Mykyta DERMENZHI [1]

Supervisor: Oleksii FRAZE-FRAZENKO [2]

TWORZENIE APLIKACJI DO ZARZĄDZANIA WYCIECZKAMI PANORAMICZNYMI

Streszczenie: W artykule opisano główne etapy i wdrożenie oprogramowania do tworzenia aplikacji wycieczek panoramicznych. Przeprowadzana jest analiza platform i technologii realizacji zadania, algorytm jest wdrażany i opracowywana jest struktura projektu, opisano główne moduły i funkcje użytkowania. Opracowano autonomiczny moduł do tworzenia i wizualizacji wycieczek panoramicznych ze zoptymalizowanym przechowywaniem danych i wynaleziono algorytm renderowania. Umożliwia to zmianę danych obrazu w czasie rzeczywistym, ładowanie lub usuwanie ich oraz tworzenie nowych panoram. Ponadto ta wersja aplikacji pozwala użytkownikom zaoszczędzić czas na programowaniu i zoptymalizować zasoby niezbędne do obsługi. Platforma została stworzona do zarządzania każdą ramką jako fragmentem całej struktury, składa się z kilku części, takich jak: ASP Core 2.1, JSON Parsers, Serialization, UI Management i Pannelum.

Słowa kluczowe: panorama, ASP Core, JSON, zarządzanie

PANORAMA TOURS MANAGEMENT APPLICATION DEVELOPMENT

Summary: The article describes the main stages and the software implementation of panorama tours application development. The analysis of platforms and technologies for the task implementation is carried out, the algorithm is implemented and the project structure is developed, the main modules and features of use are described. An autonomous module for creating and visualizing panoramic tours with optimized data storage was developed and a rendering algorithm was invented. This allows to change image data in real time, load or delete them, and create new panoramas. In addition, this version of the application allows users to save their time on development and optimize the resources necessary for support. The platform is created to manage every frame as a piece of the whole structure, is composed of several parts, such as: ASP Core 2.1, JSON Parsers, Serialization, UI Management and Pannelum framework.

Keywords: Panorama, ASP Core, JSON, Management

[1] Student of Computer Science, Management and Administration Faculty, Odesa State Environmental University, nikita.dermenzhi@geekgroupllc.com

[2] Engineering Science Ph.D., Odesa State Environmental University, associate professor at the department of information technology, frazenko@gmail.com

1. Introduction

During designing a modern web services and applications now, software developers more often occur necessity to create panorama tours. It can be connected because of marketing goals and such as involving new partners or users into territory particular qualities. And the most important points while creating these systems are procuring easier ways to collect and to process the information. Nowadays, the main examples for panorama tours are Google Panorama's and Facebook 360. But they are involved in the large systems and can't be used like a particular model. Some of the problems while using these systems can be: slow callbacks and non-optimized data. When the project was being started it was decided that the solution must follow three important points: optimized data structure, faster workflow than average large systems and user friendly interface.

To implement everything was planned it was necessary to define which platforms and technologies would be used and was determined that the main platform as framework for web application would be ASP.NET Core. It's needed to note some features were taken.

ASP.NET Core 2.1 is one of the latest update to Microsoft's open source and cross-platform Web framework [1]. Core 2.1 includes loads of new functionality that's also worth checking out:

- Improved build performance
- A turbo-charged HttpClient
- Creation and sharing of .NET Core global tools
- New runtime types for efficient memory usage
- Alpine Linux support
- ARM32 support
- Brotli compression support
- New cryptography APIs and improvements
- SourceLink compatible package building capabilities
- Tiered JIT compilation

Securing Web apps with HTTPS is more important than ever before. Browser enforcement of HTTPS is becoming increasingly strict. Sites that don't use HTTPS can be labeled as insecure. Browsers are also starting to enforce that new and existing Web features must only be used from a secure context. New privacy requirements, like the Global Data Protection Regulation (GDPR), require the use of HTTPS to protect user data. By using HTTPS during development, you're better prepared to run your app under HTTPS in production.

ASP.NET Core 2.1 also includes features for enforcing HTTPS at runtime. You can redirect all HTTP traffic to HTTPS and direct browsers to enforce access to your site over HTTPS using the HTTP Strict Transport Security (HSTS) protocol. The ASP.NET Core project templates enable these HTTPS enforcement features for you [2].

ASP.NET Core helps you write dynamic UI-rendering logic using a mixture of HTML and C# called Razor (cshtml). Razor class libraries are a new feature in ASP.NET Core that let you compile Razor views and pages into reusable class libraries that can be packaged and shared [3]. In internal lab runs at Microsoft, .NET Core 2.1 throughput performance is 10% greater than .NET Core 2.0 for the plain text and JSON scenarios, and a whopping 123% greater for the more realistic data access scenario, solidifying .NET Core's position as one of the fastest frameworks available (fig. 1).

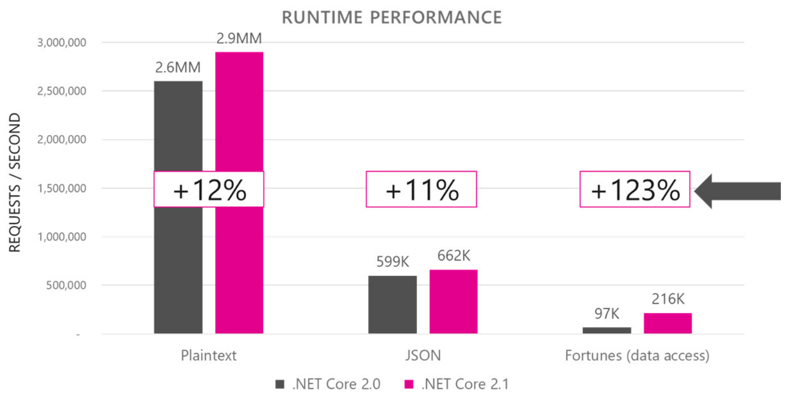


Figure 1. NET Core 2.1 runtime performance on the TechEmpower benchmarks

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition − December 1999. JSON is a text format that is completely language independent (fig. 2) but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language [4].

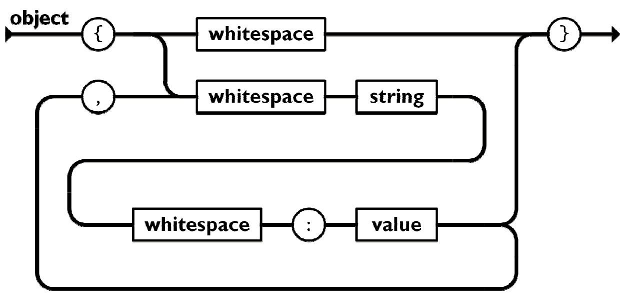


Figure 2. An object in JSON

JSON is built on two structures: a collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array. An ordered list of values, in most languages, this is realized as an array (fig. 3), vector, list, or sequence.

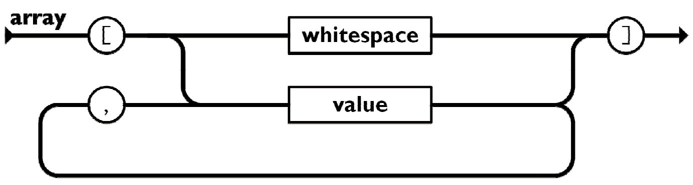


Figure 3. An array in JSON

When it was being selected the way how to store date, it was preferred the shortest format and the fastest reading structure. So, nowadays, JSON includes two points and has one more advantage - it's simply to understand and way to develop data structure using it is simple.
As per above the most valueable framework and data structures was involved in the endpoint project. Topical data saving model, the fastest and the most modern JSON, had been selected at the beginning and developing web framework provided the strong platform for design.

2. Structure and implementation

In computer science, in the context of data storage, serialization (or serialisation) is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer) or transmitted (for example, across a network connection link) and reconstructed later.
When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object. For many complex objects, such as those that make extensive use of references, this process is not straightforward. Serialization of object-oriented objects does not include any of their associated methods with which they were previously linked.
This process of serializing an object is also called marshalling an object. The opposite operation, extracting a data structure from a series of bytes, is deserialization [5].
The quickest method of converting between JSON text and a .NET object is using the JsonSerializer. The JsonSerializer converts .NET objects into their JSON equivalent and back again by mapping the .NET object property names to the JSON property names and copies the values for you.
For simple scenarios where you want to convert to and from a JSON string, the SerializeObject() and DeserializeObject() methods on JsonConvert provide an easy-to-use wrapper over JsonSerializer.

```
Product product = new Product();
product.Name = "Apple";
product.ExpiryDate = new DateTime(2008, 12, 28);
product.Price = 3.99M;
product.Sizes = new string[] { "Small", "Medium", "Large" };
string output = JsonConvert.SerializeObject(product);
//{
//  "Name": "Apple",
//  "ExpiryDate": "2008-12-28T00:00:00",
//  "Price": 3.99,
//  "Sizes": [
//    "Small",
//    "Medium",
//    "Large"
//  ]
//}
Product deserializedProduct =
JsonConvert.DeserializeObject<Product>(output);
```

SerializeObject and DeserializeObject both have overloads that take a JsonSerializerSettings object. JsonSerializerSettings lets you use many of the JsonSerializer settings listed below while still using the simple serialization methods. For more control over how an object is serialized, the JsonSerializer can be used directly. The JsonSerializer is able to read and write JSON text directly to a stream via JsonTextReader and JsonTextWriter [6]. Other kinds of JsonWriters can also be used, such as JTokenReader/JTokenWriter, to convert your object to and from LINQ to JSON objects, or BsonReader/BsonWriter, to convert to and from BSON:

```
Product product = new Product();
product.ExpiryDate = new DateTime(2008, 12, 28);

JsonSerializer serializer = new JsonSerializer();
serializer.Converters.Add(new
JavaScriptDateTimeConverter());
serializer.NullValueHandling = NullValueHandling.Ignore;

using (StreamWriter sw = new StreamWriter(@"c:\json.txt"))
using (JsonWriter writer = new JsonTextWriter(sw))
{
    serializer.Serialize(writer, product);
    // {"ExpiryDate":new Date(1230375600000),"Price":0}
}
```

JsonSerializer has a number of properties on it to customize how it serializes JSON. These can also be used with the methods on JsonConvert via the JsonSerializerSettings overloads.

2.1 Alghoritm's implementation

The project structure is combined of several aspects. Virtually, the project separates for three main parts: admin site, user site and algorithm for storing and creating panorama tours.
While reviewing part for users: it's simply to understand and was clear for developing. The structure is looking like: controllers, middlewares, settings, views, configurations, scene loader and exactly scenes.
In it's turn settings can be expanded with Hotspots, Scene helper and SettingProvider. The main part which builds the full panorama tour is designed by razor pages in cshtml extension. Scenes section is generated by this code:

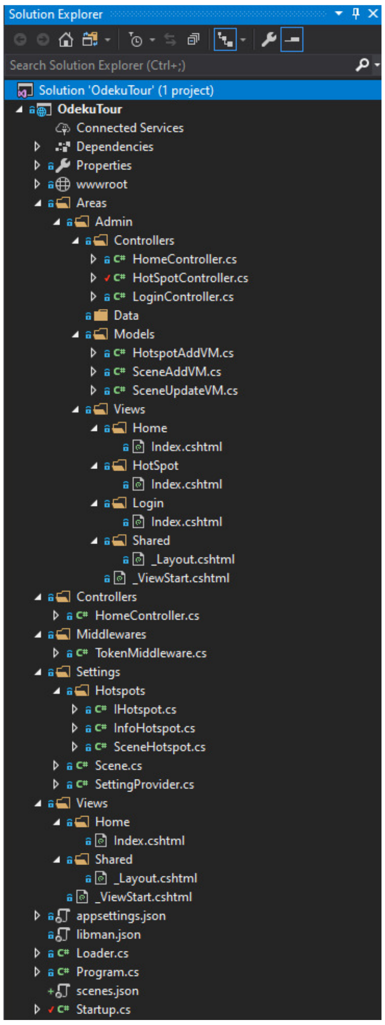
```
'scenes': {
    @foreach (var scene in scenes)
    {

    @: '@scene.Name.GetHashCode()': {
    @:   'panorama': '@scene.Panorama',
    @:   'showFullscreenCtrl':
@scene.ShowFullscreenCtrl.ToString().ToLower(),
    @:   'autoLoad': @scene.AutoLoad.ToString().ToLower(),
    @:   'hfov': @scene.Hfov.ToString().Replace(",", "."),
    @:   'hotSpots':[
    foreach (var hotspot in scene.Hotspots)
```

```
          {
          @:{
          @: 'pitch': @hotspot.Pitch.ToString().Replace(",",
".")
          @: 'yaw': @hotspot.Yaw.ToString().Replace(",",
"."),
          @: 'type': '@hotspot.Type',
          @: 'createTooltipFunc': hotspot,
          @: 'createTooltipArgs': {
          @:      'title': '@hotspot.Title',
          @:      'subtitle': '@hotspot.Subtitle',
          @:      'description': '@hotspot.Description',
          @:      'footer': '@hotspot.Footer',
          @: },
     if(hotspot is InfoHotspot infoHotspot)
          {
          @: 'URL': '@infoHotspot.URL',
          }
          else if (hotspot is SceneHotspot sceneHotspot)
          {
          @: 'clickHandlerFunc': hotspotClickHandler,
          @: 'clickHandlerArgs': { 'id':
'@sceneHotspot.SceneId.GetHashCode()' },
          @: 'sceneId': '@sceneHotspot.SceneId.GetHashCode()'
          }
          @:},
          }
     @: ]},
}
```

Pointer (or hotspot) builder was developed for creating card info every time we select it. It looks like:

```
function hotspot(hotSpotDiv, args) {
    $(hotSpotDiv).mouseenter(function (e) {
        $('#description.card>.card-body>.card-
title').text("").append(args.title);
        $('#description.card>.card-body>.card-
subtitle').text("").append(args.subtitle);
        $('#description.card>.card-body>.card-
text').text("").append(args.description);
        $('#description.card>.card-footer>.card-
text').append("");
        $('#description.card>.card-footer').remove();
        if (args.footer && args.footer.trim() != '') {
            var  footer  =  '<div  class=\"card-footer\"><p
class=\"card-text text-right text-warning\">'
                + args.footer + '</p></div>';
            $('#description.card').append(footer);
        }
        $('#description').removeClass('fadeIn       animated
fadeOut invisible').addClass('animated fadeIn');
    });
    $(hotSpotDiv).mouseout(function () {
        $('#description').addClass('animated fadeOut');
    });
}
```

There are several scopes in the created application. Actually, it separated on administrators, users and core parts (fig. 4). As per screenshot below you can see the whole structure as in folder from Visual Studio 2017 IDE.



```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'OdekuTour' (1 project)
  OdekuTour
    Connected Services
    Dependencies
    Properties
    wwwroot
    Areas
      Admin
        Controllers
          HomeController.cs
          HotSpotController.cs
          LoginController.cs
        Data
        Models
          HotspotAddVM.cs
          SceneAddVM.cs
          SceneUpdateVM.cs
        Views
          Home
            Index.cshtml
          HotSpot
            Index.cshtml
          Login
            Index.cshtml
          Shared
            _Layout.cshtml
          _ViewStart.cshtml
    Controllers
      HomeController.cs
    Middlewares
      TokenMiddleware.cs
    Settings
      Hotspots
        IHotspot.cs
        InfoHotspot.cs
        SceneHotspot.cs
      Scene.cs
      SettingProvider.cs
    Views
      Home
        Index.cshtml
      Shared
        _Layout.cshtml
      _ViewStart.cshtml
    appsettings.json
    libman.json
    Loader.cs
    Program.cs
    scenes.json
    Startup.cs
```

Figure 4. Project structure

Hotspot click handler is called for changing our current location in our panorama tour:

```
function hotspotClickHandler(obj, args) {
    $('#description').addClass('animated fadeOut');
    $('.nav-link').removeClass('active');
    $('#' + args.id).addClass('active');
}
```

And other helpers which allows user switch between units and change active statuses for them:

```
$('#@main.GetHashCode()').addClass('active').click(function
() {
    viewer.loadScene('@main.GetHashCode()');
    $('.nav-link').removeClass('active');
    $(this).addClass('active');
});
@foreach (var unit in units)
{
    @: $('#@unit.Name.GetHashCode()').click(function() {
    @:     viewer.loadScene('@unit.Name.GetHashCode()');
    @:     $('.nav-link').removeClass('active');
    @:     $(this).addClass('active');
    @: });
}
```

The loader body is also important because it serializes and deserializes all data for our panorama tour. It has static modifier and several methods for adding, updating and reading scenes in the system:

```
public static class Loader
{
    public static IEnumerable<Scene> Scenes { get; private
set; }

    public static string Main => Scenes.FirstOrDefault(scene
=> scene.IsMain)?.Name;

    public     static     IEnumerable<Scene>     Units     =>
Scenes.Where(scene => scene.IsUnit && !scene.IsMain);

    static Loader()
    {
        Scenes = SettingProvider.Deserialize();
        if (Scenes == null)
        {
            Scenes = new List<Scene>();
        }
    }

    public static SettingMessage UpdateScenes()
    {
        return SettingProvider.Serialize(Scenes);
    }

    public               static               SettingMessage
UpdateScenes(IEnumerable<Scene> scenes)
    {
        var message = SettingProvider.Serialize(scenes);
        if (!message.IsError)
        {
            Scenes = scenes;
        }
        return message;
    }
    public static SettingMessage AddScene(Scene scene)
    {
        var newScenes = new List<Scene>(Scenes);
        newScenes.Add(scene);
        var message = SettingProvider.Serialize(newScenes);
        if (!message.IsError)
        {
            Scenes = newScenes;
        }
        return message;
    }

    public static SettingMessage UpdateScene(Scene scene)
    {
        var sceneToUpdate = Scenes.FirstOrDefault(x => x.Name
== scene.Name);
        sceneToUpdate.AutoLoad = scene.AutoLoad;
        sceneToUpdate.Hfov = scene.Hfov;
        sceneToUpdate.Hotspots = scene.Hotspots;
        sceneToUpdate.Image = scene.Image;
        sceneToUpdate.IsMain = scene.IsMain;
        sceneToUpdate.IsUnit = scene.IsUnit;
        sceneToUpdate.Name = scene.Name;
        sceneToUpdate.ShowFullscreenCtrl                   =
scene.ShowFullscreenCtrl;
        var message = SettingProvider.Serialize(Scenes);
        return message;
    }
}
```

Also, it should be described entities SettingProvider and SettingMessage. These classes are used for saving data to files into file system thanks streams.

```
public static class SettingProvider
{
    private const string _path = "scenes.json";
    public static SettingMessage Serialize(IEnumerable<Scene>
scenes)
    {
        SettingMessage  message  =  new  SettingMessage("Сцены
были успешно обновлены!");
        try
        {
            var indented = Formatting.Indented;
            var settings = new JsonSerializerSettings()
            {
                TypeNameHandling = TypeNameHandling.All
            };
            var @string = JsonConvert.SerializeObject(scenes,
indented, settings);
            using (StreamWriter sw = new StreamWriter(_path,
false))
            {
                sw.Write(@string);
            }
        }
        catch (Exception ex)
        {
            message = new SettingMessage("При обновлении сцен
произошла ошибка!", ex);
        }
        return message;
    }
    public static IEnumerable<Scene> Deserialize()
    {
        using    (var    fs    =    new    FileStream(_path,
FileMode.OpenOrCreate))
        {
            using (var sr = new StreamReader(fs))
            {
                var settings = new JsonSerializerSettings()
                {
                    TypeNameHandling = TypeNameHandling.All
                };
                var @string = sr.ReadToEnd();
                var                 scenes                  =
JsonConvert.DeserializeObject<IEnumerable<Scene>>(@string,
settings);
                return scenes;
            }
        }
    }
}

public class SettingMessage
{
    public string Text { get; set; }
    public Exception Exception { get; set; }
    public bool IsError => Exception != null;

    public SettingMessage() { }
    public SettingMessage(string text)
    {
        Text = text;
    }
    public SettingMessage(Exception exception)
    {
        Exception = exception;
        Text = exception.Message;
    }
    public SettingMessage(string text, Exception exception) :
this(text)
    {
        Exception = exception;
    }
}
}
```

Eventually, below (fig. 5) is shown the main project structure with all internal entities, which consists of two parts (view models and processors):

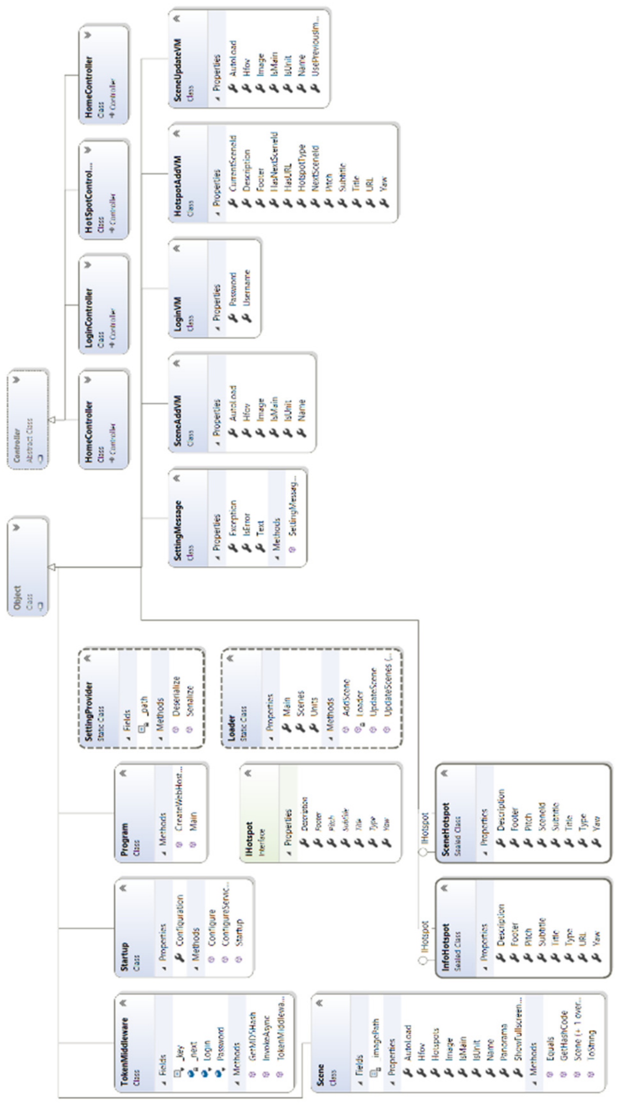
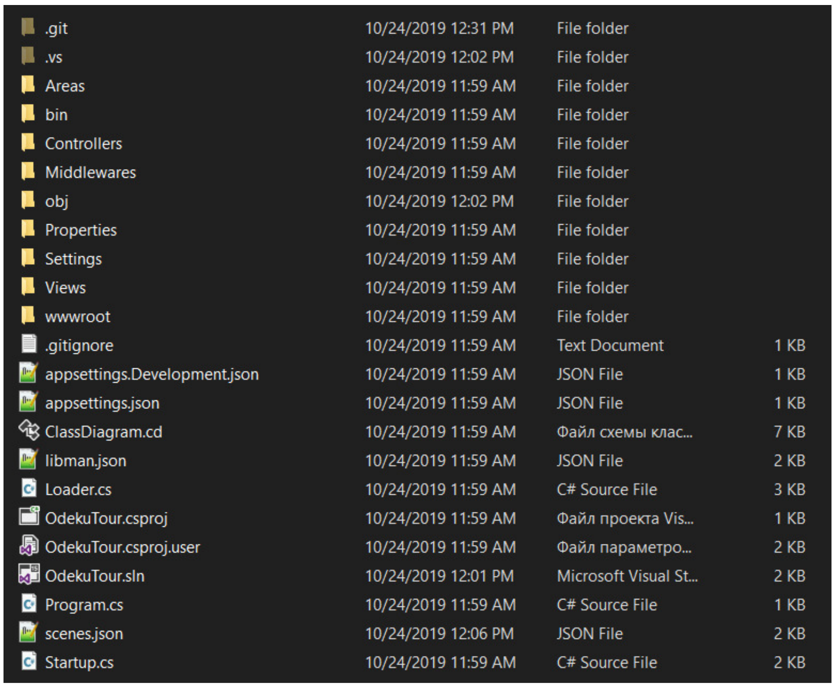


Figure 5. Project structure

2.2 Project realization

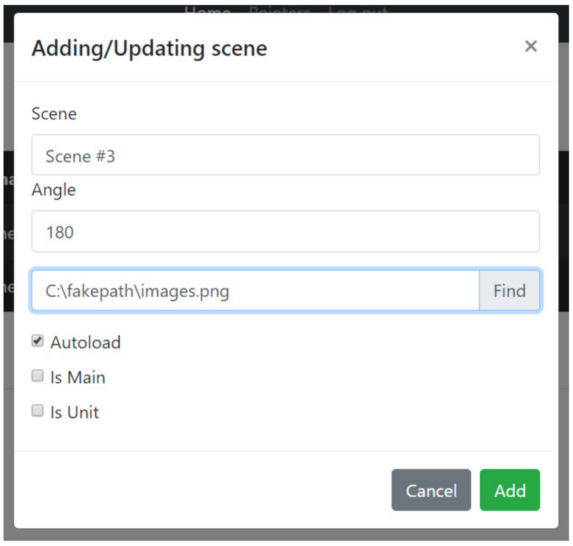
Eventually, was implemented project which provides functionality of creating new panorama tours and loads of features for setting the system up.
This project should be reviewed as a particular module, even though it can be as independent web application. Should be noticed, that application provides basic interfaces to connect to other system or parts of this system.
In order to log in we need to enter domain_name/admin/login in the address bar. Then enter the username and password, they can be changed in the appsettings.json file (fig. 6):



```
.git                            10/24/2019 12:31 PM   File folder
.vs                             10/24/2019 12:02 PM   File folder
Areas                           10/24/2019 11:59 AM   File folder
bin                             10/24/2019 11:59 AM   File folder
Controllers                     10/24/2019 11:59 AM   File folder
Middlewares                     10/24/2019 11:59 AM   File folder
obj                             10/24/2019 12:02 PM   File folder
Properties                      10/24/2019 11:59 AM   File folder
Settings                        10/24/2019 11:59 AM   File folder
Views                           10/24/2019 11:59 AM   File folder
wwwroot                         10/24/2019 11:59 AM   File folder
.gitignore                      10/24/2019 11:59 AM   Text Document          1 KB
appsettings.Development.json    10/24/2019 11:59 AM   JSON File              1 KB
appsettings.json                10/24/2019 11:59 AM   JSON File              1 KB
ClassDiagram.cd                 10/24/2019 11:59 AM   Файл схемы клас...     7 KB
libman.json                     10/24/2019 11:59 AM   JSON File              2 KB
Loader.cs                       10/24/2019 11:59 AM   C# Source File         3 KB
OdekuTour.csproj                10/24/2019 11:59 AM   Файл проекта Vis...    1 KB
OdekuTour.csproj.user           10/24/2019 11:59 AM   Файл параметро...      2 KB
OdekuTour.sln                   10/24/2019 12:01 PM   Microsoft Visual St... 2 KB
Program.cs                      10/24/2019 11:59 AM   C# Source File         1 KB
scenes.json                     10/24/2019 12:06 PM   JSON File              2 KB
Startup.cs                      10/24/2019 11:59 AM   C# Source File         2 KB
```

Figure 6. Components

On the "Main" table is located with sections (panoramas), using the add button you can create new panoramas and set the settings for each of them. The modal window for adding a scene is as follows (fig. 7):



Adding/Updating scene

Scene: Scene #3

Angle: 180

C:\fakepath\images.png

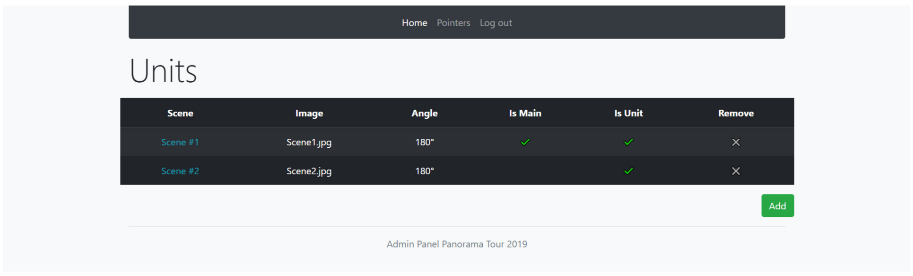
[✓] Autoload

[ ] Is Main

[ ] Is Unit

Figure 7. Adding scenes

The “Scene” field is necessary for recording the name of the panorama itself, which will be displayed on the site for users. Field "Viewing angle" - allows you to configure the viewing angle in degrees for the user. In the third line, the administrator selects a file that will be a panorama. Option "Startup" - determines whether the scene will load automatically or by clicking. "Main scene" - determines whether the panorama will be loaded when the site is first loaded for the user. “Section” - determines whether the panorama will be in the headings of the site. The successful panorama loading screen is as follows (fig. 8).



Units

Scene	Image	Angle	Is Main	Is Unit	Remove
Scene #1	Scene1.jpg	180°	✓	✓	×
Scene #2	Scene2.jpg	180°		✓	×

Admin Panel Panorama Tour 2019

Figure 8. Scene view

UI was created in the way so when the user hover the name of the image that was uploaded, a tooltip will pop up and display the compressed image. If you click on the name, a modal window will be displayed, on which the image will be displayed in full size. The scene is updated according to the same scenario as the addition. The administrator has the right to change all options and then click on the change button. Saves will be applied after page reload.

In order to delete a scene, just left-click on the cross that is in the panorama line. Click the “Delete” button and the panorama will be removed from the list.

The next section is Pointers (fig. 9). This section allows user to configure hot spots that will be used for tour navigation in the future. Namely:
The “Add” and “Delete” buttons switch modes of interaction with pointers. Pointer The point that is set on the scene to indicate additional information or to make transitions to other scenes:

- Select the "Move" mode to scroll through the scene,
- Select the "Add" mode and then click on the scene area where you want to add a pointer,
- Select the "Delete" mode, and then left-click on the pointer that you want to delete.

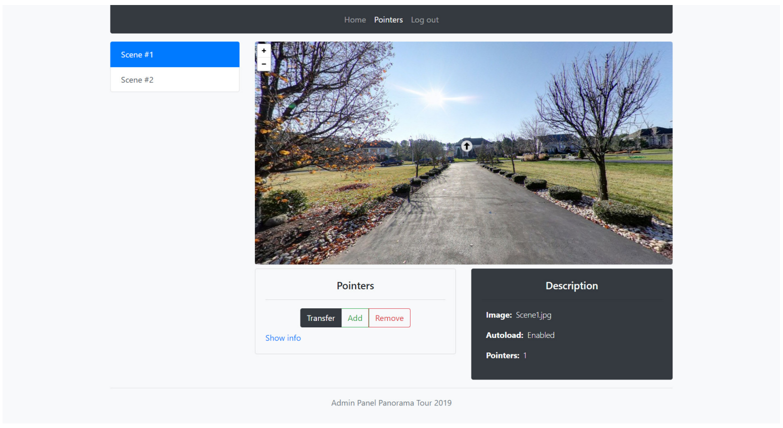
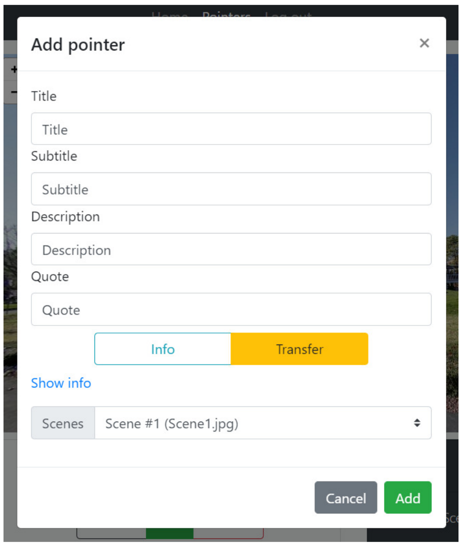


Image: Scene1.jpg
Autoload: Enabled
Pointers: 1

Figure 9. Pointer's page

The pointer is added using an additional window (fig. 10) that allows you to enter some parameters:

- Title,
- Subtitle,
- Description,
- Signature,
- Ability to switch to another site,
- Ability to switch to another scene.



Scenes: Scene #1 (Scene1.jpg)

Figure 10. Transition pointer

There is an option with the addition of a new index, which will move the user to the page www.google.com. Below is a second version of the pointer - "Transition to a new scene."

The main page will look like this (fig. 11):

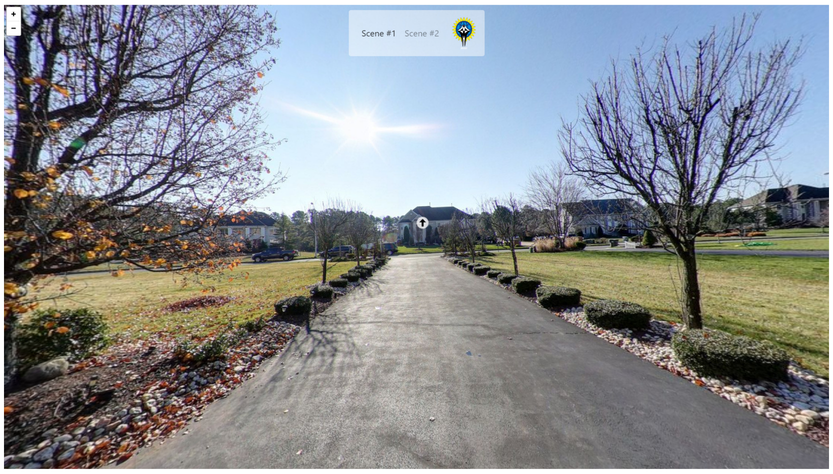


Figure 11. The main page after manipulations

4. Conclusion

All of web application structure represents Model-View-Controller (MVC). MVC is a design pattern used to decouple user-interface (view), data (model), and application logic (controller). This pattern helps to achieve separation of concerns (fig. 12).
Using the MVC pattern for websites, requests are routed to a Controller which is responsible for working with the Model to perform actions and/or retrieve data. The Controller chooses the View to display, and provides it with the Model. The View renders the final page, based on the data in the Model [7].

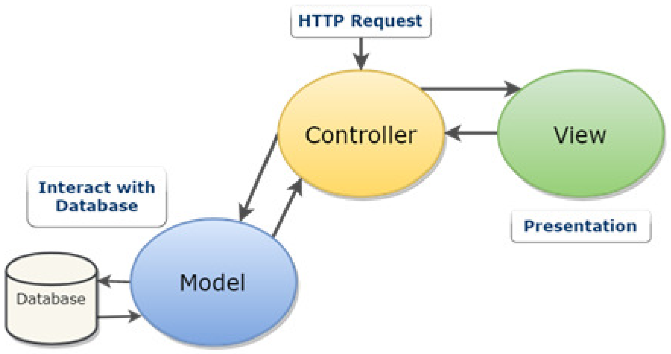


Figure 12. MVC pattern

Often, there are a plenty of solutions when MVC don't use model as database. And this project is not an exception. It had been decided to exclude database from the composition because sending requests and waiting for their responses entails several problems, such as non-optimized application, less understanding structure, problematic to change and add another logic. But when it uses only serialization this solution allows to change data in real time (fig. 13).

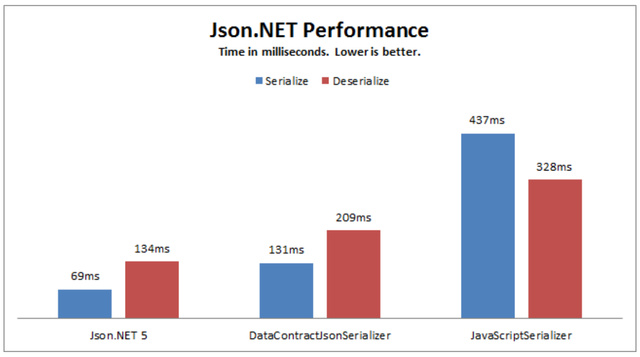


Figure 13. Advantages of JSON serializing

Finally, was developed standing alone module for creating and visualizing panorama tours with optimized data storing and invented rendering algorithm. It allows in real-time to change image data, upload or remove it and to build new panoramas. Also, this application version let users save their time for developing and optimize resources needed for supporting.
Source link: *https://github.com/nikitadermenzhi/panorama*

REFERENCES

1. FREEMAN A.: Pro ASP.NET Core MVC 2 2017.
2. TROELSEN A., JAPIKSE P.: Pro C# 7: With .NET and .NET Core 8th Edition, Kindle Edition.
3. Internet service Microsoft Docs, Introduction to Razor Pages in ASP.NET Core: *https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.0&tabs=visual-studio* 10.06.2019.
4. Internet service ECMA-404 The JSON Data Interchange Standard: *http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf* 12.12.2017.
5. Internet Service Json.NET Documentation. Serializing JSON: *https://www.newtonsoft.com/json/help/html/SerializingJSON.htm* .
6. Internet Service Newtonsoft. Popular high-performance JSON framework: *https://www.newtonsoft.com/json* 11.27.2018.
7. Galloway J., Wilson B., Scott Allen K., Matson D.: Professional ASP.NET MVC 5.