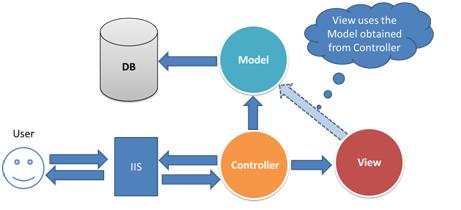
# Technologies and Techniques Used

*In the process of developing this web application a wide variety of technologies and techniques have been used in order to implement the functionality and the user experience of the application, as well as to test its functional performance requirements. The sections below enlist them in more detail.*

## ASP.NET MVC core

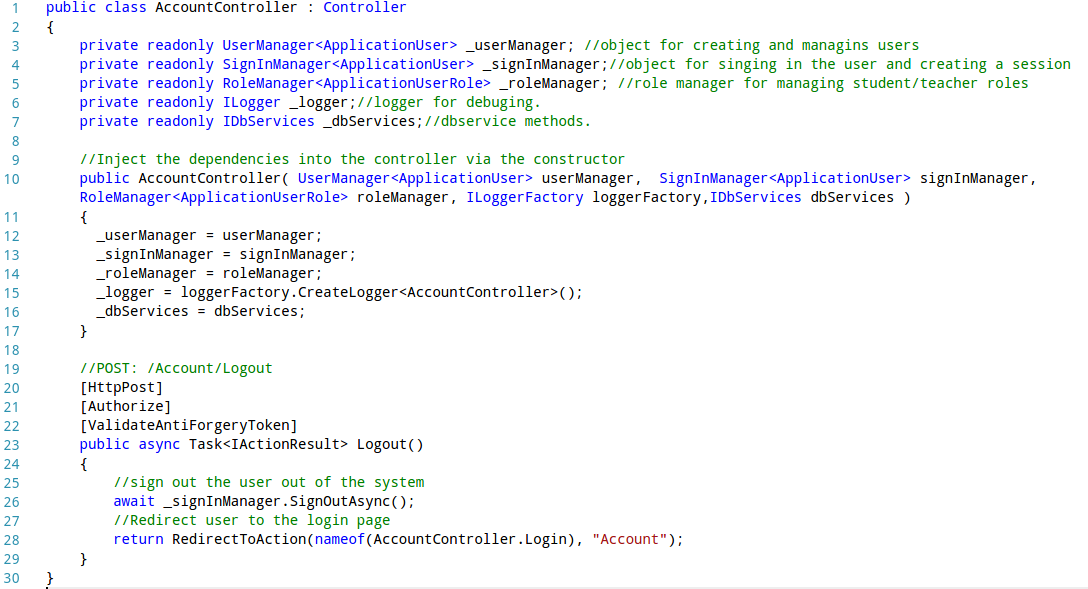
ASP.NET MVC core 1.0, used in this project, is a rather freshly franchised version 5 of the ASP.NET MVC web application framework. It is based on the .NET core 1.0 framework, a newly redesigned .NET platform, developed by Microsoft . The biggest differences from the previous released include moving away from the proprietary development and making the platform open source and cross platform. The main philosophy which was the fundament behind the redesign of the platform was making it modular in its design and architecture, which separates the compiler, runtime and the libraries to be independent components, which communicate through the interfaces with each other. The lose coupling among the components is known as a design pattern called inversion of control. This design pattern can be implemented in several ways, where dependency injection(DI) is one of them, which implies that construction dependencies of the objects are passed to during the objects initiation. The ASP.NET MVC core was built with the DI support in mind . This technique is going to be discussed in more detail under the Advanced Techniques section.

The big part of the ASP.NET MVC philosophy is the Model-View-Controller software design pattern. This pattern aspires to follow the separation of concerns methodology, where the presentational, business logic and data model layers are all separated from each other. In this pattern user requests are pipelined via the server (typically ISS) to a Controller, which is responsible to process the request, access the data model if necessary and response with a corresponding view of the application back to the user, as shown in the figure below.

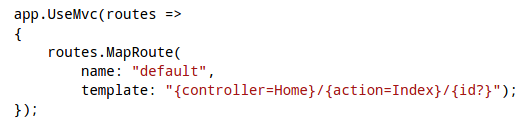


**Figure 1** – A superficial representation of a request processing in an ASP.NET MVC application.

ASP.NET MVC core provides a set of tools which eases the creation of the corresponding components in order to follow the MVC pattern. A controller representation is defined as a class, which includes a set of actions which handle incoming requests. A controller inh*e*rits from the *Controller* base class, which contains predefined methods and properties necessary to process an incoming HTTP request. A code sample bellow shows a definition of an *Account Controller* and a *Logout Action Handler*, contained under /*src/controllers/AccounConroller.cs*. The *AccountController() constructor method* defines the dependencies as the Authorization and Authentication managers, Data Model services to be injected into the controller by the

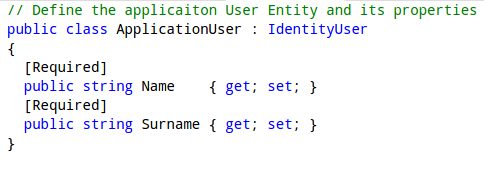
**Sample 1 –** Definition an Account controller.

frameworks services. The annotations defined in the square brackets before the *Logout() action task*,indicate the extra filters that request has to go through before reaching the request handler. *[HttpPost]* action filter defines that only and POST request can access the action handler task. *[Authorize]* authorization filter allows only authorised users to access the handler. *[ValidateAntiForgeryToken]* attribute provides a mechanism to validate the incoming anti-XSRF tokens, so that cross site request forgery cannot be performed by malicious on the forms defined in the web application. The request handler itself performs a simple task of signing the user out via the SingInManger and redirecting the user back to the login page. The routing configuration itself is defined under the */src/Startup.cs* file, as shown in the **Sample 2**. The routes parameter is set via the MapRoutes() method to use the template pattern of */{controller}/{index}/* for routing, which means that the handler above can be accessed through the /Account/Logout route.



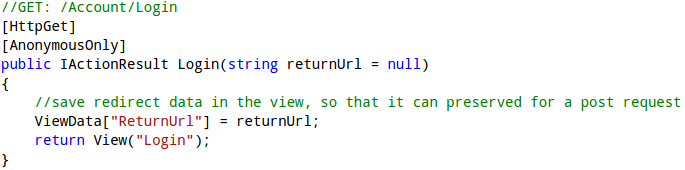
**Sample 2** – Definition of the routes template.

Data model entities within the framework are simply defined a class as shown in **Sample 3**. The properties of the entity are simply created as the properties within that class. These can include Data Annotations defined in square brackets, in this example they implicate that the property is required to be defined once the entity is created. The way these models are mapped to the database is going to be discussed under the EntityFrameworkCore section.



**Sample 3 –** Definition of the Application User entity. Available under /src*/models/ApplicationUsers.cs.*

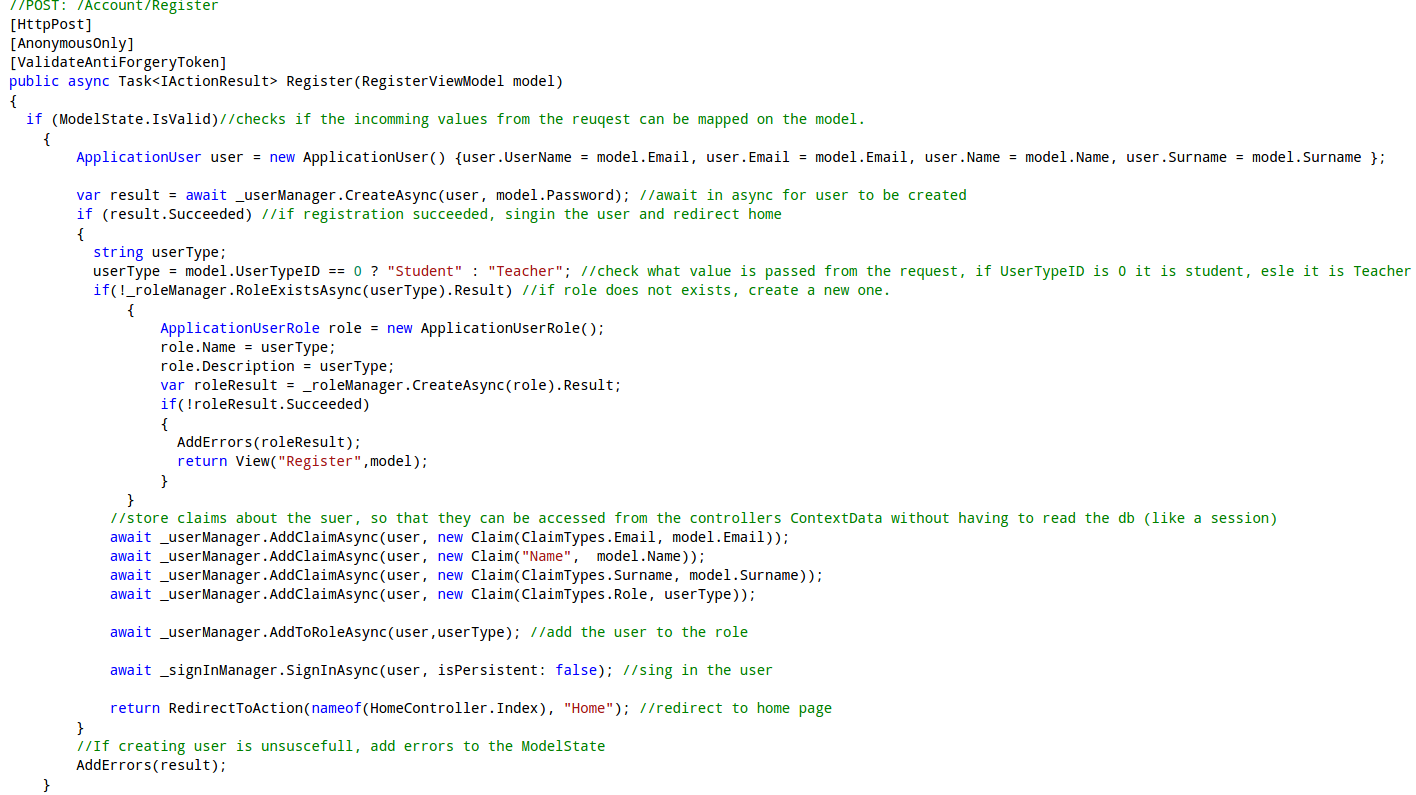
The *ViewResult* object is typically generated when an action handler returns a *View()* method as a result. The *ViewResult* object typically contains a model data that has been queried and *ViewData* that has been generated by the action handler to be passed to the Razor templating engine, which is going to be discussed in more detail in another section, as well as the status code to be sent back as a HTTP response.



**Sample 4–**Login handler which returns a *View()* method which generated the *ViewResult* object. Available under /*src/Controllers/HomeControllers.cs.*

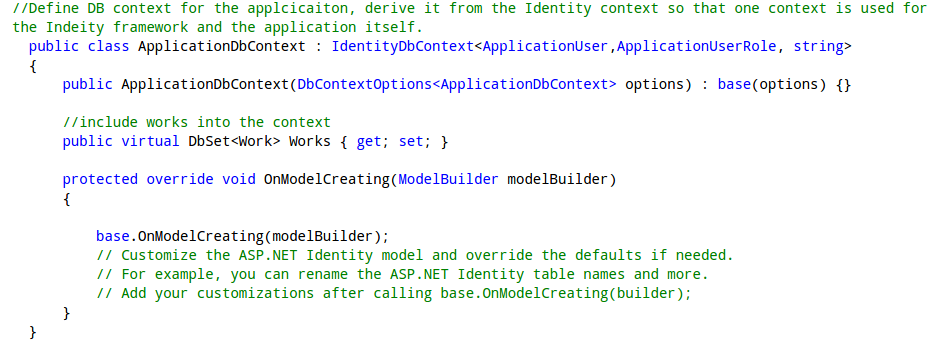
## Authentication and Authorisation

Authentication and Authorisation capabilities within the framework are achieved using the Identify system. It provides a mechanism for creating User, SignIn and Role managers to address the security concerns of a web application. In order to use them, an AplicationUser entity extending the IdentityUser has to create as shown above. The use of the Identity managers is well demonstrated under the Registration action handle in **Sample 5.** Once a post request has been sent to the register, the user data is used to create an Application User instance, which then is passed to the User Manager’s *CreateAsync()* method, which creates a new user in the database. Same principle is used to create a role for the authorization system, which uses the *CreateAsync()* method of the Role Manager. This allows to implement the role based authorization using the *[Authorize(Roles = {role})]* authorization filter. Once roles and users have been created, the user is being added to the role via the *AddRoleAsync()* method and then logged in using the SignIn Manager’s *SingInAsync()* method. Another design decision is demonstrated in this code snippet, which includes using User Claims as session variables, which is a common approach. It enables the authentication cookie to serve as a session cookie and makes user’s related claims information being available from the controller’s context properties, without having to access the database.

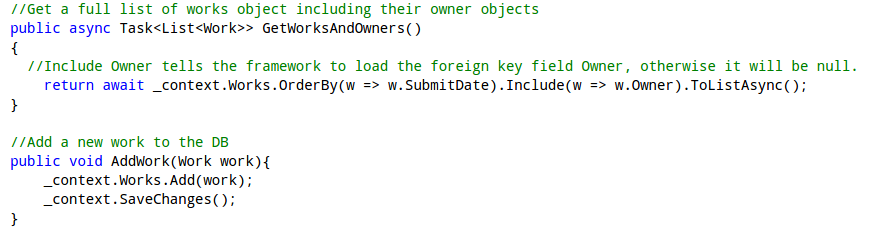
**Sample 5 *–*** Registration action handler, Available under /*src/Controllers/AccountControllers.cs.*

## Entity Framework Core

The core – lightweight and cross platform version of the Entity Framework was used as an Ojbect-Relational mapper to map the .NET model objects to the SQLlite database. SQLlite was used to ensure the cross platform development, as the MS SQLServer is not compatible with the Linux systems. Entity Framework uses a predefined DB context (**Sample 6**) to establish a connection and query or modify the database.

**Sample 6** – Definition of a database context. Available under /*src/Data/* *ApplicationDbContext.cs.*

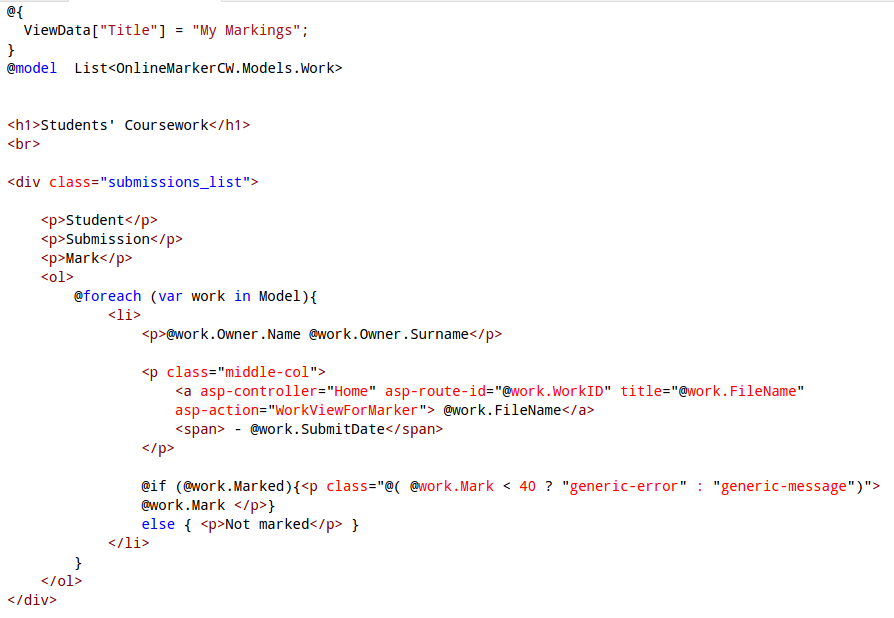
Modifications and Queries are performed via the context on the selected entities, as shown in the **Sample 7** for the Work entity. The ORM mechanism reads the DB and returns a usable .NET object which can be passed to the controller to perform necessary logic operations.



**Sample 7** – Samples of the Database manipulation and queuing using the DBContext. Available under /*src/Services/* *Services.cs.*

## Razor based Templating Engine

Razor based Templating engine provides a simple syntax which consist of Razor markup, C# and HTML for rendering HTML pages. It renders a default layout with a body content which corresponds to the view returned by the controlled, which is inserted into the layout via the *@RenderBody()* method call*.* **Sample 8** shows the view that is rendered for the H*omeController/MyMarkings* route.



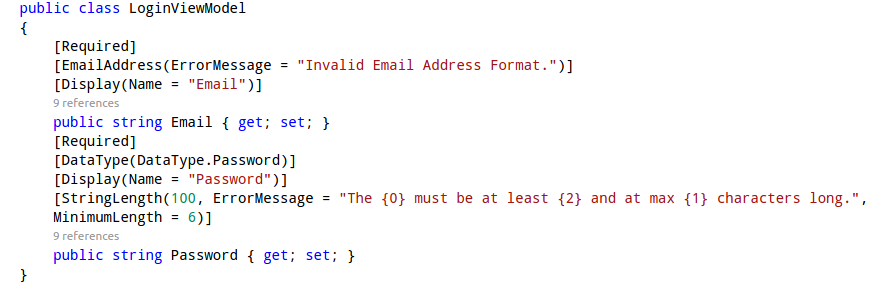
**Sample 8 –** My Markings view, available under */src/Views/Home/MyMarkings.cshtml*

Symbol *@* specifies all Razor specific syntax, followed by C# code, that includes loops and branching statements, etc. – as seen in the sample. Local variables and model can be declared to be used by the view , which are defined at the begging of the it.

A new construct of tag helpers have in introduced under ASP.NET core MVC. Tag helpers replaced big part of the HTML helpers’ functionality, for generating forms and action links, as seen in **Sample 10** of the next section. They use html attribute like syntax, hence easier to read and manipulate to generate the HTML code. For example a pair of *asp-control* and  *asp-action* tag helpers in an anchor element generate a link an action in a controller; *asp-for* tag helper generates an input field for the respective ViewModel property with the necessary validation tags, as discussed in the next section.

## Input Validation

User data input validation on both server and client side can be easily achieved with combination of ViewModel and tag helpers. **Sample 9 shows** definition of a ViewModel, which follows same convention as a Model Enmity definition. It defines the data model that is passed from the HTTP context to the controller under a POST request. The attributes which decorate the properties of the class indicate what validation is to be done on the server side. Many of them are transferred via the tag helpers to the client side.



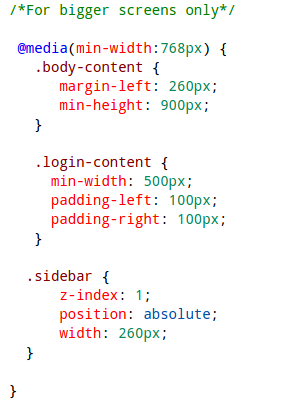
**Sample 9** – Login ViewModel, available under */src/ViewModels/AccountViewModels.cs*

Client side validation attributes such as *min, max length and required* are not generated by the tag helpers hence have to be defined manually as seen in the sample below.

**Sample 10** – Login page View, available under */src/Views/Account/Login.cshtml*

## Professional Layout and Responsiveness

The professional layout, and responsiveness of the application was achieved though combination of the CSS3 techniques and the Bootstrap Framework. The CSS3 *@media* rule was used extensively in order to achieve scalable design. For instance the code sample below shows that only for screen sizes of width of 768px over the sidebar should have certain width, login container should have bigger padding and body content should have bigger height.



**Sample 11** – Layout size parameter definitions for bigger screens, available under */src/wwwroot/css/site.css*

Bootstrap was used to generate the scalable hamburger menu, which incorporated creation of a menu button with attributes *data-toggle="collapse"* and *data-target=".sidebar-collapse"*. These attributes are used by Bootstraps JS library to indicate that the button is used to collapse an element and which particular DOM element to collapse. Combination of *.navbar-collapse* and *.collapse* Bootstrap classes for the sidebar indicate that the side bar should be hidden only after media screen side is smaller than 768px.

Moreover the Hihlight.js JavaScript framework has been used for the HTML code highlighting. Its implementation is highly trivial and involves usage of the HTML *<code></code>* block with predefined language mark-up classes.

# ASP.NET MVC core Evaluation and Conclusions

*This section is going to compare the ASP.NET MVC core framework with Python based frameworks Django.*

## Features Comparison

Both Django and ASP.NET MVC core follow the MVC development pattern, even though a View in Django is more of a call-back function, which is invoked by the router, when a certain route is accessed. Hence in a context of ASP.NET MVC it represents the functionality of a controller. As result Django can be considered a “MTV” framework, which stands for “model”, “template” and “view” .

Compared to Django, ASP.NET MVC core exhibits somewhat less sophisticated model definition capabilities. Django exercises use of predefined fields for the model definition , which can be described as instances of smaller models used as properties of a Model in ASP.NET. For example, in contrast with ASP.NET there exists a predefined File/Image field, which will automatically map the file path, image dimension to the model and will take care of the file streams, which simplifies the development greatly.

Django comes with out of a box ORM, whereas ASP.NET MVC uses the Entity ORM Framework. Both ORMs where designed to work with SQL databases only, hence alternative have to used to work with noSQL. Both ORMs support use of their respective OOP style C# and Python like database expressions for modifying and querying the database, as well as SQL like ones native querying capabilities. For it .NET comes with the Language Integrated Query component, but Django with the RawSQL module .

One aspect where ASP.NET MVC core excels is the user management. ASP.NET MVC core uses the Identity system, which offers more capabilities such as Roles and Claims based authentication, built in third party login and OAuth 2 support. On the other hand, Django comes with an automatically generated administration web interface, which extremely simplifies the management and overview of users.

## Ease of Use, Efficiency and Productivity

ASP.NET MVC core is built on top of c# sharp language, which is a compiled, OOP proprietary language developed by Microsoft for the .NET stack . On the contrary, Django runs on top Python , which is an interpreted, open source, OOP language. Due to its straightforward syntax, lose typing and non compiled nature, Python is more intuitive and easy to use, as result Django applications are faster to write, to get up running, to manually validate and more importantly easier to learn to code. At the same time the less restrictive development approach opens room to creating more bugs and less robust web applications. In Kurt Grandis produces a case study of his two development teams using the both frameworks and concluding that the team using Django was as twice as productive than one using ASP.NET. This closely coincides with the sentiment and experiences of authors of this report.

Moreover ASP.NET and Entity core are rather new technologies and suffer from lack of a proper documentation and developer’s community. The full API reference for both was published only in end of October, early November, hence the development speed of the application suffered in the early stages from trying to guess how to implement certain features. Moreover couple of bugs present in Entity core prevented the model testing to be sufficiently implemented. In comparison Django is mature and has got excellent documentation and hefty community.

## Cross Platform Development

Both frameworks support cross platform development and deployment. They both can be deployed on Linux-based and IIS servers, as both frameworks simply produce web applications which just require the language and libraries compatibility with the relevant system, although it is uncommon to deploy Django application on IIS and the ASP.NET hosting framework became separated from IIS only under the core release.

This project was developed using Windows and Linux systems hand in hand. As results authors run into couple of issues which had influenced the design decisions. The biggest issue was lack of the MS SQL server support for the Linux systems, which lead to use of the rather limited SQLlite database. The Visual Studio IDE, upon which ASP.NET MVC heavily relies, is not available for the Linux systems. That leaves Linux users to purely using CLI and a text editor for development, which may seem to be unappealing for many developers, although Visual Studio like text editor Visual Studio Code is available for Linux. Due to lack of Visual Studio, .NET package management is somewhat frail, due to lack of good a overview outside of Visual Studio of the NuGet packages available and their compatibility with already installed components. In summary, regardless of all the drawbacks, the cross platform development is not impossible and leaves promising impression.

# Bibliography

1. **Fritz, Jeffrey T.** ASP.NET 5 is dead – Introducing ASP.NET Core 1.0 and .NET Core 1.0. [Online] Microsoft, 19 January 2016. [Cited: 2 December 2016.] https://blogs.msdn.microsoft.com/webdev/2016/01/19/asp-net-5-is-dead-introducing-asp-net-core-1-0-and-net-core-1-0/.

2. **Phillip Carter, Zlatko Knezevic.** .NET Core - .NET Goes Cross-Platform with .NET Core. [Online] Microsoft, April 2016. [Cited: 2 December 2016.] https://msdn.microsoft.com/magazine/mt694084.

3. Inversion of Control. [Online] Microsoft. [Cited: 2 December 2015.] https://msdn.microsoft.com/en-us/library/ff921152.aspx.

4. Dependency Injection. [Online] Microsoft. [Cited: 2 December 2016.] https://msdn.microsoft.com/en-us/library/ff921152.aspx.

5. **Steve Smith, Andy Pasic.** Dependency Injection. [Online] Microsoft, 14 October 2016. [Cited: 2 December 2016.] https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection.

6. —. Overview of ASP.NET Core MVC. [Online] 14 October 2016. [Cited: 2 December 2016.] https://docs.microsoft.com/en-us/aspnet/core/mvc/overview.

7. **Sukesh, Marla.** WebForms vs. MVC. [Online] 26 Sep 2014. [Cited: 2 December 2016.] https://www.codeproject.com/articles/528117/webforms-vs-mvc.

8. **Steve Smith, Rick Anderson.** Controllers, Actions, and Action Results. [Online] Microsoft, 14 10 2016. [Cited: 2 12 2016.] https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/actions.

9. **Anderson, Rick.** XSRF/CSRF Prevention in ASP.NET MVC and Web Pages. [Online] Microsoft, 14 March 2013. [Cited: 2 December 2016.] https://www.asp.net/mvc/overview/security/xsrfcsrf-prevention-in-aspnet-mvc-and-web-pages.

10. —. Introduction to Identity. [Online] Microsoft, 14 October 2016. [Cited: 2 December 2016.] https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity.

11. **rawel.** choose between asp identity claims and sessions data. *Stack OwerFlow.* [Online] 8 Feb 2016. [Cited: 2 December 2016.] http://stackoverflow.com/questions/35231107/choose-between-asp-identity-claims-and-sessions-data.

12. **Miller, Rowan.** Entity Framework Core. [Online] 27 10 2016. [Cited: 2 December 2016.] https://docs.microsoft.com/en-us/ef/core/index.

13. **Taylor Mullen, Rick Anderson.** Razor Syntax Reference. [Online] Microsoft, 14 10 2016. [Cited: 2016 December 2016.] https://docs.microsoft.com/en-gb/aspnet/core/mvc/views/razor.

14. **Anderson, Rick.** What are Tag Helpers? [Online] Microsoft, 14 10 2016. [Cited: 2 December 2016.] https://docs.microsoft.com/en-gb/aspnet/core/mvc/views/tag-helpers/intro.

15. **Appel, Rachel.** Model Validation. [Online] Microsoft, 2016 10 14. [Cited: 5 December 2016.] https://docs.microsoft.com/en-gb/aspnet/core/mvc/models/validation.

16. FAQ: General¶. *django.* [Online] [Cited: 5 December 2016.] https://docs.djangoproject.com/en/1.10/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names.

17. Model field reference¶. [Online] django. [Cited: 5 December 2016.] https://docs.djangoproject.com/en/1.10/ref/models/fields/.

18. Query Expressions¶. [Online] django. [Cited: 5 September 2016.] https://docs.djangoproject.com/en/1.10/ref/models/expressions/#django.db.models.expressions.RawSQL.

19. **Don Box, Anders Hejlsberg.** LINQ: .NET Language-Integrated Query. [Online] Microsoft, February 2007. [Cited: 5 December 2016.] https://msdn.microsoft.com/en-us/library/bb308959.aspx.

20. The Django admin site. [Online] django. https://docs.djangoproject.com/en/1.10/ref/contrib/admin/.

21. **Hasan, Noorul.** History of C# Programming. [Online] 27 09 2012. [Cited: 05 December 2016.] http://aboutcsharpprogramming.blogspot.co.uk/2012/09/history-of-c-programming.html.

22. **Grandis, Kurt.** Python + Django vs. C# + ASP.NET: Productivity Showdown. [Online] 24 02 2010. [Cited: 5 December 2016.] http://kurtgrandis.com/blog/2010/02/24/python-django-vs-c-asp-net-productivity-showdown/.

23. **VORAS, IVAN.** Installing Django on IIS: A Step-by-Step Tutorial. [Online] TOPAL. [Cited: 5 December 2016.] https://www.toptal.com/django/installing-django-on-iis-a-step-by-step-tutorial.

24. **Strahl, Rick.** ASP.NET vNext: The Next Generation. [Online] [Cited: 5 December 2016.] http://www.codemag.com/article/1501061.