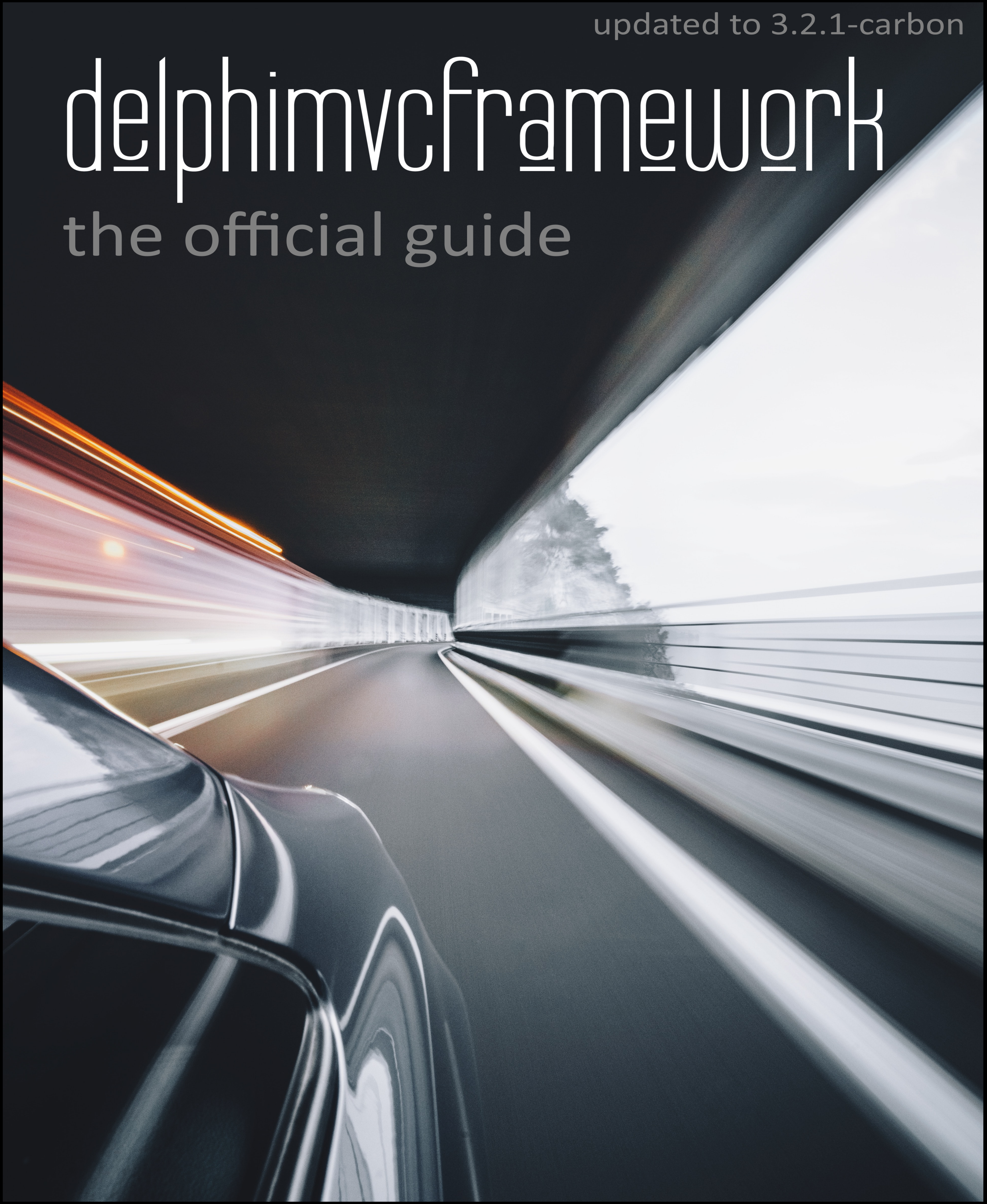
updated to 3.2.1-carbon

delphimvcframework

the official guide



foreword by Jim McKeeth

Daniele Teti

DelphiMVCFramework

Leverage the power of REST and JSON-RPC using the most popular framework for Delphi

DANIELE TETI

This book is for sale at http://leanpub.com/delphimvcframework

This version was published on 2020-11-05

Tweet This Book!

Please help DANIELE TETI by spreading the word about this book on Twitter!

The suggested hashtag for this book is #dmvcframeworkhandbook.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#dmvcframeworkhandbook

To my beloved wife Debora and our little boy Mattia.

Contents

Foreword

Written by **Jim McKeeth**

Chief Developer Advocate & Engineer

Embarcadero Technologies

Everyone knows Delphi is the best solution for connecting to databases and building mobile and desktop applications. The Delphi MVC Framework (DMVCFramework) brings the amazing productivity of Delphi to the web!

The DMVCFramework is based on Delphi's WebBroker module, which was introduced in Delphi 3 back in 1997. A lot has changed since WebBroker first appeared, both with Delphi and with the web.

The First Web Browser War began in 1997. Microsoft just released Internet Explorer version 4 and left a ten-foot-tall letter "e" logo on Netscape's front lawn, with a "From the IE team …We Love You," sign attached. After the Netscape employees knocked it over they returned the favor with a Mozilla dinosaur mascot holding a sign reading "Netscape 72, Microsoft 18," the browser market share at the time. Opera, Apple's Cyberdog, the Lynx text-only browser, and others made up the remaining 10%.

As the web evolved we see the rise of XML and SOAP web services. Both XML (Extensible Markup Language) and HTML (HyperText Markup Language) are based on the Standard General Markup Language (SGML), which is the source of the angle brackets. SOAP is the Simple Object Access Protocol that allowed for Remote Procedure Calls (RPC) and promised a decentralized collection of remote services easily invoked via web requests.

Delphi got it's XML based SOAP Web Services in 2001's Delphi 6. It added both the ability to build XML SOAP Servers and consume existing XML SOAP services, even those built with other tools. I contributed a chapter to a book on building a web-based SOAP Client that consumed a .NET SOAP Services. It used Kylix to deploy the Delphi web server on Linux.

SOAP never realized its dreams of revolutionizing the way services were called, only ever receiving limited adoption. Things changed again when Roy Fielding introduced the term *representational state transfer* (REST) in his doctoral dissertation. Where XML SOAP was a heavyweight officially adopted protocol, the new REST challenger was lightweight and vague. Really more of a descriptive recommendation.

REST is built on the simplicity of the HTTP protocol. This allows for a regular web browser to view a simple REST endpoint. The resulting data is typically in JSON (JavaScript Object Notation made up of curly braces), but the data is returned.

Further, a service is considered RESTful if it meets the following architectural constraints:

1. **Uniform interface** - REST makes use of the Uniform Resource Locators (URL) standard of the web to define endpoints as the interface of REST
2. **Client–server** - This is all about separation of concerns, and the improved scalability that comes with it. The data storage and the user interface are decoupled and allowed to evolve independently.
3. **Stateless** - This requires that each request must contain all of the information necessary and cannot take advantage of any stored state. So two identical requests will produce the same identical response. This allows for greater horizontal server scalability. Many rest implementations do allow of an authentication token to maintain a session, but beyond that, it is important that the order of requests does not impact the results.
4. **Cacheable** - Since it is stateless, it is easily cacheable. If you request a specific page of data, then that same request will always return the same results (until the data is changed). A non-cacheable system may automatically give you the next page on future requests, but it wouldn't be stateless either.
5. **Layered system** - This is where the popular term "microservices" comes from, with the idea that a RESTful service architecture is made of multiple layers of REST services, each only aware of its neighboring layers.

Finally, with REST we see the realization of a distributed and open web services architecture. Most every site that offers a service makes it available via a RESTful API.

What is the Delphi MVC Framework, and why does it matter?

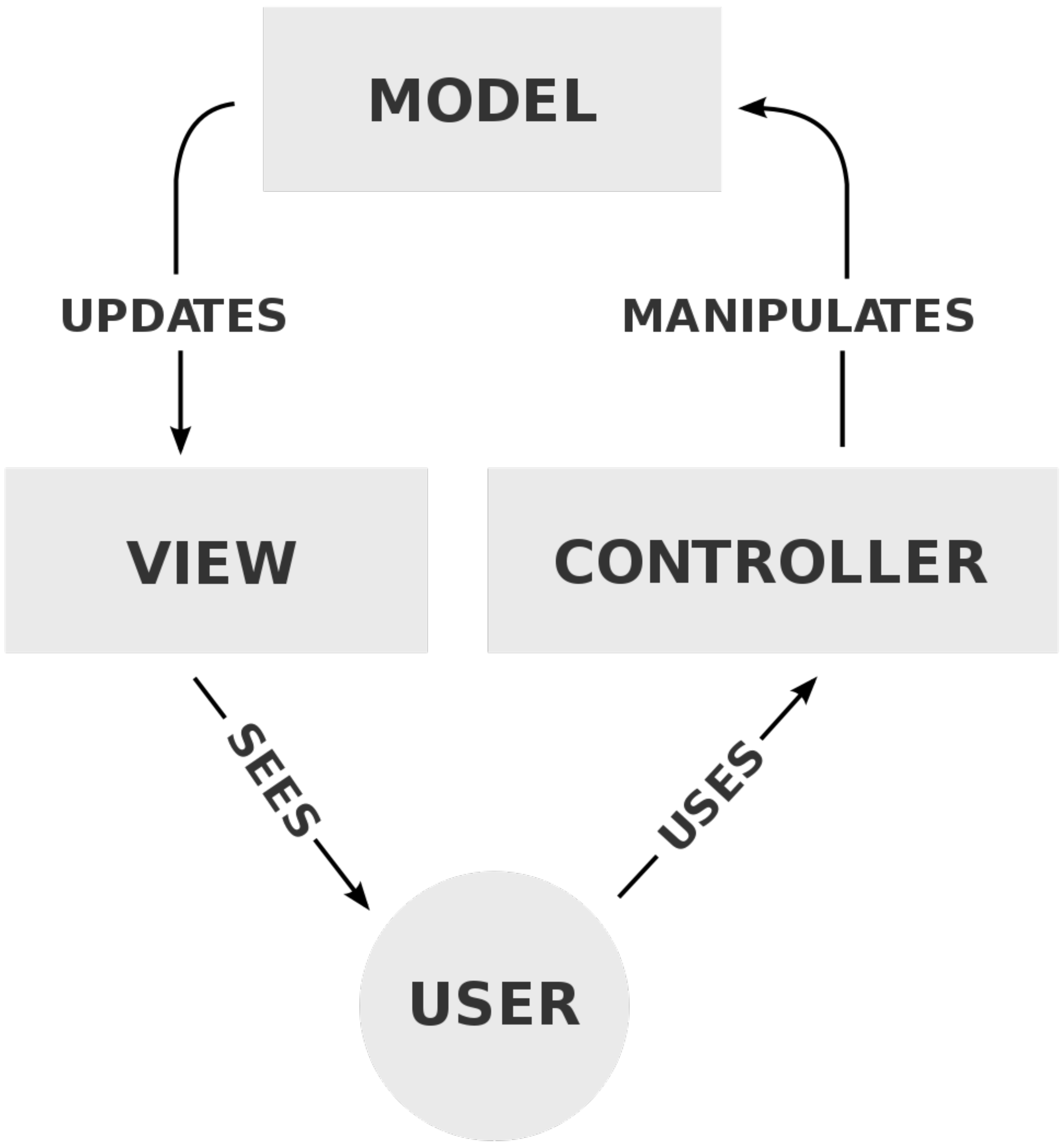


Figure 1

MVC is a design pattern all about decoupling the Model, View, Controller to make it easier to build an application where the user interface (view) is independent of the data (model) and functionality (controller). This makes it easier to create a multi-tier application with multiple user interfaces: web, mobile, desktop, etc.; that all use the same backend. The user observes the view, interacts with the controller, which manipulates the model, updating the view. (See Figure 1)

With the DMVCFramework this is expressed as a REST-based web service. The controller is made up of the REST endpoints that provide the functionality to manipulate the model or data (usually a database, but not always), then the view is the REST endpoints that provide the data to the user. The advantage of this decoupled pattern is the view is just as easily implemented through a web page as through a mobile or desktop application. They all just call the REST endpoints.

DMVCFramework is one of the most popular frameworks for building web applications with Delphi. It includes full support of RESTful (RMM Level 3) JSON-RPC 2.0 APIs making the building and publishing as simple and flexible as possible. It isn't the only REST Server solution for Delphi, and that is one of the things I love about Delphi: the variety and choice available to the end-user.

If someone were to ask me which REST Framework to choose, I would always tell them it depends on their implementation and requirements, but if you are new to building a REST Server solution, then the DMVCFramework is a great place to start. As an open-source project, the barrier to entry couldn't be lower, especially with the availability of this book. You may find it is the perfect solution for you.

-Jim McKeeth

Chief Developer Advocate & Engineer

Embarcadero Technologies

jim.mckeeth@embarcadero.com

Reviewers

- Marco Cotroneo *Senior Full Stack Developer at bit Time Professionals*
- Alessio Festuccia *Full Stack Developer at bit Time Professionals*
- Nirav Kaku *CEO at VEditIndia and Embarcadero MVP*

While a huge work has been done by the author and the reviews to make the book and the examples well written, complete and effective, things can be always improved. For any suggestions, complains or requests there is the official Github book project (https://github.com/danieleteti/dmvcframeworktheofficialguide) where you can fill an issue and get in touch directly with the author.

Collaborators

Chapter 13: Document and test your REST API with the Swagger middleware (added in 1st edition, update 4) has been almost completely written by João Antônio Duarte which is the main author of the Swagger middleware. Thank you for your support, João.

Translations

DMVCFramework - the official guide is also available in Brazilian Portuguese (Thanks to Diego Farisato for its translation and for its knowledge of the Brazilian Delphi scenarios)

DMVCFramework - the official guide is also available in Spanish (Thanks to Josè Davalos for its translation and for its knowledge of the Spanish Delphi scenarios)

What users say about DMVCFramework

"DMVCFramework and the Entity utility are fantastic!" – Roberto

"DMVCFramework is a great framework. It's very intuitive, fast, easy to use, actually there is nothing more to ask for." – Samir

"Wow! To do that in J2EE it takes 2 days" – a training participant after a 5 minutes demo.

"I'm starting with the DMVCFramework and I'm finding it fantastic, congratulations for the project!" – Rafael

"I'm looking at DMVCFramework project in it works great - for my use case scenarios is much better than 'Similar commercial product'." – Luka

"It's fantastic! Just define your entities and you are up and running in 5 minutes. Nothing comparable on the market." – Marco

"The best framework for creating web servers with Delphi! It is very easy to create Delphi servers and publish APIs and Rest resources. Congratulations to Daniele Teti and all the staff for the excellent work!" – Marcos N.

"We started the process of migrating our systems to micro services and are loving the DMVCFramework. DMVCFramework is definitely part of our lives right now". – E. Costa

"Thank you for the great framework! We are very happy with this!" – Andreas

Fetching This Book’s Code

Source code for the book’s examples can be fetched as extra content from the book page in the editor’s web site Of course, the examples work best in the context of their appearance in this book, and you’ll need some DMVCFramework background knowledge to make use of them.

Using This Book’s Code

Book Release Notes

2020-08-31 - 1st edition, update 1

- The book is at 95%
- Covering DMVCFramework-3.2.1-carbon

2020-09-15 - 1st edition, update 2

- The book is completed
- Added the foreword by Jim McKeeth
- Fixed some errors in the English text
- Added Nirav Kaku as reviewer
- Covering DMVCFramework-3.2.1-carbon

2020-10-17 - 1st edition, update 3

- Fixed some typos
- Added *Tip #7* about RQL in chapter 13
  - If you are looking info about RQL, note that Chapter 13 will become 14 in 1st edition, update 4.
- Updated section *“Change the properties case”* in chapter 3 to consider `ncSnakeCase` too.

2020-11-05 - 1st edition, update 4

- Added paragraph for the ETag middleware in *Chapter 10 Middlewares*.
- Added Chapter 13 about the Swagger/OpenAPI support.
- *Chapter 13: Tips and Tricks* in previous editions, now is chapter 14.
- Added some missing images captions

Chapter 1: Getting Started with DelphiMVCFramework

What you’ll learn

DelphiMVCFramework, or DMVCFramework (or even just DMVC as many users like to say) is a Delphi framework which allows to create powerful RESTful and JSON-RPC servers without effort. At the time of writing DMVCFramework is the most popular open source Delphi project on Github with the biggest community. Many users reported that they switched from a commercial product to DMVCFramework and got a simple development cycle, higher performance and more flexibility.

You can create a full-flagged RESTful server in a couple of clicks, that’s it. As you guess to create a full-flagged web solution you have to work on it to implement your own logic and rules, but having a framework easy to understand and fast to customize and “hack” is a really big advantage. DMVCFramework has been designed to have a double soul (speed and simplicity) and in its 3rd version I think that we have almost reached the goal. I suggest to subscribe to the dedicated Facebook group which, at the time of writing, has more than 3400 active users.

DMVCFramework is “batteries included”

DMVCFramework is composed by a number of units and sub frameworks that work together. This collection of modules makes some set of tasks within a particular problem domain simpler to implement. However some parts of DMVC are not tied with the framework itself and can also operate in an application (or an app) different from a DMVCFramework project.

In this handbook you will learn about the following DMVCFramework sub-projects:

- MVCActiveRecord
- LoggerPro (which is also released as separate project)
- The serialization/deserialization framework
- The Authentication/Authorization subsystem
- The RQL compilers
- The multi threaded cache

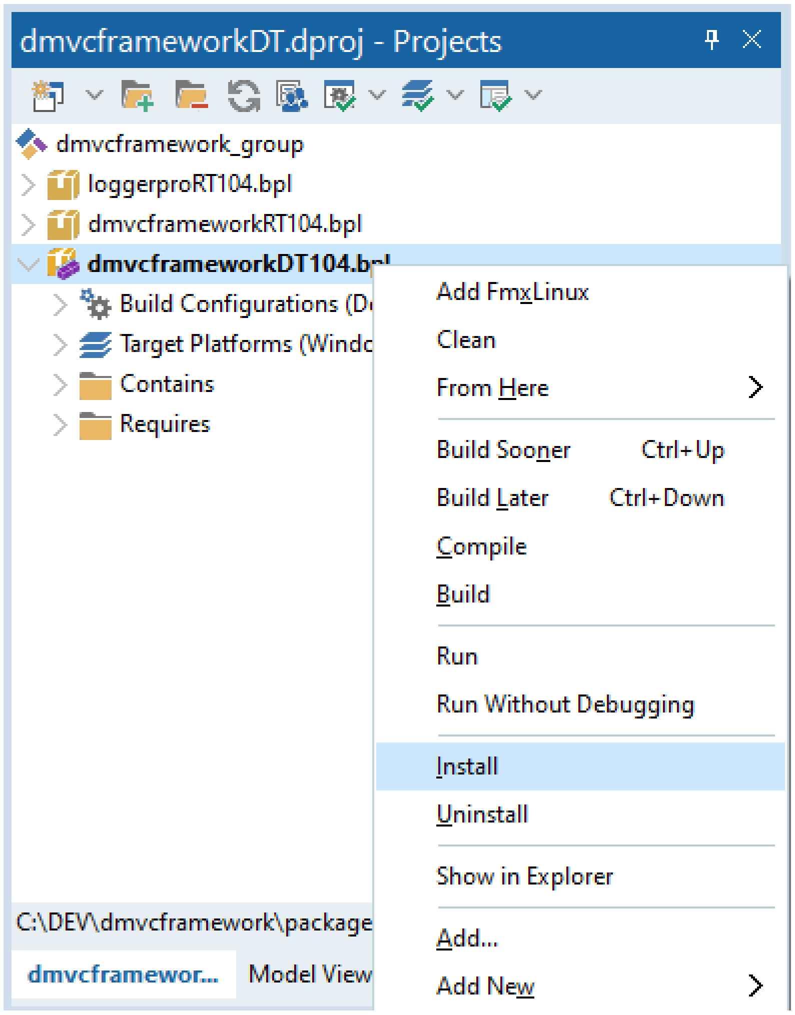
Another good news for the new DMVCFramework users is related to one of the most powerful available feature: the middleware. As you will learn, middleware allows to extend and customize the default DMVCFramework behavior.

Installation of DelphiMVCFramework 3.2.1-carbon

DMVCFramework is released as Github release, which is a plain zip file so that you can use and update each version in a very simple way.

The installation procedure is the following:

- Navigate to https://github.com/danieleteti/delphimvcframework/releases and click on `Latest release`.
- Scroll the *What's New* section till the end, open the *Asset* group.
- Here you will find 4 downloadable files:
  - DMVCFramework-[version].zip (contains the actual DMVCFramework release source code - *this is the file to download*)
  - DMVCFramework-[version]_samples.zip (contains all the samples - download it as reference)
  - Source code (zip) (automatically generated by Github, don't download)
  - Source code (tar.gz) (automatically generated by Github, don't download)
- Unzip the file `DMVCFramework-[version].zip` in a folder named `C:\DelphiLibs\DMVCFramework` (or wherever you prefer). From now on we will indicate this folder as `$(dmvchome)`;
- Launch RAD Studio and open `$(dmvchome)\packages\d104\DMVCFramework_group.groupproj` or the project that suits your IDE version;
  - `d104` stands for "Delphi 10.4 Sydney"
  - `d103` stands for "Delphi 10.3 Rio"
  - `d102` stands for "Delphi 10.2 Tokyo"
  - `d101` stands for "Delphi 10.1 Berlin"
  - `d100` stands for "Delphi 10.0 Seattle"
- Build all the projects;
- Install `DMVCFrameworkDT104` selecting the project in the Project Manager and using `Right Click->Install` menu as shown in the next image



Installing DMVCFramework

- That's it, close all;

DMVCFramework even works on Delphi Professional and the free Delphi Community (which is aligned to the Professional one).

> WARNING! In case you have Delphi Professional or Community you cannot install the IDE expert because of a lack of a required package, but you can still use the framework. It only happens for some Delphi versions but in such cases you can learn how to create a DelphiMVCFramework by hand or just copying a sample project and starting to change it.

Now, DMVCFramework is installed and the next step before creating your first project is to configure the library paths.

Go to `Tools->Options->Language->Delphi->Library->Library Path` and add the following paths:

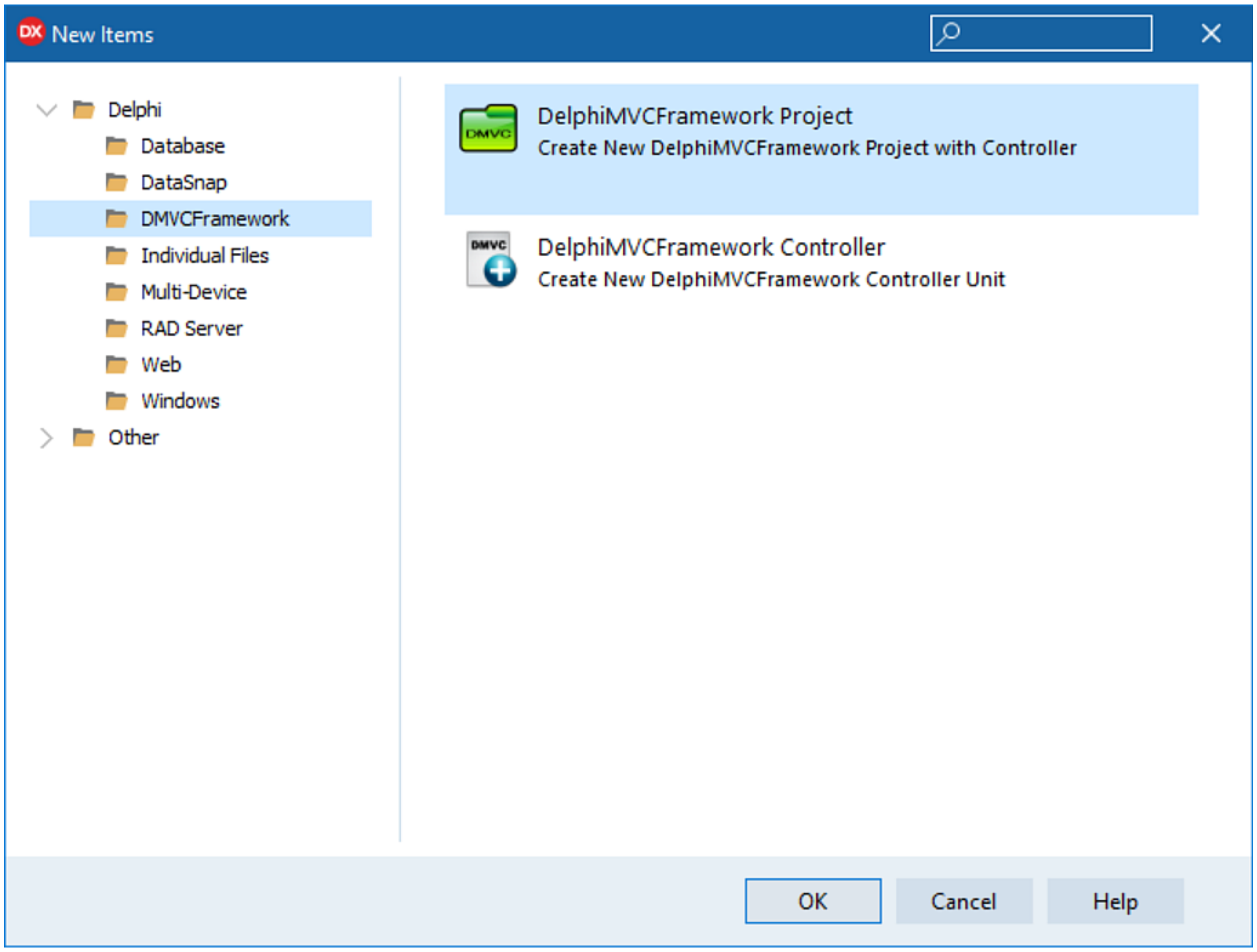
```
$(dmvchome)\sources
$(dmvchome)\lib\dmustache
$(dmvchome)\lib\loggerpro
$(dmvchome)\lib\swagdoc\Source
```

Click `OK` and close the dialog. Now, we are ready to create our first amazing RESTful service!

Your first RESTful server with DelphiMVCFramework

Creating the first RESTful service with DelphiMVCFramework is straightforward.

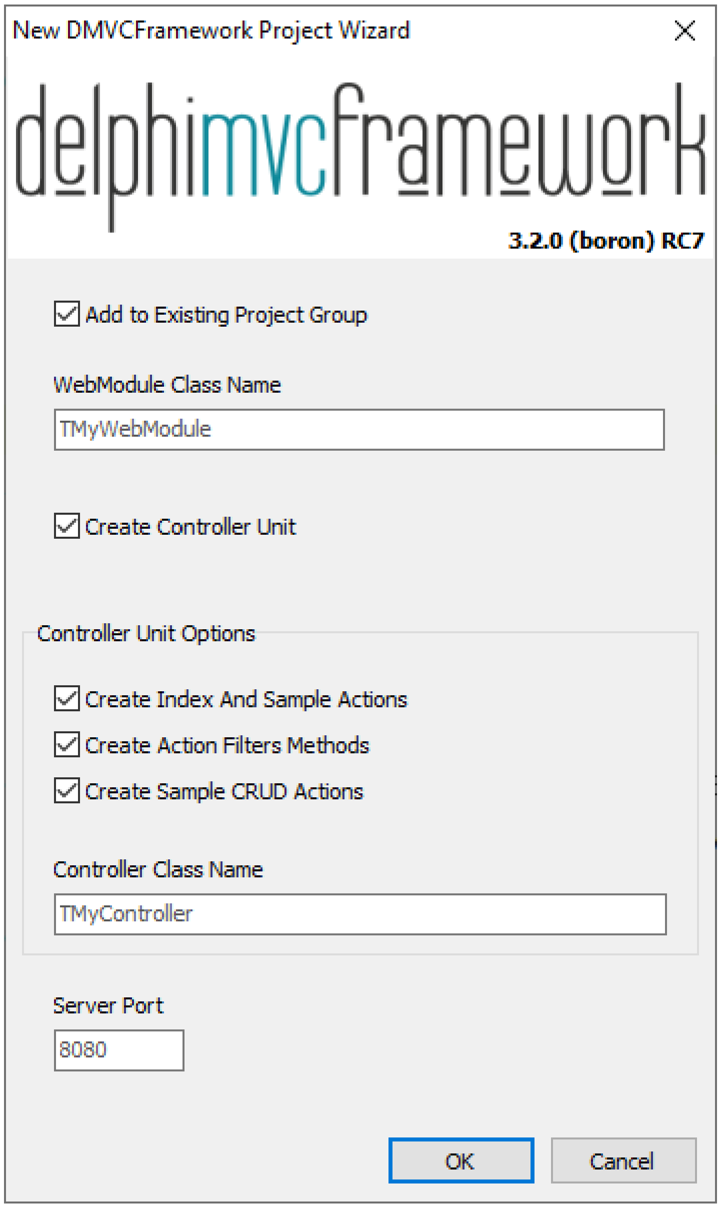
- Go to menu `File->New->Other`;
- From the resultant dialog select `Delphi->DMVCFramework->Delphi MVC Framework Project` as shown below;



DelphiMVCFramework Project
Create New DelphiMVCFramework Project with Controller

DelphiMVCFramework Controller
Create New DelphiMVCFramework Controller Unit

- Click `OK` and you will get the following (your version might be different from the one reported in the image)



New DMVCFramework Project Wizard

3.2.0 (boron) RC7

[✓] Add to Existing Project Group

WebModule Class Name: TMyWebModule

[✓] Create Controller Unit

Controller Unit Options

[✓] Create Index And Sample Actions

[✓] Create Action Filters Methods

[✓] Create Sample CRUD Actions

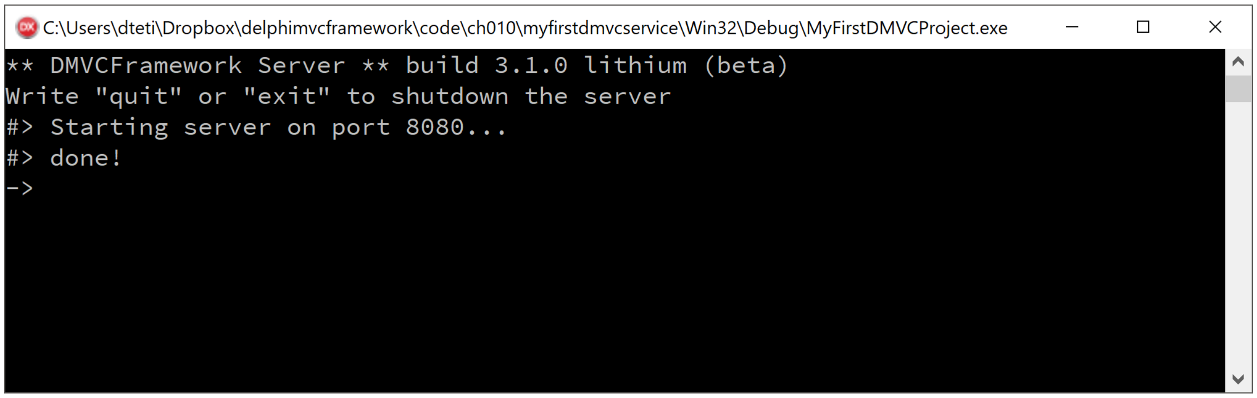
Controller Class Name: TMyController

Server Port: 8080

The DMVCFramework Wizard

- Leave all the default settings and click `OK`.
- As you can see, an "empty" DMVCFramework project is composed by:
  - a web module unit (which contains the `TMVCEngine` instance)
  - a controller unit (which contains the first controller of our application)
- Save the web module unit as `WebModuleU.pas`, the controller as `MyControllerU.pas` and the project as `MyFirstDMVCService.dproj`.

- Compiling the project you will get some errors because the wizard used the default unit name for the controller, that we changed in the previous step. In the implementation section replace the wrong name (e.g. `Unit2`) with `MyControllerU` (or what you choose before) which defines the controller. Save it and recompile.
- Run the project using the `Run` button or hitting `F9` and you will get the following:


C:\Users\dteti\Dropbox\delphimvcframework\code\ch010\myfirstdmvcservice\Win32\Debug\MyFirstDMVCProject.exe

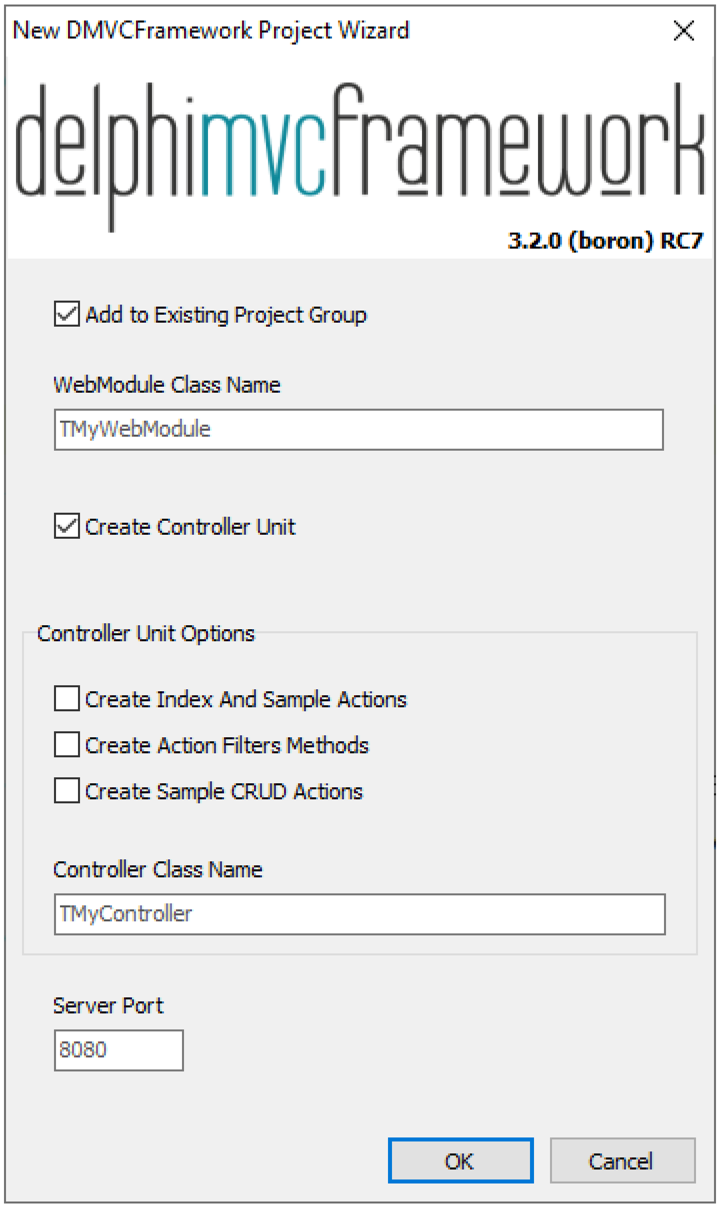
```
** DMVCFramework Server ** build 3.1.0 lithium (beta)
Write "quit" or "exit" to shutdown the server
#> Starting server on port 8080...
#> done!
->
```

- Yes! The DMVCFramework default console application is running serving you first DMVCFramework server. Simple, isn't it?

Now, let's give honor to the "Hello World Tradition" with…

Your first DMVCFramework-style "Hello World"

- Launch the DMVCFramework new project wizard dialog, remove all the checks in the group `Controller Unit Options` as shown below.


New DMVCFramework Project Wizard

3.2.0 (boron) RC7

☑ Add to Existing Project Group

WebModule Class Name: TMyWebModule

☑ Create Controller Unit

Controller Unit Options

☐ Create Index And Sample Actions

☐ Create Action Filters Methods

☐ Create Sample CRUD Actions

Controller Class Name: TMyController

Server Port: 8080

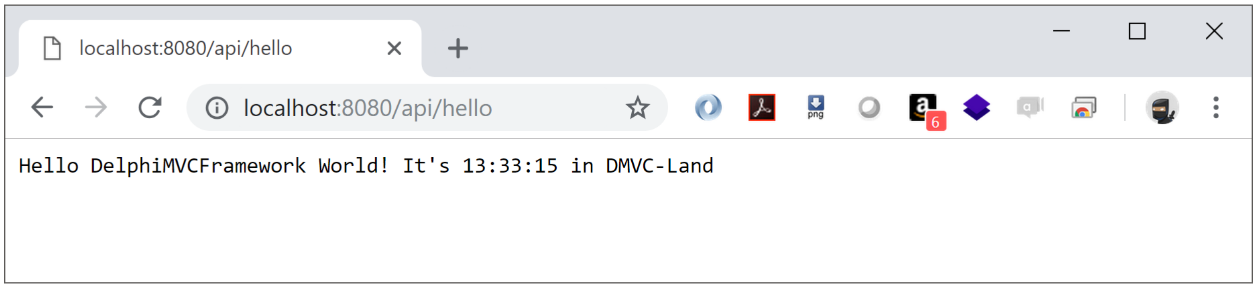
- Hit `OK` and save the project as `MyDMVCHelloWorld.dproj`, the WebModule as `WebModuleU.pas` and the controller as `MyControllerU.pas`.
- Open the controller unit
- As you can see, now the unit doesn't contain all the wizard generated code of the first project we did. It is a very good opportunity to add just only what we really need.
- In the controller class declaration add a procedure method named `Index` (only procedure methods can be called by DMVCFramework router, so you don't have to use functions as method) hit `Ctrl+Shift+C` to autocomplete your class with the `Index` method implementation.
- In the `Index` method write the following code

```
procedure TMyController.Index;
begin
  Render('Hello DMVCFramework World! It''s ' + TimeToStr(Now) +
          ' in DMVCFramework-Land');
end;
```

OK, we defined the method and its implementation. However the engine doesn't still know how to call it because `Index` is just a method, and not a proper DMVCFramework action. To transform a plain method to an action we have to instrument it with the `MVCPath` attribute (we'll see `MVCPath` and all the other attributes in the next chapter). So, go in the class declaration and change the method declaration as shown below.

```
type
  [MVCPath('/api')]
  TMyController = class(TMVCController)
  public
    [MVCPath('/hello')]
    procedure Index;
  end;
```

- Run the program and open a web browser (We'll use Google Chrome in ours example)
- In the browser address bar write `http://localhost:8080/api/hello` and hit return. You should get something like the following.



```
Hello DelphiMVCFramework World! It's 13:33:15 in DMVC-Land
```

Congratulations! You have just written your DMVCFramework Hello World!

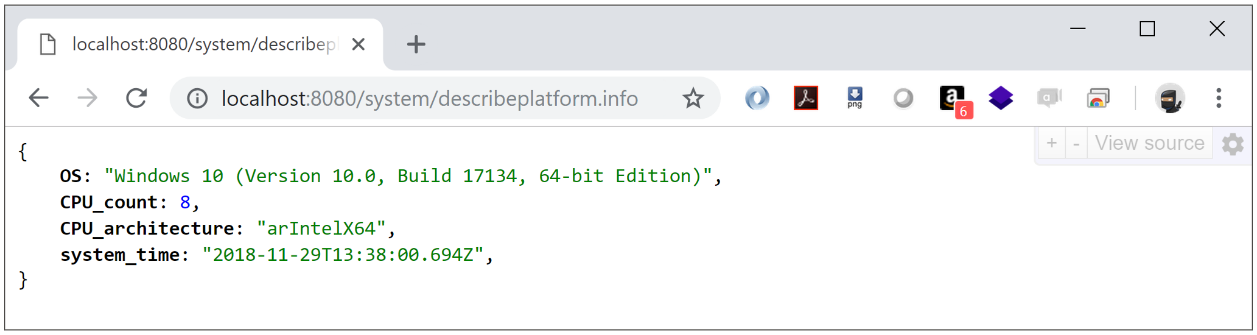
Built-in System Actions

Our small *Hello World* already contains some interesting features called **System Actions**. That's it, every DMVCFramework project always contains an automatic registered `TMVCSystemController` . This controller provides some useful information. All the system actions are invokable only by localhost and you can also decide to completely remove the system controller (check *Chapter 16: Tips and Tricks*). All the System Actions provide only JSON responses - no other formats are supported at this time.

System Actions: describeplatform

Launch the hello world server and open a browser.

If you write `http://localhost:8080/system/describeplatform.info` you get the following


localhost:8080/system/describep
localhost:8080/system/describeplatform.info
View source

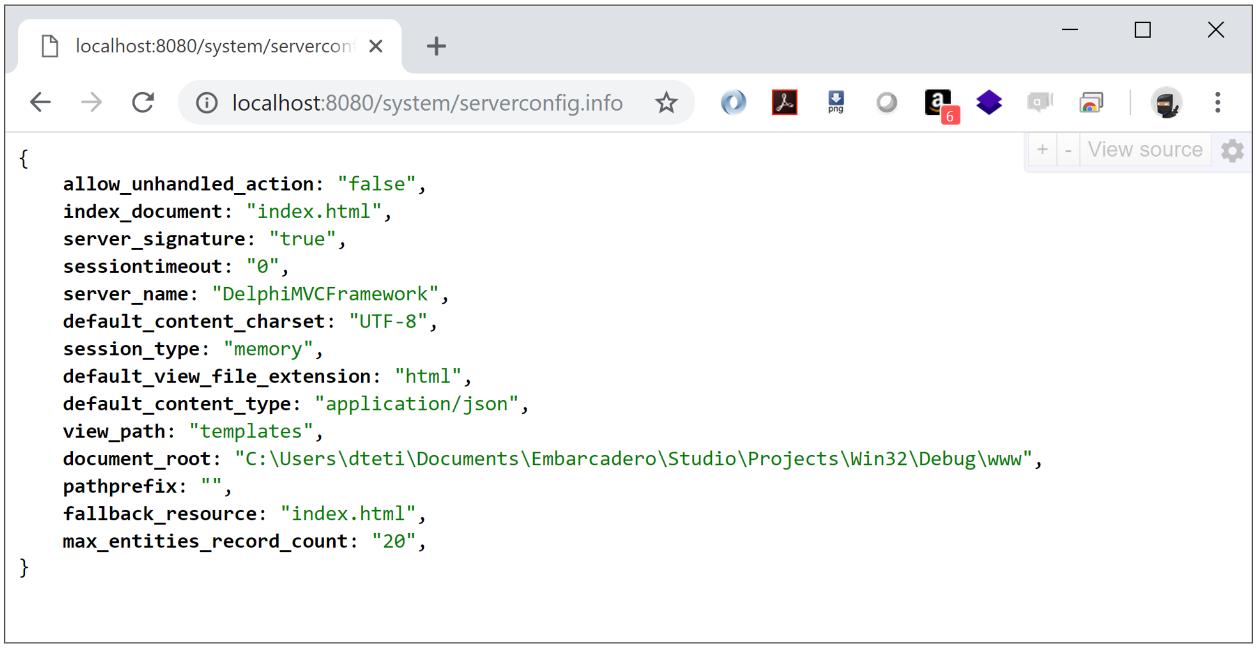
```
{
    OS: "Windows 10 (Version 10.0, Build 17134, 64-bit Edition)",
    CPU_count: 8,
    CPU_architecture: "arIntelX64",
    system_time: "2018-11-29T13:38:00.694Z",
}
```

This action provides information about the server machine. It can be useful to get some information of just to check if the service is up and running.

System Actions: serverconfig

Launch the hello world server and open a browser.

If you write `http://localhost:8080/system/serverconfig.info` you get the following


localhost:8080/system/servercon
localhost:8080/system/serverconfig.info
View source

```
{
    allow_unhandled_action: "false",
    index_document: "index.html",
    server_signature: "true",
    sessiontimeout: "0",
    server_name: "DelphiMVCFramework",
    default_content_charset: "UTF-8",
    session_type: "memory",
    default_view_file_extension: "html",
    default_content_type: "application/json",
    view_path: "templates",
    document_root: "C:\Users\dteti\Documents\Embarcadero\Studio\Projects\Win32\Debug\www",
    pathprefix: "",
    fallback_resource: "index.html",
    max_entities_record_count: "20",
}
```

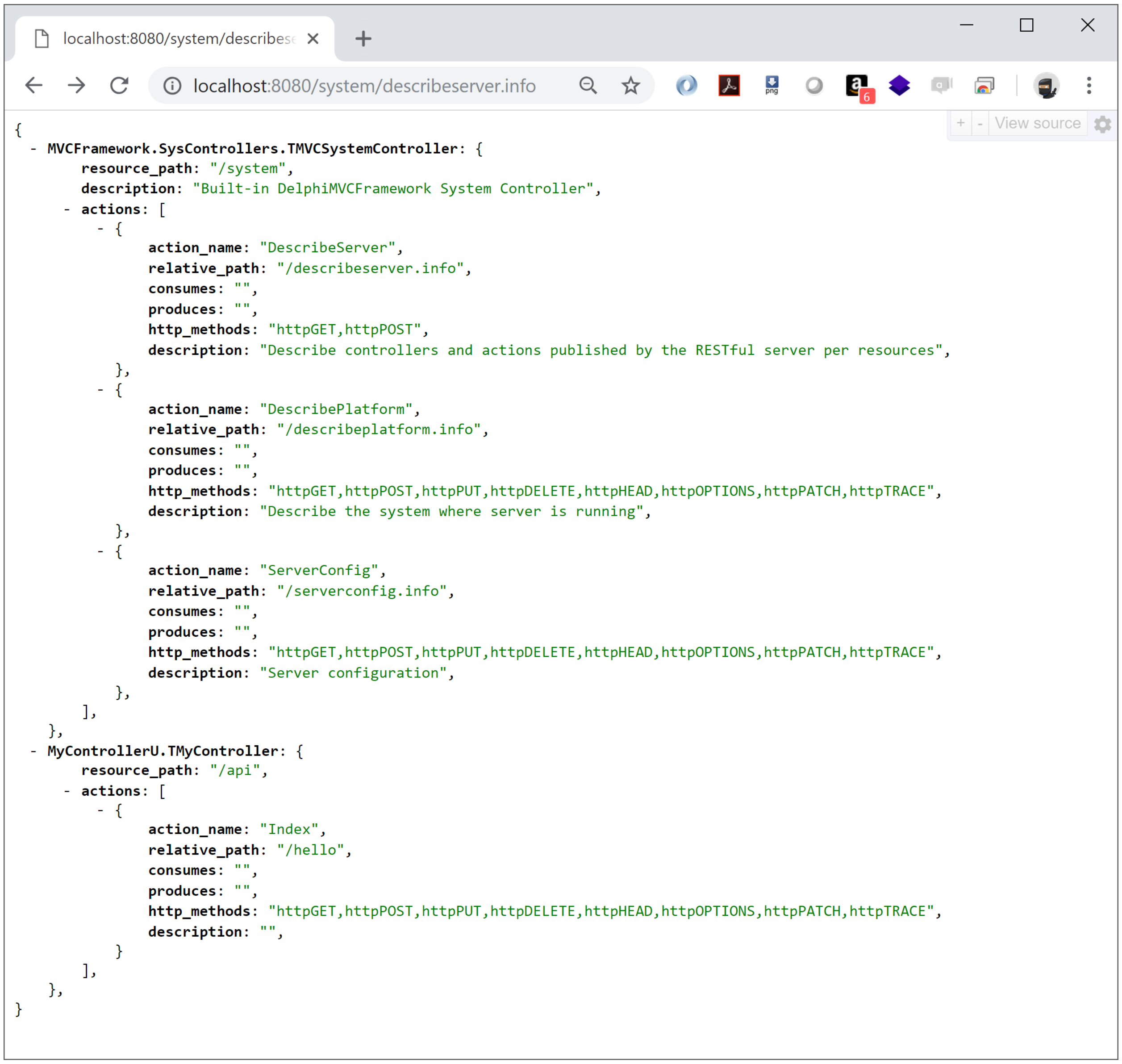
This action provides information about the DMVCFramework engine and its configuration. Can be useful to get all the current configuration of the `TMVCEngine` instance which run inside the server.

Take care of these information because an attacker could get important insights from them. Check the tips in the "How To" chapter to know how to disable the System Controllers registration in your production servers.

System Actions: describeserver

This is the most important System Action. Launch the "hello world" server and open a browser.

If you write `http://localhost:8080/system/describeserver.info` you get the following


localhost:8080/system/describese
localhost:8080/system/describeserver.info
+ - View source

```
{
  - MVCFramework.SysControllers.TMVCSystemController: {
        resource_path: "/system",
        description: "Built-in DelphiMVCFramework System Controller",
      - actions: [
          - {
                action_name: "DescribeServer",
                relative_path: "/describeserver.info",
                consumes: "",
                produces: "",
                http_methods: "httpGET,httpPOST",
                description: "Describe controllers and actions published by the RESTful server per resources",
            },
          - {
                action_name: "DescribePlatform",
                relative_path: "/describeplatform.info",
                consumes: "",
                produces: "",
                http_methods: "httpGET,httpPOST,httpPUT,httpDELETE,httpHEAD,httpOPTIONS,httpPATCH,httpTRACE",
                description: "Describe the system where server is running",
            },
          - {
                action_name: "ServerConfig",
                relative_path: "/serverconfig.info",
                consumes: "",
                produces: "",
                http_methods: "httpGET,httpPOST,httpPUT,httpDELETE,httpHEAD,httpOPTIONS,httpPATCH,httpTRACE",
                description: "Server configuration",
            },
        ],
    },
  - MyControllerU.TMyController: {
        resource_path: "/api",
      - actions: [
          - {
                action_name: "Index",
                relative_path: "/hello",
                consumes: "",
                produces: "",
                http_methods: "httpGET,httpPOST,httpPUT,httpDELETE,httpHEAD,httpOPTIONS,httpPATCH,httpTRACE",
                description: "",
            }
        ],
    },
}
```

This action provides all the information about the controllers and the actions provided by each controllers. As you can see, there is our *Hello World* controller called `TMyController` declared in

`MyControllerU.pas`. Moreover, the `/system/describeserver.info` action shows all the information needed to call the APIs, so it is a very valuable tool to understand what your server is able to do. If you need to provide documentation about your APIs, the document can be generated by reading the output of this action, which is standard JSON.

> DMVCFramework support the OpenAPI Specification too (former Swagger) through a middleware. The specification creates a RESTful interface for easily developing and consuming an API by effectively mapping all the resources and operations associated with it.

What's Next

In this chapter we saw how to create a DMVCFramework project and how to generate some kinds of output. Moreover, we saw what the System Actions are and when they can be useful. Even the simplest DMVCFramework application contains a lot of functionalities ready to be used. We'll explore each of them in the next chapters. Now you can follow the rest of the handbook to learn all the nice things about DMVCFramework.