Web-based Reporting

Contents

Introduction

Enterprise solutions like Adroit Smart SCADA are able to generate vast amounts of real-time as well as historical data that is often stored in SQL databases. In its raw form, as large numbers of rows and columns structured into tabular datasets, the true meaning and value of all this data is often impossible for the average user to assimilate. Hence a requirement that is common to most organisations: a web-application that can be hosted on the company Intranet/Internet server offering secure yet easy and meaningful user access to this data in graphical and/or text formats.

Most organisations have unique aspects to their underlying business model, which translate into varying requirements and priorities as to how all this data should be interpreted and visualised. This means that a one-size-fits-all reporting solution simply cannot be provided as part of some shrink-wrapped, off-the-shelf software offering.

So, over the years, Adroit Technologies have been commissioned by a variety of organisations to produce custom, web-based reporting solutions out of which a common approach to doing this has emerged.

This data sheet endeavours to provide an introductory overview of the fundamental approach taken and ends off by showing some screenshots of the different report screens that have been produced. You may prefer to read it online at http://adroit-europe.com/WebReporting since the screenshots render more precisely, and the content is updated from time to time.

Model View Controller Architecture

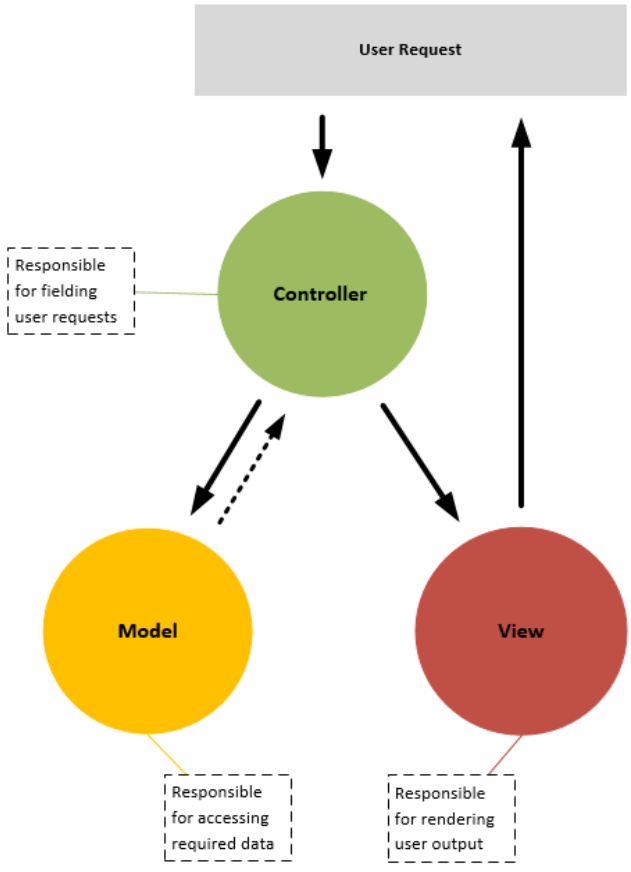


Figure 1 - Model View Controller Architecture

Figure 1 - Model View Controller Architecture, alongside, illustrates the very robust, and well-proven three-tier MVC architecture that is used.

User requests in the form of URLs sent from a web-browser are fielded by the Controller component of our MVC web application. Each distinct URL of the form …/Controller/Action is implemented as an Action method in the relevant Controller class. For example, URL …/AmHistoricalIncidents/Index is implemented by the Index method of the controller AmHistoricalIncidents (prefix Am refers to Alarm Management).

Generally, an incoming request requires some kind of SQL query to be carried out on the data in the *Model.* This latter has usually been *scaffolded* in Visual Studio IDE from SQL database(s) containing raw data gathered by, for example, an Adroit SCADA system.

For most cases, SQL query times are not a performance turnaround issue, and we make use of a technology known as LINQ - Language Integrated Query. This is a .NET Framework

component that adds native data querying capabilities to programming languages like C# which is the language we use to build these web applications.

Where SQL query times are an issue, for example when querying massively large datasets, the *Controller* executes native SQL queries on the underlying model data instead of the much more simple and flexible LINQ queries.

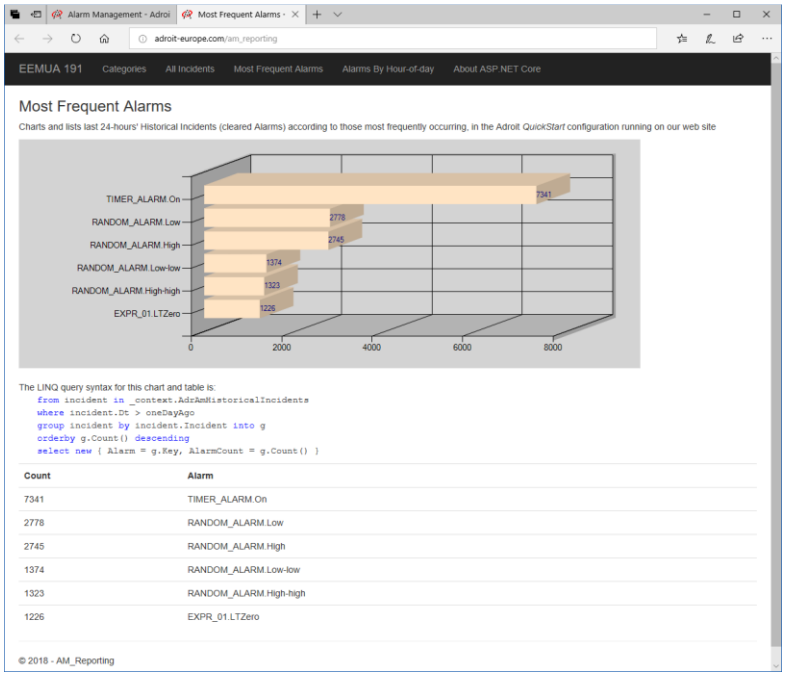
More often than not, data returned by the *Model* in response to LINQ or native SQL queries (dotted arrow in Figure 1 - Model View Controller Architecture above) is transformed by the *Controller* into some kind of *ViewModel* before being passed to the *View* component of our MVC web application.

There is generally a one-to-one correspondence between MVC controller *Actions* and *Views* implemented in the *View* component, in that *Action* methods call *View* methods of the same name, passing the *ViewModel* constructed from the data returned by the *Model*. In our Alarm Management example above, the *Index Controller* method ends by calling the *Index View* method, passing it a list of most frequently occurring alarms.

Each *View* method is implemented as a page of HTML, specifically a file with *.cshtml* extension since we make use of ASP.NET *Razor* syntax, meaning that C# code can be embedded in the page. This is in order to be able to carry out any required processing logic on the *ViewModel* data that has been passed to the page, before returning native HTML back to the web-browser that initiated the original URL request.

As well as providing a systematic, logical way of factoring the web application into different functional areas, the fact that *Views* are totally separate from the rest of the application means that the look and feel of *View* pages can easily and flexibly be modified without in any way affecting the underlying compiled core base of the application. One very pertinent example of why this is such a good thing is that *View* pages can very easily be translated into non-English languages by end users without having any impact on the rest of the application which is shipped as compiled DLL or EXE file(s). Some of the screenshots shown later are from *Views* that have been translated into Serbian by Serbian end-users.

Screenshots



As a first example, this is the page that is rendered by the controller action referred to earlier, i.e. the *AmHistoricalIncidents Index* action produces a horizontal bar-chart display of most frequently occurring alarms sorted in descending order of occurrence frequency.

In the middle of the page, and purely for informational purposes we show the LINQ syntax used to achieve the query the page is based on.

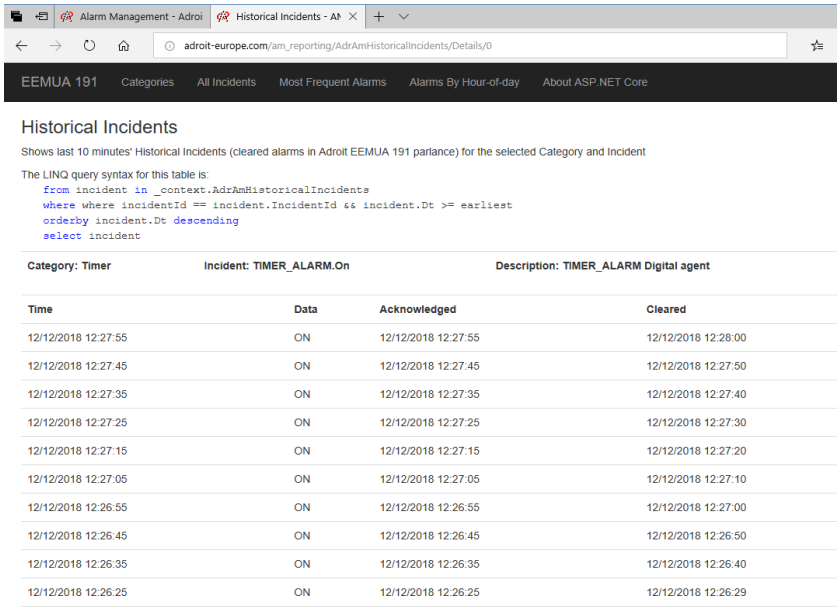
At the bottom of the page, the results of the query are shown in tabular form.

Another Alarm Management example.

This is the page rendered by *AmHistoricalIncidents* controller *AlarmsByHour* action and shows which hours of day are responsible for producing the most alarms.

The current hour is shown as a vertical bar in a different colour.

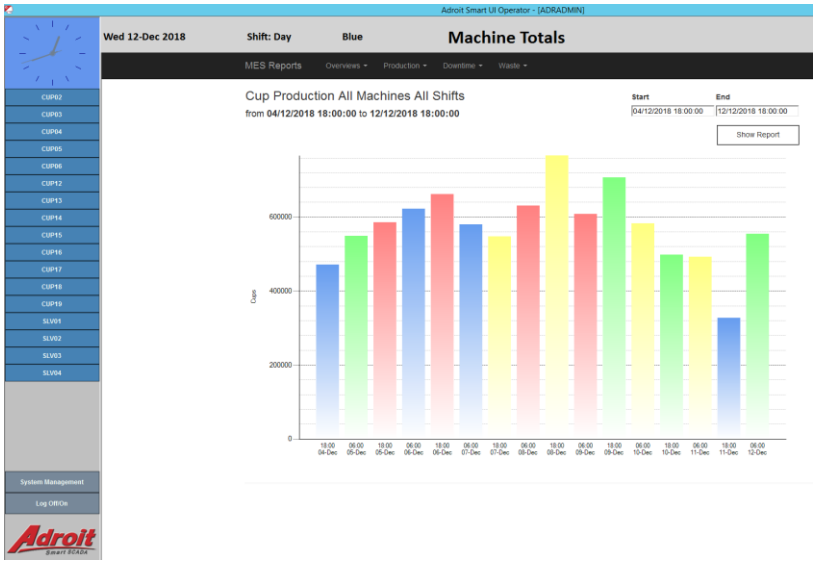
Once again the LINQ syntax appears below the vertical column-chart.



A final Alarm Management example is a tabular display of Historical Incidents (EEMUA191 parlance for acknowledged alarms) for a selected category and incident.

This time the LINQ syntax is shown above the query results.

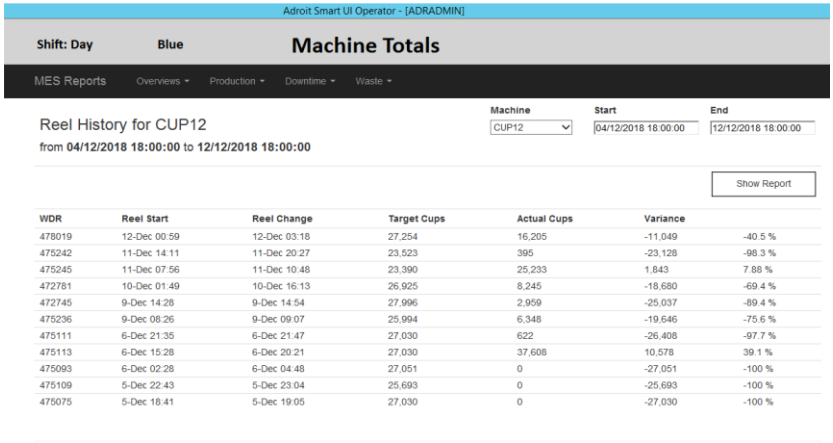
The controller action to render this page is *AmHistoricalIncidents* action *Details/IncidentId*.



Here's an example of a web page rendered inside a Web Browser control hosted inside the Adroit Smart UI Operator application.

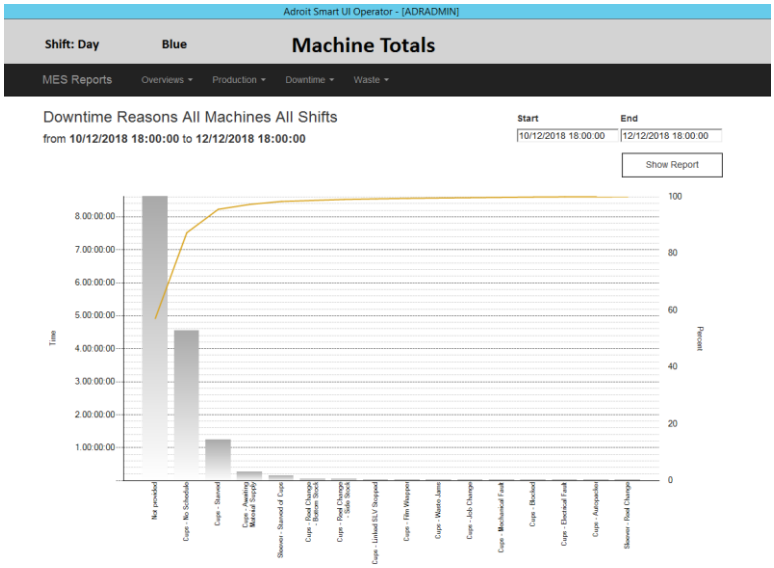
In this case we are showing simulated cup production over the last 8-day shift cycle, with the colour of vertical bars corresponding to shift colour (this factory has four shifts designated Red, Blue, Green, and Yellow)

The controller in this case is *ShiftTotals* and the action is *ProductionAllAshifts*.



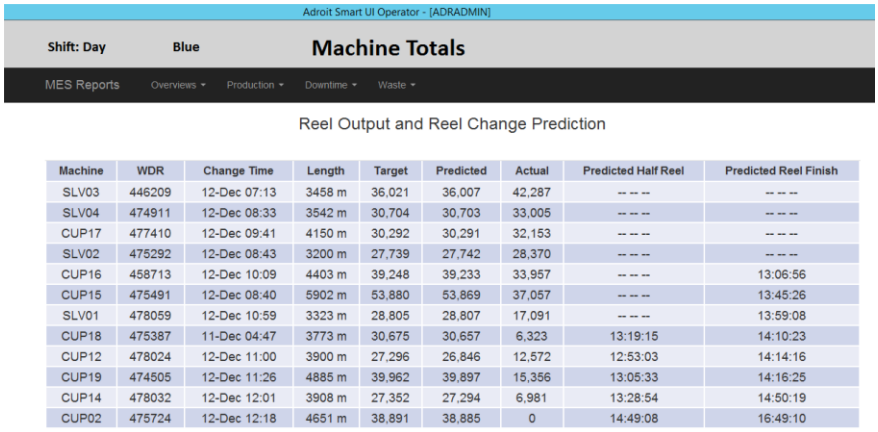
Another example from the paper cup factory simulation, this time a tabular display showing performance over a selected time for a given machine in terms of actual versus target production.

The controller is *ShiftTotals* and the action is *ReelHistory* in this case.



Another quite interesting graphic from our simulated cup factory, this time showing reasons for machine downtimes. There is also a Pareto curve enabling easy application of the 80-20 rule as to downtime causes.

The controller is *ShiftTotals* and the action is *DowntimeReasons* in this case.



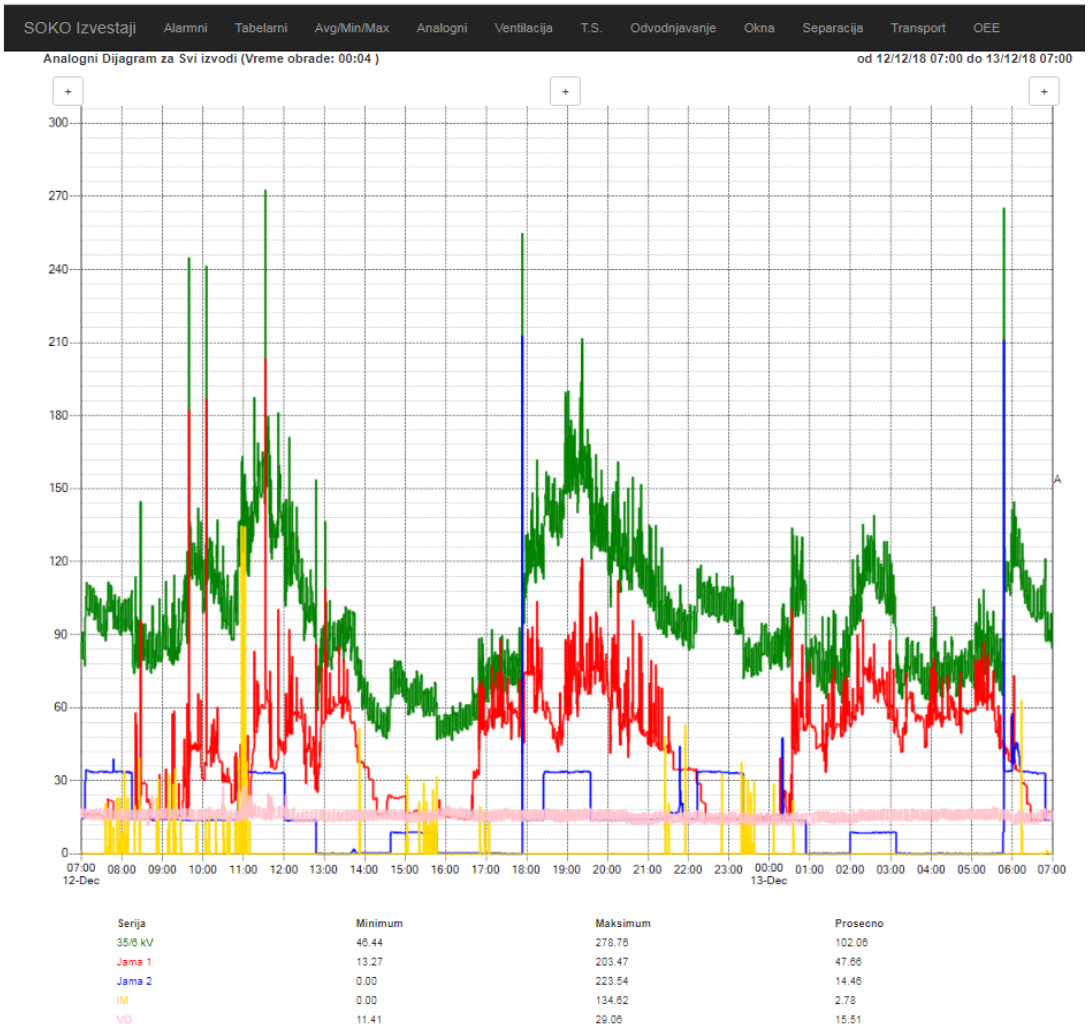
Machine	WDR	Change Time	Length	Target	Predicted	Actual	Predicted Half Reel	Predicted Reel Finish
SLV03	446209	12-Dec 07:13	3458 m	36,021	36,007	42,287	-- -- --	-- -- --
SLV04	474911	12-Dec 08:33	3542 m	30,704	30,703	33,005	-- -- --	-- -- --
CUP17	477410	12-Dec 09:41	4150 m	30,292	30,291	32,153	-- -- --	-- -- --
SLV02	475292	12-Dec 08:43	3200 m	27,739	27,742	28,370	-- -- --	-- -- --
CUP16	458713	12-Dec 10:09	4403 m	39,248	39,233	33,957	-- -- --	13:06:56
CUP15	475491	12-Dec 08:40	5902 m	53,880	53,869	37,057	-- -- --	13:45:26
SLV01	478059	12-Dec 10:59	3323 m	28,805	28,807	17,091	-- -- --	13:59:08
CUP18	475387	11-Dec 04:47	3773 m	30,675	30,657	6,323	13:19:15	14:10:23
CUP12	478024	12-Dec 11:00	3900 m	27,296	26,846	12,572	12:53:03	14:14:16
CUP19	474505	12-Dec 11:26	4885 m	39,962	39,897	15,356	13:05:33	14:16:25
CUP14	478032	12-Dec 12:01	3908 m	27,352	27,294	6,981	13:28:54	14:50:19
CUP02	475724	12-Dec 12:18	4651 m	38,891	38,885	0	14:49:08	16:49:10

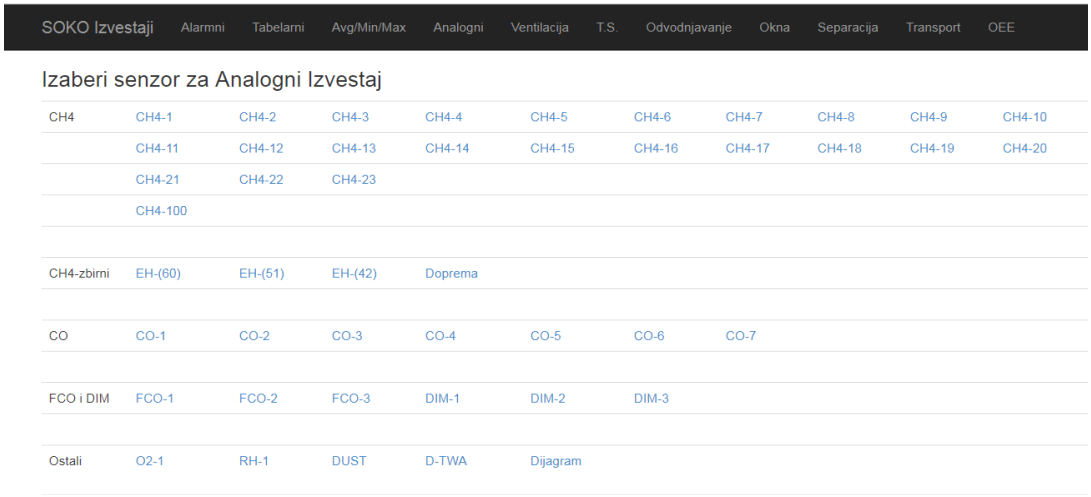
A final page from the cup factory simulation. This one shows predicted reel completion times and production targets in a tabular display with alternate lines in contrasting colours.

The controller is *ShiftTotals* and the action is *ReelDynamics* in this case.

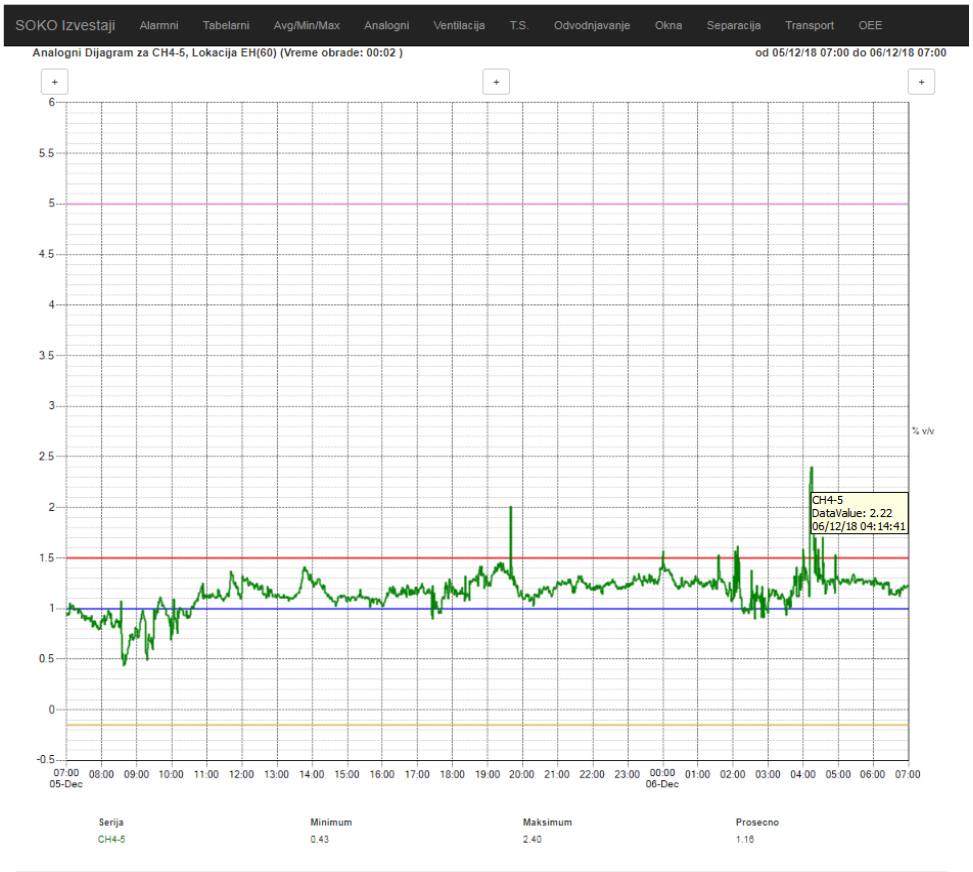
Below is a page showing various parameters over a 24-hour period from a coal mine in Serbia. Note that the entire web-page including navigation bar, data legend, etc. are all in Serbian not English.

The controller in this case is *DL_* (Data logging controller) and the action is *EditDiagram1?....* The question mark following the action name is a long parameter string.





The page shown above is the *Index* action for the *_DL* controller and is a good example of how users can create their own report navigation framework. Each of the blue *ActionLinks* shown on the page above represents another parameterized URL report link into the application. In this way users are able to implement their own report navigation strategy in their own language, and with a layout that makes operational sense to them.



Another graph from the Serbian coal mine. This example shows that tool-tips can be displayed on the web page when hovering over a data point in order to provide more detail.

Some End-user Testimonials

Ranko Radoja, Master of Mining Engineering and Director of Soko Mine, Serbia: *"Every working day starts out with a cup of coffee and Soko Reports (The mine works 24h in three shifts, the most important part to me is what happened in the second and third shift)."*

Vladimir Cvetkovic, Master of Mining Engineering and Soko Mine Manager: *"I use Soko Reports application via my tablet, mobile phone and desktop PC, regularly during non-work hours, so I can see exactly what is happening in the pit. All the necessary reports have been created, I can literally see all the technical aspects and all the parameters that I need."*

Dusan V. Pokrajac, Master of Electronics Engineering and main electronics engineer: *"The simple installation of the application, the stability and quick response time, the easy creation of reports, the monitoring of who accessed the application, the compatibility with all web browsers: Firefox, Chrome, Opera... etc., and accessibility from all platforms (Windows, Android, iOS). These features make the application great."*