statistical purposes.

At the moment, there are services that solve individual tasks of visualizing data. (Example: the criminal map of Minsk, a map of cellular coverage, etc.) However, there is no available ready-made solution that each developer could use to visualize the necessary data in a few clicks.

**Main part**

### **I.Used technologies**

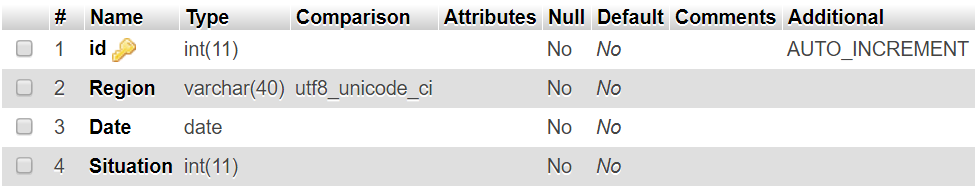
**Languages:** PHP(server part), JS(client part),HTML+CSS(markup + style), SQL(Database requests).

**DBMS:** MySQL

**Third-party APIs:** Yandex map API (map rendering, polygon coloring, polygon click actions), Nominatim API(obtaining requested geographical region polygon).

**Use example:** site brakh.men

### **II.How to use**



Pic 1. Structure of data table

SQL request to create table:

***CREATE******TABLE*** *`TableName` (  
 `id`* ***INT****(100)* ***NOT******NULL****,  
 `Region`* ***VARCHAR****(40)* ***COLLATE*** *utf8\_unicode\_ci* ***NOT******NULL****,  
 `Date`* ***DATE******NOT******NULL****,  
 `Situation`* ***INT****(20)* ***NOT******NULL****,  
 ) ENGINE=InnoDB* ***DEFAULT*** *CHARSET=utf8* ***COLLATE****=utf8\_unicode\_ci;****ALTER******TABLE*** *`TableName`* ***ADD******PRIMARY******KEY*** *(`id`);****ALTER******TABLE*** *`TableName`* ***MODIFY*** *`id`* ***INT****(100)* ***NOT******NULL******AUTO\_INCREMENT****,* ***AUTO\_INCREMENT****=1;  
COMMIT;*

*Database completion guide for developers:*

*Region - the geographical name of the object to be painted (as precisely as possible). Example: Brooklyn, New York City, New York, USA, Earth, Sol, Milky Way, Universe (Unsign)*

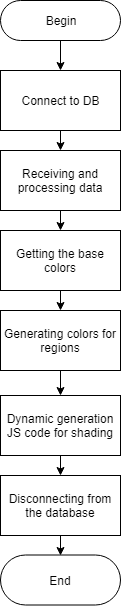
*Date - date, tied to the situation.*

*Situation - situation number. (Each situation is assigned a unique number)*

*Yaer - year of origin of the situation (For convenience of sorting by years)*

*Config.php file have to be filled*

**III. How it works**



Pic 2. Algorithm scheme

**Receiving and processing data:**

At this stage, a query is made to the database

**SELECT** \* **FROM** TableName **WHERE** age = '$age' **ORDER** **BY** Region

on the basis of the extracted data, objects of the class **City**

***class*** *City {* ***var*** *$name;* ***var*** *$color;* ***var*** *$SitArr = array(); // Array of situations. Each N element stores how many times N + 1 was repeated* ***function*** *\_\_construct($n, $name){  
 $this->name = $name;  
 for ($i = 0; $i < $n; $i++) {  
 $arr[] = 0;  
 }  
 $this->color = 0;  
 }  
}*

**Receiving basis colors:**

Regarding the given basis (of situations), the color spectrum is separated and the unique color is assigned to each situation.

**function** getBasicColors ($n) {  
 $colors = array(  
 'red' => 255,   
 'green' => 0,   
 'blue' => 0  
 );  
 $basiccolor = array();   
 $basiccolor[] = $colors;   
   
 $coef = 255\*5.78;  
 $shift = round($coef / $n);  
   
 $trg\_plus = **True**;  
 $status = 'green';  
   
 for ($i = 1; $i < $n; $i++){  
 $exit\_bool= **False**;  
 $delta\_shift = $shift;   
 while(!$exit\_bool) {  
   
 if ($trg\_plus) {  
 while(($delta\_shift != 0) && ( $colors[$status] < 255)) {  
 $colors[$status]++;   
 $delta\_shift--;  
 }  
 }   
 else {  
 while(($delta\_shift != 0) && ( $colors[$status] > 0 )) {  
 $colors[$status]--;   
 $delta\_shift--;  
 }   
 }  
 if ($delta\_shift == 0) {  
 $basiccolor[] = $colors;  
 $exit\_bool = **true**;  
 }   
 else {  
 switch($status) {  
 case 'green':  
 $status = 'red';  
 break;  
 case 'red':  
 $status = 'blue';  
 break;  
 case 'blue':  
 $status = 'green';  
 break;  
 }  
 if ($trg\_plus) {  
 $trg\_plus = **false**;   
 }  
 else {  
 $trg\_plus = **true**;  
 }  
 }  
 }  
 }  
 $newbasiccolor = array();  
 for ($i = 0; $i < count($basiccolor); $i++) {  
 $m = array(  
 'r' => $basiccolor[$i]['red'],  
 'g' => $basiccolor[$i]['green'],  
 'b' => $basiccolor[$i]['blue']  
 );  
 $newbasiccolor[] = rgbToHex($m);  
 }  
 return $newbasiccolor;  
}

**Color generation for every region**

At this stage, the number of repetitions of the necessary situations is already known for each region.

Creation of an array of maximum indicator of the number of situation for all the regions. (for each of the situation)

*Example:*

*Minsk district: 1, 3, 5, 6*

*Maladzyechna district: 3, 4, 1 ,6*

*Borisov district: 0, 9, 0, 1*

*Maximum: 3, 9, 5, 6*

After that for each of the regions their color is determined.

This happens in several stages:

1. A relative color is generated for each situation in the region. The formula for finding the coefficient of "clarification”:

***coef = 100 - (Sit[i] - Max[i])\*100***

where Sit [i] is the number of repetitions of the i-th situation in the current region,

Max [i] - the maximum number of repetitions of the i-th situation for all regions

Then the base color on the coef% is clarified with the help of the procedure:

***function*** *LighterColor($Color, $Percent) {  
 $rgb = array (  
 'r' => 0,   
 'g' => 0,   
 'b' => 0  
 );  
   
 if ($Percent <= 0) {  
 return $Color;  
 }  
 if ($Percent > 100) {  
 $Percent = 100;  
 }  
   
 $Result = hexToRgb($Color);  
 $rgb['r'] = $Result['r'] + round((255 - $Result['r']) \* $Percent / 100);  
 $rgb['g'] = $Result['g'] + round((255 - $Result['g']) \* $Percent / 100);  
 $rgb['b'] = $Result['b'] + round((255 - $Result['b']) \* $Percent / 100);  
   
 return rgbToHex($rgb);  
 }*

1. Mixing the obtained colors of all situations.

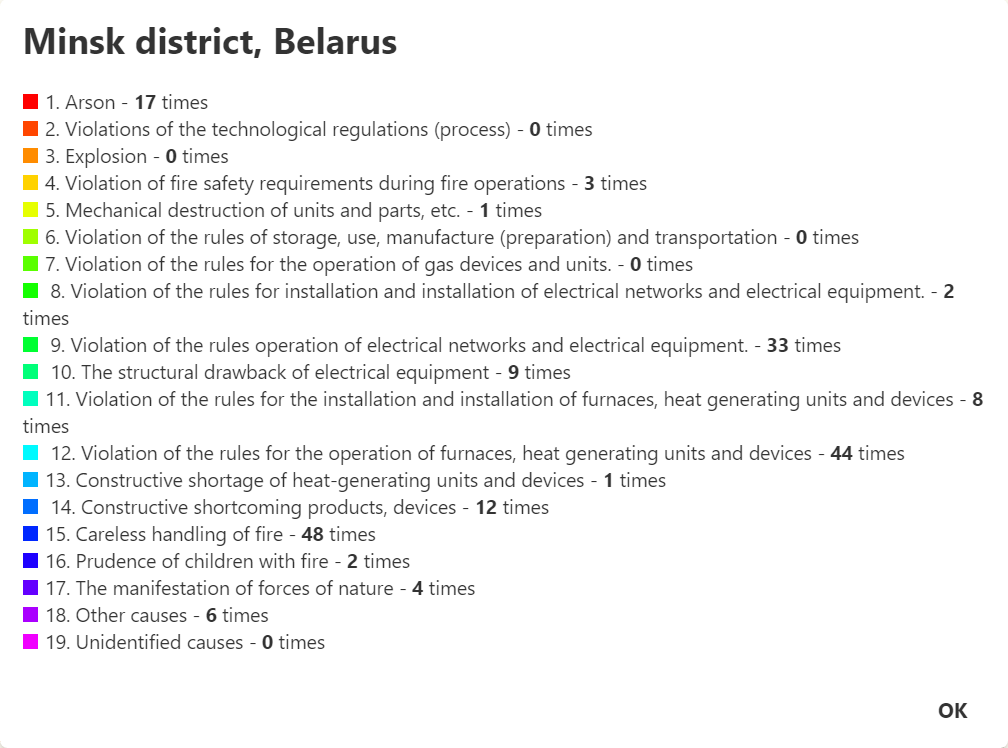
**function** MixColors($Colors)  
 {  
 $rgbsum = array (  
 'r' => 0,   
 'g' => 0,   
 'b' => 0  
 );  
 for ($i = 0; $i < count($Colors); $i++) {  
 $Result = hexToRgb($Colors[$i]);  
 $rgbsum['r'] += $Result['r'];  
 $rgbsum['g'] += $Result['g'];  
 $rgbsum['b'] += $Result['b'];  
 }  
 $rgbsum['r'] = round($rgbsum['r'] / count($Colors));  
 $rgbsum['g'] = round($rgbsum['g'] / count($Colors));  
 $rgbsum['b'] = round($rgbsum['b'] / count($Colors));  
 return rgbToHex($rgbsum);  
 }

**Dynamic generation of JS code for coloring**

On this stage a new object of map is created with Yandex API.

Then function ShowCity(Region, city, color, Desc) are dynamically called in JS code

As Desc operand, situation list and reiteration count in region.

Example: 

Pic 3. Example of Popup description window

$JScode = '';  
for ($i = 0; $i < count($CityList); $i++) {  
$Desc = '';  
*//Reiteration count of each situation*

for ($j = 0; $j < $n; $j++) {  
$count = $CityList[$i]->arr[$j] == 0 ? 0 : $CityList[$i]->arr[$j];  
$Desc = $Desc.'<div> <div style="height: 10px; width: 10px; margin-right: 5px; display: inline-block; background: '.$BasicColors[getRor($j, $ror, $n)].'"></div>';  
$Desc = $mem.getSitName($SituationNameArr, $j, $n, $SitSortArr,$sort).' - <b>'.$count.'</b> раз</div>';  
 }  
$JScode = $JSCode."ShowCity('" .$CityList[$i]->name."','".getColors($CityList[$i]->arr, $BasicColors,$MaxArr, $n, $ror)."', '".$Desc."');"; *// Формируем JS-код, вызывающий функцию, которая будет отрисовывать города/районы на карте*  
}  
echo $JScode;

In its turn, Js-function ShowCity requests Nominatim Api for receiving the polygon of

geographical object.

$.getJSON("https://nominatim.openstreetmap.org/search.php?q=" + Region + "&format=json&polygon\_geojson=1&limit=1")

After that we create colored polygon on the map with Yandex Map API

***var*** *p =* ***new*** *ymaps.Polygon(place.geojson.coordinates, {  
 hintContent: Region // Description on mouse up  
 }, {  
fillColor: color,  
 strokeColor: color,  
 // makes polygon transparent for map events  
 interactivityModel: 'default#transparent',  
 strokeWidth: 2,  
 opacity: 0.7*

*}*

*});*

And assign a handler for the click event on the polygon

(p.events.add('click', function (){ … }); )

**Conclusion**

Developed solution allows to visualize multidimensional information effectively.It has an user-friendly interface. Code is easy to modify for any sphere of usage. Application of color mixing enhances perception and analyzation of information.