FILTERING DATA FOR THE SALE OF MACHINE PARTS IN ASP.NET CORE WEB APPLICATIONS USING MVC DEVELOPMENT TECHNOLOGY

UDC: 004.738.12:004.658
Original scientific paper https://doi.org/10.18485/aeletters.2019.4.4.1

Andrea Vujić[1], Djordje Dihovicni[1]*

[1]Technical College, Bulevar Zorana Djindjica 152a, Belgrade, Republic of Serbia

Abstract:
In this paper it is presented how to create a part of a dynamic website that is used to filter data and create a user basket for online purchase of machine products. It is explained how the ASP.Net Core 2.2 MVC technology works. The difference between code first and database first access is shown as well as are different ways to implement a database into a project. This project was built gradually starting from system setup, through the creation of a database that was customized to customer needs, until the layout itself, in which are used markup and styling languages such as HTML5 and CSS.

ARTICLE HISTORY
Received: 13.09.2019.
Accepted: 14.10.2019.
Available: 31.12.2019.

KEYWORDS
MVC, ASP.NET, C#, Web, Database, HTML, CSS

1. INTRODUCTION

MVC (Model - View - Controller) is a high-level architectural layer for creating web applications based on an open source MVC form. The first version of ASP.Net MVC was released in May 2009, evolving until version 6.0 to expand further from May 2016 to the open-source version of ASP.Net Core MVC 1.0, [1]. The latest version of .Net Core 2.2 came out in December 2018. MVC architecture is based on the idea of reusing existing software code, facilitating the development and subsequent maintenance of application software, and splitting it into special components: model (Model), data view (View) and controller (Controller), with the information display component separated from user interactions with those information. MVC architecture consists of specific components in which each is assigned to perform specific functions, [2]. Such a solution is much better than previous one-in-one solutions, which made it difficult to read the source code, make it difficult to find and debug, as well as significantly less functionality and flexibility.

Norwegian computer scientist Trygve Reenskaug first introduced the MVC architecture applied to the Smalltalk-76 during his visit to the Xerox Palo Alto Research Center (PARC) in the 1970s. Then, in the 1980s, Jim Althoff and others used the MVC version of the Smalltalk-80 class library. It was not until a 1988 article that MVC was introduced as a separate term.

Today's classic MVC approach places all logic on the web server, while new technologies (Ajax, AngularJS, AmberJS, JavaScriptMVC, Backbone) allow MVC components to be partially executed on the client side as well [3-4].

CSS (Cascading Style Sheet) is a set of rules that allow the formatting or positioning of elements of a Web page. CSS provides a standard way that describes how a browser will display content on a Web page, [5-7]. The CSS selector defines the element or group of elements to which the rule applies.

Each CSS rule has the same basic syntax that looks like this:

```
selector{
property1: value;
property2: value;
…
}
```

Each CSS rule has a first selector tag followed by a curly bracket. Inside the curly brackets, some CSS

CONTACT: Dj. Dihovični, e-mail: ddihovicni@gmail.com

properties are specified, followed by a certain value, and after that value a semicolon (;).

There are three types of selectors:

Element selector (e.g. h2 () that returns all h2 elements),

A class selector (e.g. .blue () that returns all elements whose class attribute is set to blue) and

Id selector (#id () that returns an element with the given id).

Through CSS, the user can create the style independently, and can also use the finished style sheets, [8-9].

There are three basic ways to add styles to a Web page:

Internal style sheet (page-level formatting) i.e. <style> </style> elements within the header of a web page, adding a link to an external style sheet global formatting and adding inline styles i.e. formatting for a specific html element.

2. MVC ARCHITECTURE

The MVC architecture is divided into three interconnected layers, Model, View, and Controller.

The model processes data received through the Controller, it represents one or more classes with their states that can be displayed at the request of the View or can be modified by the Controller. The folder models is presented at Figure 1.

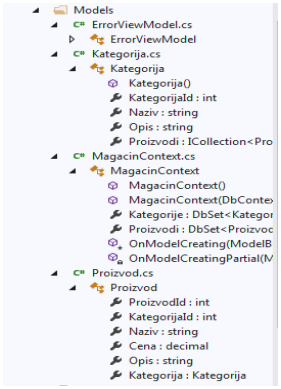

Fig.1. Folder Models

The model contains major program data, such as information about objects from the database [10-12]. It consists of a set of classes that model and support application troubleshooting. It is usually a stable component, lasting as long as the problem itself.

All the business logic of the application is contained in the model. There are several ways to construct a skeletal model. Specifically, a programmer can first construct a model, define classes and attributes within classes, and later, based on the class, form a database. This is called code-first access. All data can be obtained through the model [13-15], but the model cannot be called directly, rather, it is done through a controller, in the form of a request. This request is then processed by the model and returned to the controller with the necessary data. The controller further displays the received data to the end user.

The code for creating one model is shown below:

```
namespace WorkShop.Models
{
[Table (“Product”)]
 public partial class Product
 {
   public int ProductId {get; set; }
   public int CategoryId {get; set; }
   [Required]
   [StringLength (120)]
   public int Name {get; set; }
   [Column (TypeName = “decimal (10.2)”)]
   public int Price {get; set; }
   [StringLength (120)]
   public int Description {get; set; }
      [ForeginKey (“CategoryId”)]
   [InverseProperty ("Products")]
   public virtual Category Category {get; set;}
 }
}
```

Square brackets that are printed above the fields, are called annotations and serve to better describe the field and to add additional values to it, such as the required field [Required] or the number of characters allowed to enter [StringLength (120)].

The controller interprets the user's request and forwards it to Model or View components. The controller interprets the input of the user and passes it to Model or View. It decides how the model should change as a result of user input and which View to use. The folder controllers are shown at Figure 2.

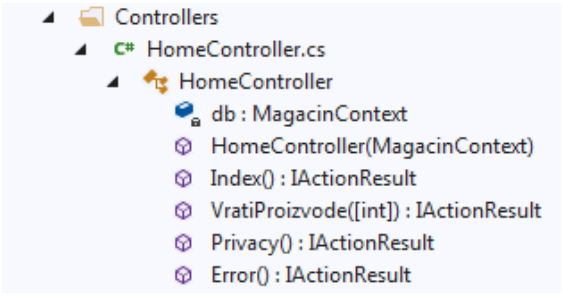

Fig.2. Folder Controllers

The controller represents the mediator between the data view and the model. It contains the main mechanisms for controlling the flow of the program, that is, the behavior of the application itself, and manages user requests. It is the most demanding part of the application programmatically. The controller interprets the user input and passes it to the model or displays it to the user. It contains some of the application logic and has the ability to influence the state of the model, or to decide how the model should be changed, as a result of user requirements, and how the data will be displayed [16-18].

View is responsible for viewing the data according to the changes in the model, as it is described at Figure 3.

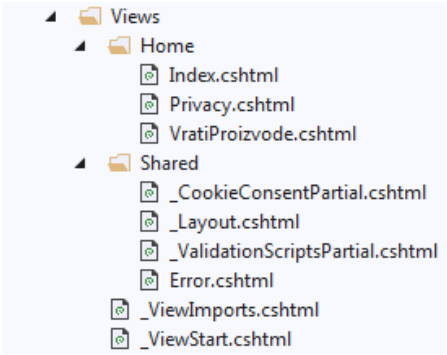

Fig.3. Folder View

View is the last layer of the MVC architecture, containing the application environment, or providing different ways of representing model data through the controller. It is printed in HTML but with the help of Html and tag helpers code can be implemented in C # language, [19-20].

The user can only see what is seen in this component, while the model and controller are usually hidden from the user. The most important feature of this component is its simplicity because it is a visual representation of the model that contains display methods and allows the user to change the data.

In addition to splitting the application into three separate components, the MVC architecture also enables interaction between them:

View - shows the user the state of the model, provides the user interface through which the user enters the data and calls the appropriate operations to be performed on the data.

Controller - listens to and accepts requests from the client to perform an operation, then invokes the operation defined in the model, and if the model changes status, displays it to the user.

Model - represents the state of the system that can be changed by performing operations on objects in the data model. The model has nothing to do with how the data is managed or how the data is presented to the user.

It is important to remember that the display in the user environment and controls are dependent on the model, while the model does not depend on the view or on the component controller, which allows the model to be developed and tested independently of its presentation logic. The web application receives some request or data from the user, the web server recognizes the request and processes the data based on the built-in business logic. The result of the process is a web page that will be displayed to the user. In doing so, the data can be stored in files or in a database.

Using MVC provides the ability to view models in different ways, and make it easier to process and add new or modify existing reports, modify user interaction, and modify business logic.

There are two ways to create .Net Core applications based on Entity Framework, code first and database first access. With code first access, a class model was created and a database was created based on it, while database first access created a database first and based on that database its model class was created through scaffolding.

3. CREATING A .NET CORE APPLICATION

In this example, it will be addressed one of the ways in which data is filtered in .Net Core Web applications, via database first access. To start creating an application, it is important to set up the system based on our needs. The system is configured in the Startup.cs file that contains the services, connections and objects that are required for operation. The first thing to do after setting up the system in the Startup.cs file is to create a system database. Within the Data folder a class named ApplicationUser is created and endowed with properties that are necessary for the registration of new users. There is also one Context class called ApplicationDbContext added to this class and configured to work with the ApplicationUser class. The Startup.cs file modifies the Identity System service so that it also works with the ApplicationUser class and not with its base IdentityUser class. The next thing that is done is creating an identification database through

Scaffolding. This generated a new folder called Areas. Within that folder is the Identity folder and inside it a Pages subfolder that has its own subfolder called Account. The Account subfolder provides logging, registration, and logout views as well as their models. Migrations are used to keep the data that needs to be entered in the database up to date. The following is typed in the Package Manager console:

```
Add Migration Migration Name –Context ApplicationDbContext
```

When updating the migration database, it is indicated which Context class should be updated by adding –Context and the name of the Context class after Add Migration. After the system for user identification was set up, the database for logging through Entity Framework was created and the following script was executed inside:

```
CREATE TABLE Buyer (
BuyerId nvarchar (300) NOT NULL PRIMARY KEY,
Name nvarchar (30) NOT NULL,
Surname nvarchar (30) NOT NULL,
Country nvarchar (30) NOT NULL DEFAULT 'Serbia',
City nvarchar (30) NOT NULL,
Address nvarchar (100) NOT NULL
);
GO

CREATE TABLE Shopping (
ShoppingId int IDENTITY (1,1) PRIMARY KEY,
BuyerId nvarchar (300) NOT NULL FOREIGN KEY REFERENCES Buyer (BuyerId),
DateBuy datetime NOT NULL DEFAULT GETDATE ()
);
GO

CREATE TABLE CategoriesMachine Parts (
CategoriesId int IDENTITY (1,1) PRIMARY KEY,
Name nvarchar (100) NOT NULL
);
GO

SET IDENTITY_INSERT ID ON
INSERT INTO CategoriesMachine Parts (CategoriesId, Name) VALUES (1, 'Pumps')
INSERT INTO CategoriesMachine Parts (CategoriesId, Name) VALUES (2, 'Valves')
INSERT INTO CategoriesMachine Parts (CategoriesId, Name) VALUES (3, 'Actuators')
INSERT INTO CategoriesMachine Parts (CategoriesId, Name) VALUES (4, 'Reducers')
INSERT INTO CategoriesMachine Parts (CategoriesId, Name) VALUES (5, 'Manifolds')
SET IDENTITY_INSERT CategoriesMachine Parts OFF
GO

CREATE TABLE MachinePartsOffer (
PartsOfferId int IDENTITY (1,1) PRIMARY KEY,
CategoriesId int FOREIGN KEY REFERENCES CategoriesMachineParts (CategoriesId) NOT NULL,
Manufacturer nvarchar (50) NOT NULL,
Name nvarchar (20) NOT NULL,
Type int NOT NULL,
Model nvarchar (20) NOT NULL,
Price decimal (10,2) NOT NULL,
Description nvarchar (900) NULL
);
GO

INSERT INTO OfferMachine Parts (CategoriesId, Price, Manufacturer, Name, Type, Model, Description) VALUES (1, CAST (1200 AS Decimal (10,2)), 'DAVID BROWN', 'Pump', 'Gear Pump', 'With stand mounting, with flanges together with oil tank ',' DAVID BROWN gear pumps have found wide application in our market. Robust, compact, easy to install and minimal maintenance are the reasons why they are used in all areas of the industry. ')

INSERT INTO OfferMachine Parts (CategoriesId, Price, Manufacturer, Name, Type, Model, Description) VALUES (2, CAST (800 AS Decimal (10,2)), 'DAVID BROWN', 'Valve', 'TYPE 3022', 'Electromagnetic , ball, butterfly, latches, shut-off ',' NPT thread 0d 1/4 "to 4", full ball opening, chrome-plated brass body, PTFE seats, thread F / F ends, maximum temperature 180 ° C, PN 25, manually operated, steel handle ')

INSERT INTO OfferMachine Parts (CategoriesId, Price, Manufacturer, Name, Type, Model, Description) VALUES (3, CAST (1500 AS Decimal (10,2)), 'DAVID BROWN', 'Actuator', 'SAR 07.1-16.1', 'II - 409 Beck rotary electronic control drive', 'Actuators are supplied from the simplest OPEN - CLOSE modes, with high protection, with explosion protection to allow safe process automation to the most complex modes with full rotation or partial rotation. ')
```

INSERT INTO Machine Parts Offer (CategoriesId, Price, Manufacturer, Name, Type, Model, Description) VALUES (4, CAST (1600 AS Decimal (10,2)), 'FALK', 'Reducer', 'Ultramax Gear Unit', '2060FC2A Ratio i = 2.827 ',' Gearboxes are mechanical power transmissions that are used to transfer power from the drive machine to the machine and adjust the speed at the drive shaft to the required speed at the machine shaft. ')

INSERT INTO Machine Parts Offer (CategoriesId, Price, Manufacturer, Name, Type, Model, Description) VALUES (5, CAST (1200 AS Decimal (10,2)), 'NORGREN', 'Manifold', '5/2 and 5/3 electromagnetic and pneumatic actuated valve ',' Reciprocating, rotary, plate, valve ',' Switch the mechanical structure composed of a solid mechanical elements. they are used to start, stop and direction of energy i.e., the flow of the compressed air in the tires or the flow of oil under pressure in the hydraulics The manifold is practically the executive body of the control system and performs the commands created by the control logic. It should be emphasized that the manifold valve is not designed to perform control tasks, therefore it cannot change the intensity of pressure or flow. The principle of operation of hydraulic and pneumatic manifolds is almost identical, and the symbols are practically the same except that the connections are marked with other gachies. ')

```
CREATE TABLE Basket
(
ID int IDENTITY (1,1) NOT NULL PRIMARY KEY,
ShoppingId int NOT NULL FOREIGN KEY REFERENCES ShoppingId,
ID int NOT NULL FOREIGN KEY REFERENCES CategoriesMachine Parts (ID),
Quantity int NOT NULL
);
GO
```

This script provides the tables necessary to connect customers to the basket that will store the items the customer wants to buy. The script contains a table listing the categories of machine parts, a table listing the parts themselves on offer, a table relating to the customer and his order, and a basket table representing the header of the invoice. It is a table that associates a foreign key with the customer table and the shopping table and adds to it only the quantity of products purchased. The data in the customer table is automatically entered by logging the user to the site.

Further within the Register.cshtml.cs class, the Post registration method is modified and the application's DbContext is accessed. Data is taken from the ApplicationUser object to create a user class object. From the AspNet.Users system table that takes care of users and is created with the Entity Framework, the data is copied to the user table. The central database table is a basket table because a basket consists of multiple items, and each item belongs to a specific order. The basket table specifies the product that was purchased and the quantity of product purchased. It is characteristically that a single product cannot appear repeatedly as an order item. When data is entered into the database, an order is first created, and an order is created by inserting the customer ID to which the order relates. The order date is automatically entered as well as the order ID, so that the items selected by the user can be tracked and these items are further inserted into the basket table. Communication with the database takes place as soon as the contents of the consumer basket are fully defined.

Within the Models folder a class named BasketItem is created, this is a class that has an automatic property Product - type product and automatic property Quantity - type quantity, those are the basket items, the left column is a product and contains the product name, price, quantity of product purchased, etc.

Consumer basket data cannot be stored inside the controller because each new request creates a new controller and thus the old data will be lost. This is why this data is stored within a user session. A session is one object that is stored in server memory that has its keys and values. Keys and values stored inside a session are strings. In this case, not only strings can be used to preserve the entire consumer basket and therefore extensions are used. In SolutionExplorer, a folder called Extensions was created and a new class was created inside it called SessionExtensions. An extension class is a static class and contains all static methods. As for the session itself, it can only be accessed inside the controller. The session must be registered as a service and must also be registered within the Middleware component. If a session is accessed from a class other than the controller class, two new accessors are added to the services, IHttpContextAccessor and HttpContextAccessor.

A static class that contains two static methods called SessionExtensions has been created. A method has been created for serialization and for deserialization of data. The goal of these methods is to keep the entire consumer basket under lock and key by using the JsonConvert class. With the serialization method, the string is stored under a key and the entire basket class object is serialized, and the basket class object contains everything that the user has purchased while visiting the site. The deserialization method works the other way around, it contains the session variable and the JsonString assigned to that session variable. A key-based method approaches that session variable that represents the contents of the basket, performs deserialization, and creates a basket class object. A Basket class can now be created. This class serves to provide information with the consumer basket. The basket saves the data in the form of a generic list of type BasketItem. The basket class has as its field a generic list of basket item types and now additional methods can be inserted into it. An AddItem method has been created that has the product and the quantity of that product as input parameters, this method checks if a product with a given ID is already in the basket, if not, if the item is the same as NULL, a new item is created by specifying a specific product and quantity per product purchased, and that item is added to the ListItems object. If the item is not NULL it means that it is already in the basket and only the quantity is further incremented. Next, a method has been created to manipulate the list. The DeleteItem method allows an item to be deleted from a list. To do this, it is enough to just pass the ID of that particular product. An item from the generic list whose product ID is the same as the given ID is found and then the Remove method is called and this item is ejected from memory. The ChangeItem method specifies a product and specifies a new quantity for that product. First, it was checked to see if that product was found as a basket item. The BasketValue method allows the user to know the value of the basket at any time. The code block goes through a generic list that stores items and for each member of the generic list accesses the product. The item as its property contains the product from which its price is taken, and from the item itself quantity. When the quantity is multiplied by the price, the value of the item is obtained. The DeleteBasket method, as its name implies, gives the ability to delete all data from the recycle bin i.e. to wipe the basket. This is accomplished by deleting the entire collection of items:

```
list Items.Clear ();
```

When the creation of the Basket class was completed, a new folder called Services was created in SolutionExplorer, where the BasketServis class was further created. The task of this class is to search for a session variable and to create an object of the basket class on the basis of it, or vice versa, to store the corresponding value in the session variable based on the object of the class basket. An IhttpContextAccessor accessor is defined within the BasketService class to allow access to a session variable outside the controller. This accessor accesses the session, when accessing the session the key under which the basket is serialized can be read. In order to read the current contents of the basket, or vice versa, the contents of the basket are serialized within the basket. The SaveBasket method accesses the session in which the SerializeBasket method is located. The contents of the basket are the product and the quantity of the product, the entire contents of the basket class are stored within the session variable as a key. The ReadBasket method works the other way around, it reads a session variable.mJsonString was added to the session variable, based on the JsonString and the DeserializeBasket method, an object of the basket class is created, which is what our service does. This service is defined as a scoped service because it is session related. For this reason, it is also defined in the Startup class. The role of this service is to serialize a basket within a session variable or deserialize that session variable into a basket class object.

The next item on the list is to create a controller that works with the basket, called the BasketController. It has the same methods as the Basket class, however, every time we insert a new product into the cart, we invite BasketServis to store that information within the session variable. That session variable is named the same as the key. The value of the basket is the contents of the basket serialized in Json, which has a DbContext and a service that is inserted through a DI container. The basket field fills the basket service as follows

```
BasketService.ReadBasket ();
```

This means in translation: when a controller, a basket controller is instantiated, it checks if there

is stored data for an existing basket, if any, it will create a basket class object, and if not, it will create an empty basket class object.

In this case, within the Index Action method, there is only an Index Action to which the basket object is passed. As the customer purchases, he adds one product at a time to the cart. The product is added to the cart by calling the AddItem method. The AddItem method obtains a product ID using a hidden field, if a product already exists, it is first added to the basket object that was created by deserializing the session variable. In order for this data to be stored and made available for later, a basket service is called which re-serializes that data within the session variable. In this case, it always returns to the Index page because the task of this controller is to display the contents of the basket, to give a user interface that will allow the product to be thrown out of the basket, and to display a user interface that will allow the quantity of products in the basket to be changed. The user can see the contents of the basket and possibly later buy that content. The basket view index as a model directive has the Basket model, which has a generic list containing items and methods for working with the basket. In Index view via HTML, a chart is drawn by looking at the contents of the basket. The Foreach loop passes through the basket items from the model.

The next step is creating a shopping controller that is called by clicking the Buy button. It's called the BasketController. Creating this controller requires the DbContext class StoreContext. Now it is needed to move data from a session to a database, a UserManager is needed to read which user wants to buy products, as well as a basket service (instantiated as kService) that will convert the session to a basket class object at any time. The constructor requires inserting a DbContext object (_db), inserting a UserManager (_um), and instantiating a service (_kService). The index method is authorized by an appropriate annotation. This means that no purchase can be made until the user logs in to the system. Index method first calls the method ReadBasket, this method looks at the session variable and tells whether or not there is data inside the session variable. When it looks for a session variable, it creates a basket class object. If the number of items in the basket is 0, it means that the basket is empty, the system does not communicate with the database but calls the index action of the HomeController. Code line

```
ApplicationUser user =
await um.GetUserAsync ( User);
```

serves to access the logged in user. Since this method is authorized, it cannot be invoked unless the user is logged in. If the user is satisfied with the contents of the basket and wants to make a purchase, clicking the Buy button first opens the Index action of the HomeController, which asks the user to sign up for the service. When typing in the data and logging in, a view of the PurchaseController is displayed. First, the user who wants to buy the selected products is read. Throuhg the UserManager the user ID required for the data to be entered in the Order table is found. After that, the Order class is instantiated by using the logged-in user ID as the customer ID, and the purchase date is the date when the Buy button was pressed, the moment when the user is completely satisfied with the contents of the basket. Next, the order that is the invoice header is instantiated. It consists of items. The order object is inserted as follows:

```
db.Orders.Add ( o1);
```

The SaveChanges () method is then called. This method records the data in the order table. An order ID is taken from that table to insert order related items. Items are accessed by going through the foreach loop through all the items in the basket. A basket object is created using a service basket and has items as its property. Since the Items table is already in the database, an instance of the Item class is created based on the items in the basket. The item contains the order ID, product ID, and quantity. The order ID is the order that was just inserted. As the items go through the foreach loop, it is concluded that the entry in the Item table is repeated as many times as the items are entered into the basket. The basket as a whole is deleted using the service basket with the code:

```
kServis.DeleteBasket ( );
```

this frees up the session variable and ends the order.

4. CONCLUSIONS

There are many advantages to creating web applications using MVC architecture and some of them are easier adding of new data views, creating models in multiple ways, easy interaction with the

user, the ability to collaborate on projects with multiple developers, the ability to reuse code, etc. The disadvantages of MVC architecture are that it is too complex to be used in the development of smaller applications, which leads to a deterioration in both its design and its features, as well as frequent modifications of the model that cause data presentation to remain overburdened by changes in requirements, which may delay the response on demand.

REFERENCES

[1] M. Abdul, R. Abdisam, MVC Architecture: A Detailed Insight to the Modern Web Applications Development. *Journal of Solar and Photo energy Systems*, 1 (1), 2018: 1-7.

[2] K. Czarnecki, M. Antkiewicz, Mapping Features to Models: A Template Approach Based on Superimposed Variants. *International Conference on Generative Programming and Component Engineering*, Vol.3676, 2005, pp.422-437. https://doi.org/10.1007/11561347_28

[3] A. Ašonja, D. Mikić, E. Desnica, Ž. Adamović, Justifiability of Execution of Serbian Teleservice in Industry. *Acta Technica Corvininesis - Bulletin of Engineering*, 8 (3), 2015: 77-79.

[4] D. Tamal, A Comparative Analysis on Modelling and Implementing with MVC Arhitecture, *International Journal of Computer Applications*, (3), 2011: pp 44-49.

[5] Le Blanc Patrick, Microsoft SQL Server 2012. *O'Reilly Media Inc.*, California, 2013.

[6] H. Monga, S. Baghla., Approach to security in wireless sensors networks. *Applied Engineering Letters*, 1, (3), 2016: 80-84.

[7] P. Nielsen, Microsoft SQL Server 2008 Bible. *Wiley Publishing, Inc.,* Indianapolis, USA, 2008.

[8] Dj. Dihovicni, M. Medenica, Database linear of scalability and high availability while maintaining a system performance, 10th International Scientific Conference "Science and Higher Education in Function of Sustainable Development", Uzice, Serbia, 2017, Section 2, pp.45-53.

[9] Dj. Dihovicni, M. Medenica, Fuzzy support model for long pipelines by using DB2 approach. *Applied Engineering Letters*, 2 (2): 2017: 76-83.

[10] S. Raman, H. Baghla, Monga, Method & Implementation of Data Flow. *Applied Engineering Letters*, 1, (4), 2016: 105-110.

[11] K. Kevin, Cryptography in the Database: Last Line of Defense. *Addison-Wesley*, 2005.

[12] D. Pleskonjić, N. Maček, B. Đorđević, M. Carić, Sigurnost računarskih sistema i mreža. *Mikro knjiga*, Beograd, 2007.

[13] A. Veljović, N. Gojgić, Projektovanje baza podataka. *VŠTSS*, Čačak, 2006.

[14] L. Ramkilde Knudsen, New Potentially 'Weak' Keys for DES and LOK. *Springer Link*, 1995.

[15] K. Kenan, Cryptography in the Database. *Symantec Press*, 2006.

[16] N. Galbreath, Cryptography for Internet and Database Applications. *Wiley Publishing, Inc.* 2002.

[17] N. Yuhanna, The Forrester Wave: Database Encryption Solutions, Q3 2005. *Forrester*, 2005.

[18] J. Black, P. Rogaway, Ciphers with Arbitrary Finite Domains, RS Data Security Conference, Cryptographer's Track, Lecture Notes in Computer Science. *Springer*, 2002.

[19] M. Medenica, Dj. Dihovicni, Security Point of View of ASP.NET Application. *13th International Conference on Advanced Technologies Systems and Services in Telecommunications - TELSIKS 2017*, IEEE Conference, Nis, Serbia, 2017, pp.403-406.

[20] Dj. Dihovicni, M. Medenica, Development of software package named "Step method" for robust time delay systems". *International Journal of Latest Research in Engineering and Technology*, 3 (8), 2017: 36-43.